

University of Groningen

On the applicability of requirements determination methods

Bollen, Petrus Wilhelmus Laurentius

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version

Publisher's PDF, also known as Version of record

Publication date:

2004

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):

Bollen, P. W. L. (2004). *On the applicability of requirements determination methods*. s.n.

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

On the applicability of requirements determination methods

P.W.L. Bollen

RIJKSUNIVERSITEIT GRONINGEN

On the applicability of requirements determination methods

Proefschrift

ter verkrijging van het doctoraat in de
Bedrijfskunde
aan de Rijksuniversiteit Groningen
op gezag van de
Rector Magnificus, dr. F. Zwarts,
in het openbaar te verdedigen op
donderdag 9 december 2004
om 13.15 uur

door

Petrus Wilhelmus Laurentius Bollen
geboren op 12 juli 1959
te Maastricht

Promotor:

Prof. Dr. Ir. J.L. Simons

Beoordelingscommissie:

Prof. Dr. S. Brinkkemper

Prof. Dr. R. Wieringa

Prof. Dr. Ir. H. Wortmann

ISBN: 90-5681-211-4

Published and distributed by

P.W.L. Bollen
Pergamyndonk 210
6281 GZ Maastricht
the Netherlands

CIP-DATA KONINKLIJKE BIBLIOTHEEK

Bollen, Petrus Wilhelmus Laurentius

On the applicability of requirements determination methods.

Thesis Rijksuniversiteit Groningen

With Summary in Dutch.

ISBN: 90-5681-211-4

NUR 983, 992

Subject headings: Requirements determination, requirements specification, Natural
Language Modeling

© 2004, P.W.L. Bollen, Maastricht

All rights reserved. No part of this publication may be reprinted or utilized in any form or by any electronic, mechanical or other means, now known or hereafter invented, including photocopying and recording or in any information storage or retrieval system, without prior written permission from the copyright owner.

*To Anjie
and
Trisha*

ACKNOWLEDGMENTS

I would like to thank all the people that have made the finalisation of this thesis possible. I would especially thank John Simons, for being an inspiring Ph. D. supervisor. Furthermore, I would like to thank the members of the dissertation committee: Sjaak Brinkkemper, Roel Wieringa and Hans Wortmann for their constructive criticism, which has helped to improve the quality of this thesis.

Furthermore, I would like to thank the former members of the former department of Information Science at the faculty of Economics and Business Administration at Maastricht University, who inspired me to write this thesis. A special word of thanks is for Shir Nijssen who showed me, how to be persistent in carrying on research, even in an unfavourable academic climate.

Another word of thanks is to my former colleague in the department of Logistics and Strategy at Maastricht University: Geert Duysters. Thanks Geert for your efforts in helping me find a suitable academic environment for finalizing this thesis.

The research that is contained in this thesis, was started 11 years ago. In this period many changes have taken place in my personal life. I want to dedicate this thesis to my wife Anjie, who has helped me in achieving my goals with her love and understanding and to our daughter Trisha, who has enriched our lives and has positively effected the conditions under which this thesis has been finalized.

Peter Bollen

October, 2004

CONTENTS OF THE THESIS

| | |
|---|-----------|
| Chapter 1: Introduction | 1 |
| 1.1 Subject of study: requirements determination | 1 |
| 1.1.1 Requirements determination: Translating domain requirements into model requirements | 1 |
| 1.1.2 Requirements determination: field of study | 2 |
| 1.1.3 Organization of chapter 1 | 3 |
| 1.2 History of requirements determination | 3 |
| 1.3 Requirements determination and information systems development | 5 |
| 1.3.1 Roles in requirements determination | 5 |
| 1.3.2 The way of modeling, way of working and way of controlling in requirements determination | 6 |
| 1.3.3 Substeps in requirements determination | 7 |
| 1.3.4 Eras in requirements determination | 9 |
| 1.3.5 Conclusions on requirements determination as field of study | 10 |
| 1.4 Research goal | 11 |
| 1.5 Research approach and outline of the thesis | 11 |
| 1.5.1 Research approaches | 11 |
| 1.5.2 Justification of the ‘design-research’ approach | 12 |
| 1.6 Structure of thesis and research question(s) | 13 |
| 1.7 References | 16 |
| Chapter 2: Criteria for requirements determination methods | 23 |
| 2.1 The criteria for RDMs found in the literature | 23 |
| 2.2 The domain richness criterion | 25 |
| 2.2.1 Reasons in the literature for systems failure caused by insufficient requirements determination | 25 |
| 2.2.2 Proposed techniques in the literature to overcome insufficient requirements determination | 27 |
| 2.2.3 Characteristics of application domains | 29 |
| 2.2.4 Definition of the domain richness criterion | 30 |
| 2.3 The completeness criterion | 31 |
| 2.4 The efficiency criterion | 33 |
| 2.5 The formality criterion | 35 |
| 2.6 Conclusions on the quality criteria for a RDM | 36 |
| 2.7 References | 37 |

| | |
|--|-----------|
| Chapter 3: Evaluation of existing RDM design alternatives | 41 |
| 3.1 Introduction | 41 |
| 3.2 A survey of approaches for requirements determination from the literature | 41 |
| 3.2.1 Data oriented approaches in requirements determination | 42 |
| 3.2.2 Process oriented approaches in requirements determination | 44 |
| 3.2.3 Object oriented approaches in requirements determination | 45 |
| 3.2.4 The Business Process Engineering approach: ARIS | 46 |
| 3.3 The suitability of existing approach families for requirements determination | 47 |
| 3.4 The extended (or enhanced) Entity-Relationship approach | 48 |
| 3.4.1 Deficiencies in the (E)ER way of modeling | 48 |
| 3.4.2 Deficiencies in the (E)ER way of working | 53 |
| 3.5 Object-role Modeling (ORM) | 55 |
| 3.5.1 Deficiencies in the ORM way of modeling | 55 |
| 3.5.2 Deficiencies in the ORM way of working | 59 |
| 3.6 The Unified Modeling Language (UML) | 59 |
| 3.6.1 Deficiencies in the UML way of modeling | 60 |
| 3.6.2 Deficiencies in the UML way of working | 64 |
| 3.7 Conclusions on the way of- modeling, -working, and –controlling for the Requirements determination approaches from the literature | 64 |
| 3.7.1 Overall modeling deficiencies | 65 |
| 3.7.2 Modeling deficiencies regarding the data model for the way of modeling | 65 |
| 3.7.3 Modeling deficiencies regarding the static constraints for the way of modeling | 67 |
| 3.7.4 Modeling deficiencies regarding the dynamic constraints for the way of modeling | 68 |
| 3.7.5 Modeling deficiencies regarding the static derivation (rules) for the way of modeling | 69 |
| 3.7.6 Modeling deficiencies regarding the dynamic rules for the way of modeling | 69 |
| 3.7.7 Modeling deficiencies regarding the way of working and way of controlling | 70 |
| 3.7.8 Summary of Modeling deficiencies in the EER, ORM and UML approaches | 70 |
| 3.8 The suitability of existing approaches for requirements determination with respect to the completeness, domain richness, efficiency, and formality criteria | 71 |
| 3.9 References | 74 |

| | |
|--|------------|
| Chapter 4: Operationalized design specification | 83 |
| 4.1 Introduction | 83 |
| 4.2 RDM demands for the way of modeling | 83 |
| 4.2.1 RDM demands for completeness in the way of modeling | 83 |
| 4.2.2 RDM demands for the domain richness in the way of modeling | 85 |
| 4.2.3 RDM demands for the efficiency in the way of modeling | 85 |
| 4.2.4 RDM demands for the formality in the way of modeling | 86 |
| 4.3 RDM demands for the way of working | 88 |
| 4.3.1 RDM demands for completeness in the way of working | 88 |
| 4.3.2 RDM demands for the domain richness in the way of working | 88 |
| 4.3.3 RDM demands for the efficiency in the way of working | 89 |
| 4.3.4 RDM demands for formality in the way of working | 89 |
| 4.4 RDM demands for the way of controlling | 90 |
| 4.4.1 RDM demands for efficiency in the way of controlling | 90 |
| 4.4.2 RDM demands for formality in the way of controlling | 90 |
| 4.5 Conclusions | 91 |
| 4.6 References | 91 |
| Developing an alternative design for a requirements determination method | 92 |
| Chapter 5: The way of modeling in Natural Language Modeling | 93 |
| 5.1 Introduction | 93 |
| 5.1.1 Organization of chapter 5 | 93 |
| 5.2 Names | 94 |
| 5.2.1 Example 5.1: University Enrolment part 1 | 94 |
| 5.2.2 The name primitive | 95 |
| 5.3 The natural language axiom | 96 |
| 5.4 Roles | 96 |
| 5.5 Intention and Extension | 98 |
| 5.6 Fact types | 99 |
| 5.6.1 Naming convention fact types | 100 |
| 5.6.2 Example 5.1: university enrolment part2 | 101 |
| 5.6.3 Compound referencing schemes | 104 |
| 5.7 The Basic Information Model | 107 |
| 5.8 NLM modeling constructs for the encoding of extensional constraints | 109 |
| 5.8.1 Definition of Population state constraints | 110 |
| 5.8.2 Definition of Population state transition constraints | 110 |
| 5.8.3 Definition of Derivation rule constraints | 111 |
| 5.8.4 Definition of Event occurrence, Event, Event type and Impulse type constraints | 112 |

| | |
|---|------------|
| 5.9 The NLM requirements specification for a given UoD | 114 |
| 5.9.1 The NLM Requirements specification for the University Enrolment UoD | 115 |
| 5.9.2 The University Enrollment case study description revisited | 118 |
| 5.10 Conclusions on the way of modeling in the NLM RDM | 120 |
| 5.10.1 The added value of the NLM requirements specification language | 121 |
| 5.11 References | 122 |

Chapter 6: The way of working and the way of controlling in Natural Language Modeling **125**

| | |
|--|------------|
| 6.1 Introduction | 125 |
| 6.1.1 Organization of chapter 6 | 126 |
| 6.2 The design procedure for a simple basic information model | 127 |
| 6.2.1 The verbalization transformation | 128 |
| 6.2.2 The grouping transformation | 132 |
| 6.2.3 The classification and qualification transformation | 135 |
| 6.3 The atomization procedure | 143 |
| 6.4 The procedure for integrating BIMs in NLM | 146 |
| 6.4.1 Conflicts on naming, synonyms and homonyms | 147 |
| 6.4.2 Specialization and generalization relationships | 148 |
| 6.4.3 Identifier, cardinality and domain conflict | 149 |
| 6.5 The population state constraint modeling procedures | 151 |
| 6.5.1 The derivation of uniqueness constraints | 151 |
| 6.5.2 The derivation of set comparison constraints | 153 |
| 6.6 The population state transition constraint modeling procedure | 157 |
| 6.7 The derivation rule constraint procedure | 159 |
| 6.8 The impulse constraint procedure | 161 |
| 6.9 The way of controlling in NLM | 165 |
| 6.9.1 The demarcation of the requirements determination project | 165 |
| 6.9.2 The required precedence of the requirements determination process in terms of the way of working | 166 |
| 6.9.3 Resource planning for a requirements determination project in NLM's way of working | 168 |
| 6.10 Conclusions on the way of working and the way of controlling in NLM | 169 |
| 6.10.1 The added value of the NLM requirements determination method | 170 |
| 6.11 References | 171 |

| | |
|---|----------------|
| Chapter 7: Conclusions, general discussion and recommendations | 175 |
| 7.1 Introduction | 175 |
| 7.1.1 Organization of chapter 7 | 176 |
| 7.2 Research findings | 176 |
| 7.3 Research methodology | 186 |
| 7.4 Future MIS research proposals | 186 |
| 7.5 Recommendations for practitioners in the MIS field | 187 |
| 7.5.1 Application of NLM in practice | 188 |
| 7.6 Concluding remarks | 188 |
| Appendix A: The specification of the constraint types in the NLM requirements specification language | 191 |
| A.1 Introduction | 191 |
| A.2 Population state constraints | 191 |
| A.3 Population state transition constraints | 192 |
| A.4 Derivation rule constraints | 194 |
| A.5 Impulse constraints | 199 |
| A.5.1 Impulse and impulse type | 201 |
| A.6 References | 209 |
| Appendix B: The meta model for the NLM requirements specification: | 211 |
| B.1 References | 215 |
| Summary (in dutch) | 217 |
| Curriculum vitae | 219 |

LIST OF FIGURES AND TABLES

| | |
|--|-----|
| Fig. 1.1 Requirements determination and fields of study | 3 |
| Fig. 1.2 The roles in the requirements determination process in general..... | 6 |
| Fig. 1.3 The roles in the requirements determination process in the ERP era | 10 |
| Fig. 1.4 The design research cycle and the chapters in this thesis | 16 |
| Fig. 3.1 Domain semantics and representation in EER model I | 49 |
| Fig. 3.2 Domain semantics and representation in EER model II (taken from figure 3.17 in McFadden et al., 1999:108) | 50 |
| Fig. 3.3 (a) ER, (b) XER and (c) EER cardinality constraints (taken from Liddle et al. (1993) and McFadden et al. (1999)) | 51 |
| Fig. 3.4 Domain semantics and representation in EER model III..... | 53 |
| Fig. 3.5 Domain semantics and representation in ORM model I..... | 56 |
| Fig. 3.6 Domain semantics and representation in ORM model II | 57 |
| Fig. 3.7 Domain semantics and representation in ORM model III | 57 |
| Fig. 3.8 Domain semantics and representation in ORM model IV | 58 |
| Fig. 3.9 Lack of coherence in UML class diagram and UML state chart diagram | 60 |
| Fig. 3.10 Two different object instances of the class House Inhabitant..... | 61 |
| Fig. 3.11 Generalization transformation using abstract class construct..... | 62 |
| Fig. 5.1 Example Vandover University Enrollment (example 5.1). | 95 |
| Fig. 5.2 Roles and sentence group template for university enrollment example | 97 |
| Fig. 5.3 Example legend for sentence groups. | 97 |
| Fig. 5.4 Naming convention fact type for student at Vandover university | 101 |
| Fig. 5.5 Example integrated Ohoadover enrollment system | 102 |
| Fig. 5.6 Fact types and sentence group templates with compound referencing scheme for student from the university enrolment example part 2 | 104 |
| Fig. 5.7 Naming convention fact type for student in the integrated UoD. | 105 |
| Fig. 5.8 Extended example legend for fact types..... | 106 |
| Fig. 5.9 Basic information model Ohoadover university enrollment..... | 108 |
| Fig. 5.10 NLM requirements specification (I):BIM and population constraints..... | 116 |
| Fig. 5.11 NLM requirements specification (II):derivation rule- and impulse type constraints | 117 |
| Fig. 5.12 NLM requirements specification (III):impulse type constraints..... | 118 |
| Fig. 6.1 The way of working in the NLM requirements determination method | 126 |
| Fig. 6.2 The verbalization transformation | 128 |
| Fig. 6.3 ‘real-life’ ABC invoice document (example 6.1). | 130 |
| Fig. 6.4 Result of verbalization transformation of example 6.1. | 132 |
| Fig. 6.5 Result of grouping transformation of example 6.1..... | 135 |
| Fig. 6.6 Graphical notation sentence group for (a part of) example 6.1. | 136 |
| Fig. 6.7 Initial application concept repository for (a part of) example 6.1. | 138 |
| Fig. 6.8 Application of naming convention fact types in NLM | 139 |
| Fig. 6.9 Result of classification/qualification transformation of example 6.1. | 142 |
| Fig. 6.10 Result of atomization transformation of example 6.1..... | 146 |
| Fig. 6.11 Basic information model for example 6.1 with uniqueness constraint(s).... | 153 |
| Fig. 6.12 Basic information model for example 6.1 with uniqueness and set- comparison constraint(s)..... | 156 |

| | |
|---|-----|
| Fig. 6.13 Basic information model for example 6.1 with uniqueness, set comparison and transition constraint(s)..... | 159 |
| Fig. 6.14 Basic information model for example 6.1 with uniqueness, set comparison, transition constraint and derivation rule constraints | 161 |
| Fig. 6.15 Complete NLM requirements specification for example 6.1 that contains a basic information model with uniqueness, set comparison, transition constraint, derivation rule constraints and impulse constraints..... | 164 |
| Fig. 6.16 AON network for activities in a NLM requirements determination project (I) | 168 |
| Fig. 6.17 AON network for activities in a NLM requirements determination project(II) | 168 |
| Fig. 7.1 Relationship between research (sub) questions)..... | 184 |
| Fig. A.1 Example legend for uniqueness-, exclusion-, subset- and equality- population state constraints..... | 192 |
| Fig. A.2 Example legend for population state transition constraints | 194 |
| Fig. A.3 Example legend for derivation rule constraint..... | 199 |
| Fig. A.4 Two different event types that trigger the same derivation rule | 202 |
| Fig. A.5 Two derivation rules that create instances of the same fact type..... | 203 |
| Fig. A.6 Event triggering a derivation rule when a condition is satisfied (impulse)... | 204 |
| Fig. A.7 Example legend type constraints | 206 |
| Fig. A.8 The possible temporal characteristic of impulse(s) (types) | 207 |
| Fig. B.1 Information meta model for BIM and population constraints in NLM..... | 214 |
| Table 2.1 Criteria for Requirements Specification methods | 24 |
| Table 2.2 Dimensions that characterize the application domain | 30 |
| Table 2.3 The definition of the domain richness criterion | 31 |
| Table 2.4 Types of rules according to Loucopoulos and Layzell (1989:264) | 32 |
| Table 2.5 Types of rules versus perspectives (Olle et al.,1988; Loucopoulos and Layzell, 1989)..... | 32 |
| Table 2.6 The definition of the completeness criterion | 33 |
| Table 2.7 The definition of the efficiency criterion | 34 |
| Table 2.8 The definition of the formality criterion..... | 35 |
| Table 2.9 Summary of the RDM criteria and definitions | 36 |
| Table 3.1 Main findings of Peckham and Maryanski survey (Peckham and Maryanski, 1988:181) | 42 |
| Table 3.2 Comparison families of approaches found in the literature..... | 47 |
| Table 3.3 Summary of the comparison of EER, ORM and UML approaches on modeling deficiencies, | 71 |
| Table 3.4 Comparison EER, ORM and UML approaches on completeness, domain richness, efficiency and formality criteria for the way of modeling, way of working and way of controlling | 73 |
| Table 7.1 Types of rules within perspectives for completeness criterion..... | 178 |
| Table 7.2 Requirements method demands for the way of modeling | 181 |
| Table 7.3 Requirements method demands for the way of working | 182 |
| Table 7.4 Requirements method demands for the way of controlling..... | 183 |

CHAPTER 1

INTRODUCTION

1.1 SUBJECT OF THE STUDY: REQUIREMENTS DETERMINATION

1.1.1 Requirements determination: Translating domain requirements into model requirements

The London Stock exchange automated trading system Taurus, had to be withdrawn before it ever was used (Stock exchange kills projects to focus on Taurus, 1989). The failure of National Insurance Recording System in England lead to tax overpayments by 800,000 people (System problems leave Inland revenue with £ 20 of taxpayers' cash, 2002). These are examples of organizations that have become victims of an unsatisfactory user requirements determination process. Unsatisfactory user requirements determination is one of the most prevalent reasons for faulty information systems or information systems that turn out to be overdue and too costly. Requirements determination is the least well-defined phase in the systems development process (Flynn, 1992) and: "has been widely recognized as the most difficult activity of information systems development." (Browne and Rogich, 2001:224). Failures in the requirements determination process represent one of the leading causes of system failure: "Given an appropriate design, most information systems departments can successfully implement a system. The big problem is correctly determining information requirements and designing the right system." (Wetherbe, 1991:52). "Many IS failures can be attributed to a lack of clear and specific information requirements." (Byrd et al., 1992:118). "Often, much of post-delivery maintenance work can be traced to requirements which had been poorly or falsely described in the system requirements specification (SRS), or were missed altogether." (Lang and Duggan, 2001:161). Errors in the requirements specification caused by a faulty requirements determination process can remain latent until the later stages in the IS development process (Viller et al., 1999:666) and will cost a manifold to fix in these later stages (Boehm, 1981, 1989). The subject of study in this thesis is generally known as *requirements determination* (Browne and Ramesh, 2002; Hevner and Mills, 1995), *requirements modeling* (Agarwal et al., 1996:138), *requirements engineering* (Rolland, 1999) or *requirements specification* (Sinha and Popken, 1996). "In its simplest form, requirements determination entails eliciting and encoding into the new system the requirements that clients verbalize to the analyst" (Alvarez, 2002:86); "Requirements engineering involves investigating the problems and requirements of the user community and developing a specification of the desired information system" (Loucopoulos, 1992:1). "Requirements determination can be defined as the process of gathering and modeling information about the required functionality of a proposed system by an analyst"

(Browne and Rogich, 2001:224). The requirements determination process, therefore, can be considered as a 'bridge' that embodies the translation of an organizational context in which (a) user(s) operates(s) in a language the analyst understands (Westrup, 1999:37). In the remainder of this thesis we will use the term requirements determination

1.1.2 Requirements determination: field of study

Research on *requirements determination* is found in a number of fields. In the 1960's and 70's it was an important research topic within the field of management information systems (Ackoff, 1967; Davis and Olson, 1985). One of the important fields that include much research on requirements determination is the field of information system development methodologies (ISDM). This field is mainly directed at the comparison of literally 100's of methodologies with the aim on how these methodologies can facilitate the development of computerized information systems. In the late 70's and eighties this field of study was in its heyday and many IFIP IS methodology conferences were organized (Olle et al., 1982, 1983, 1986, 1988b). The aim of the later conferences was to develop some prescriptive theory on what a good ISDM consists of (Olle et al., 1988a).

Secondly, there is the field of 'speech-act theory' in which information systems and conclusively the requirements are put in the context of communication and coordination of activities (Johannesson, 1995; Liu et al., 2003; Medina-Mora et al., 1992). In this field of study the development of information systems is considered to be submissive to the 'communicative action' it has to support.

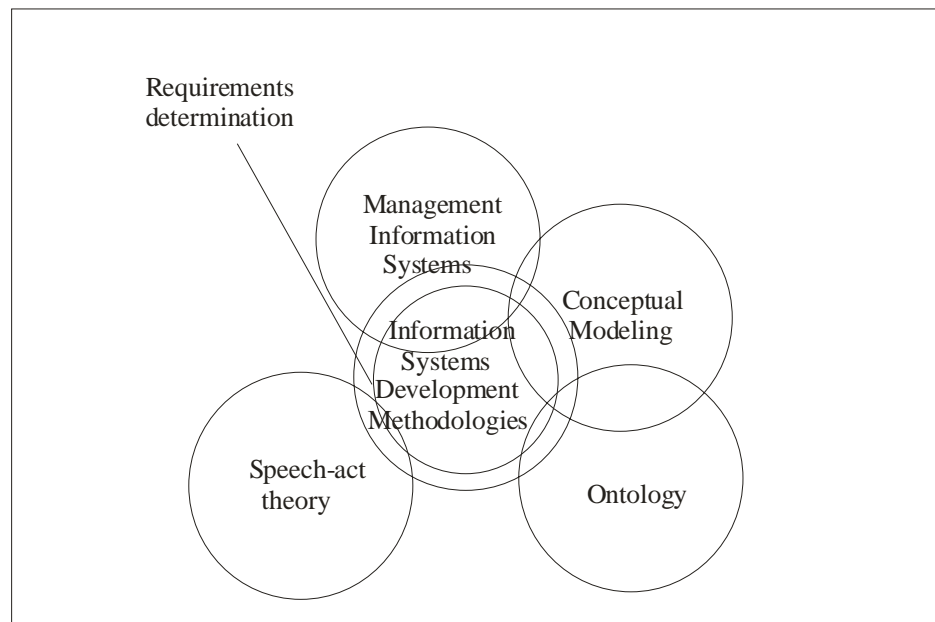


Fig. 1.1 Requirements determination and fields of study

In this study we will not focus on these communicative actions and how they are embedded in an organizational context. We will only focus on the information systems that need to be put in place for facilitating organizational communication.

The third field of study that is related to research on requirements determination is the field on ontological expressiveness in information systems analysis and design (Green and Rosemann, 2000; Rosemann and Green, 2002; Wand and Weber, 1993; Weber and Zhang, 1996). The main research question in the field of IS ontology is to establish a general domain ontology and subsequently measure the extent in which a given IS methodology has modeling provisions for expressing the elements in the domain ontology.

Another field that studies the languages that analysts can use to express domain requirements is the conceptual modeling approach (Brodie et al., 1984; Dietz, 1987; Loucopoulos, 1992). This approach emerged mainly as an answer or a cure against the fast changing standards for implementation technology, especially database management models in the 1960's and 1970's. Organizations in those days were faced with large technology transition costs because every time a new (database) technology was adapted they were forced to reprogram all their applications.

The subject of this thesis: requirements determination has predominantly been treated from the Information Systems Development Methodology and Conceptual Modeling points of view (see figure 1.1).

1.1.3 Organization of chapter 1

Now that we have introduced the subject and the fields of study to which it is related, we will give a historic overview of the main developments regarding the subject of this thesis in section 1.2. In section 1.3 we give the relationship between requirements determination and the stages in the information systems development life cycle. In section 1.4 we will give our research goal. In section 1.5 we will give our research approach. In section 1.6 we state the research (sub) questions in this thesis.

1.2. HISTORY OF REQUIREMENTS DETERMINATION

In the past 50 years computer scientists and business analysts have been struggling with the way in which the organizational knowledge and routines could be made explicit in order to apply information technology that can enable these knowledge workers to be more productive and be more effective.

The first generation of computer programmers consisted of researchers who programmed 'their' computer for solving their own information needs. In the very early days of computer use these information needs were mainly for mathematical applications and often for military applications. The focus was on *algorithms*, not on *data* (e.g. see for a discussion on *programs = algorithms + data structures*, Wirth (1976)). In the late fifties the first large-scale computer applications gradually came into use in businesses and other organizations. The functionality of these electronic data processing (EDP) systems, however, was an automated copy of the manual procedures that until then had been applied by a large number of clerks in these

corporations, e.g. general ledger and payroll applications. Information systems (IS) were primarily used to replace staff by machines, which resulted in cost savings and enhanced operational transaction efficiency (Jan and Tsai, 2002:62). The coding of these applications was completed by IS professionals leaving the end users in many cases aside (Jan and Tsai, 2002:62).

When the information systems applications that were needed in organizations during the seventies, eighties and nineties became more strategic (e.g. airline reservation systems, enterprise resource planning (ERP¹)), the developers attitude was by and large still the same. Information systems that were the result of applying the craftsmanship of the fifties and sixties, proved to be too costly and often not delivered on time. This situation has characterized the business information systems field during the sixties, seventies, eighties and nineties. In the sixties the concept of 'software crisis' was coined (Osmundson et al., 2003:1). This 'software crisis' emerged because, the way in which application information systems were developed resulted in cost overruns and long lead-times. By the time an information systems project was finished, the initial organizational requirements had already been changed. In many cases, however, the information systems development methodologies had not been able to capture those requirements in the right way.

The information systems development market place, however, changed in the early nineties of the last century when the *product software*-suppliers, e.g. MFG/PRO, IFS, SAP, BAAN, Marshal, Peoplesoft (Siriginidi, 2000: 387-389) started to sell their enterprise solutions on the waves of the Business Process Reengineering (BPR) sea (Davenport and Short, 1990; Hammer, 1990). These product software solutions, promised to solve many problems that were caused by the software crisis and were considered to be an attractive investment option in ICT for the large (Fortune 500) companies. The implementation of, for example, ERP² systems in a company, however, in most cases meant that the business process had to be reengineered or redesigned to fit one of the 'reference-model' that underlies the ERP package. This reengineering process turned out to be feasible for standard application functionality, for example, accounting, payroll, human resource management, inventory control. However, company-specific, functionality remained a problem in the first generation ERP-solutions. The second generation ERP-solutions, however, tried to redefine the concept of company-specific functionality, by developing 'standardized' software solutions for specific 'branches', for example, health-care, utilities, retail and so forth (Boudreau, 1999), for example Customer-Relationship Management (CRM) by Siebel (Molenaar, 2001). The development of the additional functionality in these second generation ERP systems, implied, in many cases, additional reengineering efforts on these branche-specific domains before an implementation could take place. In spite of the availability of the second -generation ERP solutions, many companies needed customized modules and interfaces that allows them to support the specific parts of their business (Soffer et al., 2003).

¹ ERP stands for Enterprise Resource Planning, that is an automated system in which information on all important enterprise resources, e.g. Financials, Human resources, Operations and Logistics, Sales and Marketing (Umble et al., 2003) can be stored and integrated.

² In the remainder of this dissertation we will use the term ERP as a synonym for product software, although in some contexts it can be considered to stand for a specific subclass of product software.

We will make a distinction between *development* requirements and *implementation* requirements in the context of product-software. Within the first type of requirements Dag et al. make a distinction into *customer wishes* and *product requirements* (Dag et al., 2004). Development requirements are the requirements that are of interest to the product-software developing organization, since they determine the functionality that has to be implemented in the product-software package. The implementation requirements reflect the required functionality from the point of view of the organization that is going to implement the product software. In this thesis we will focus on the implementation requirements.

1.3 REQUIREMENTS DETERMINATION AND INFORMATION SYSTEMS DEVELOPMENT

The improvement of the requirements determination processes for enterprise applications is still a relevant research subject within the field of management information systems because improving the state of the art in requirements determination methods to be applied in these requirements determination processes will have the following impact on organizations:

- It will enable them to express their (information) requirements using less (human) resources (more efficient).
- it will enable them to express their (information) requirements in a more precise, consistent and complete way

We will now give a definition of the intended outcome of such a requirements determination process: a *requirements specification* (Hevner and Mills, 1995:224; Pohl, 1994:245).

Definition 1.1. A *requirements specification* is a specification of *what* an information system must do (Wieringa, 1996:16).

1.3.1 Roles in requirements determination

If we now look back at the development in the development of (business) information systems over the past 50 years we can distinguish a number of roles in the requirements determination process:

- 1) The role of *user* or (*business domain expert*), these roles involve the knowledge of the business domain as it exist with the knowledge workers in the enterprise, for example the knowledge on how to process an invoice or how to approve a loan.
- 2) The role of the *analyst*, this role involves the knowledge on how to elicitate the knowledge of a knowledge worker in the focal enterprise in a format that can be used by a developer to develop an application

- system. The result of the work of the analyst we will call a requirements specification.
- 3) The role of the systems *developer*, this role involves the knowledge on how to transform an information systems specification into a working information system that complies with the *functional requirements* as embedded in the *requirements specification*.

In figure 1.2 we indicate the general relationships between the aforementioned roles.

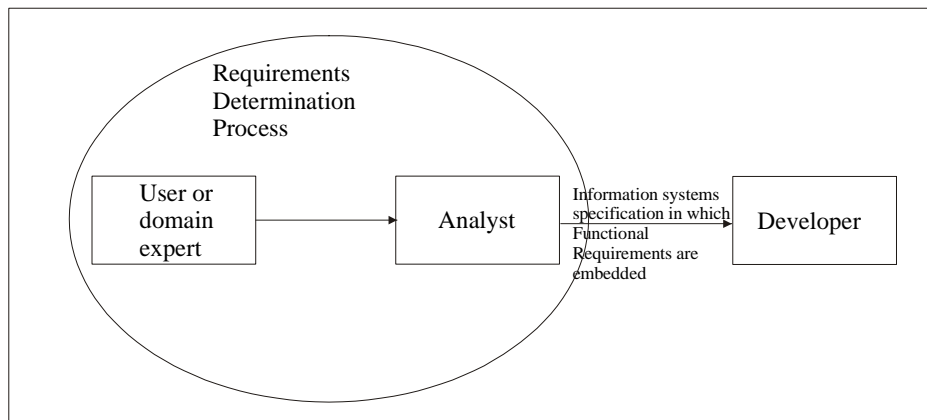


Fig. 1.2 The roles in the requirements determination process in general

1.3.2 The way of modeling, way of working and way of controlling in requirements determination

The extent in which the role of an analyst can be played perfectly in the requirements determination process depends upon the availability of ‘a way of working’, ‘a way of modeling’ and ‘a way of controlling’ (Wijers, 1991:14). A *way of modeling* refers to the model types that are required: “A way of modeling structures the models which can be used in information systems development. Several models are usually required for problem specification and solution in the application area” (Wijers, 1991:15).

Definition 1.2. A *way of modeling* in requirements determination is a specification of *what* can be contained in a requirements specification

A *way of working* or a *prescriptive process model* (Nurcan and Rolland, 2003): “is a description of processes at the type level. It defines how to use the concepts defined within a product Model.....A prescriptive Process Model is used to describe ‘how things must/should/could be done.’” (Nurcan and Rolland, 2003:62) The way of working refers to the process-oriented view of information system development, whereas the way of modeling refers to the product-oriented view of information system development.

Definition 1.3. A way of working in requirements determination is a specification of how a requirements specification can be created

Finally, a way of controlling is defined according to Verhoef (1993:8) as follows: “A way of controlling constitutes a management perspective of information systems development processes. Project management concerns considerations of time, means..., and quality... A way of controlling, therefore, includes directives and guidelines on progress control, resource allocation and quality management and control.”

Definition 1.4. A way of controlling in requirements determination constitutes a project and quality management perspective of the requirements determination process and concerns considerations of time, means, and quality in the creation of a requirements specification.

We will apply the distinction into a way of modeling, a way of working and a way of controlling for those stages in the development life cycle that focus on the requirements determination process (see figure 1.2). We will call a method that can be used by an analyst in such a requirements determination process: a *requirements determination method*.

Definition 1.5. A requirements determination method (RDM) is the combination of a specific way of modeling, a specific way of working and a specific way of controlling for creating requirements specifications, in which the way of working specifies how the elements in the way of modeling can be instantiated, and in which the way of controlling constitutes the project- and quality management perspectives for the given way of modeling and the given way of working.

1.3.3 Sub steps in requirements determination

The general requirements determination process from figure 1.2 is generally viewed as consisting of three steps (Browne and Rogich, 2001:225; Lalioti and Loucopoulos, 1994):

- 1) Information gathering (or requirements elicitation), during which an analyst elicits requirements from (a) user (s) or domain expert(s),
- 2) Representation (or requirements specification), in which those requirements are specified in some modeling language by the analyst,
- 3) Verification (or requirements validation) in which the analyst verifies the correctness of these requirements with the user.

If we consider the aforementioned steps in the requirements determination process, then we can state that the scientific research on these steps has not exclusively taken place in the fields of figure 1.1. For example, with respect to the step information gathering or requirements elicitation, substantial research has taken place within the field of Knowledge Engineering (Barrett and Edwards, 1995) leading to knowledge acquisition methods like KADS (Breuker and Wielinga, 1987). These approaches are primarily directed at ‘knowledge’ green fields, i.e. those application domains that were

generally considered to contain predominantly ‘tacit’ knowledge and these approaches were not developed for business application domains in which available knowledge has to be categorized and at most be made explicit.

With respect to the second step in the general requirements determination process: representation or requirements specification we can conclude that the definition of requirements specification languages has been a major research stream within the conceptual modeling and IS fields of study that deal with requirements determination. Major data-oriented ‘language families’ in this respect are the (extended) ER language (Chen, 1976; Teory et al., 1987) and the fact-oriented language family (Halpin, 2001; Verheijen and Van Bekkum, 1982). As an example of a ‘process-oriented’ specification language we can consider Data Flow Diagrams (DFD’s) (Yourdon and Constantine, 1979) or Activity Diagrams (A-schemas) in ISAC (Lundeberg, 1979).

With respect to the third step: requirements validation (or verification) we must make a distinction into *semantic verification* and *syntactic verification*. Semantic verification is the type of validation that we are interested in this thesis. It is concerned with the capturing of the ‘right’ domain requirements in terms of the extent in which what the analyst records is what the domain user intends to express. Dullea et al. (2003:171-172) define the concept semantic validity as follows: “An entity-relationship diagram is semantically valid only when each and every relationship exactly represents the modeler’s concept of the problem domain”. We will generalize this concept to every requirements determination method and more importantly, we will extend this concept beyond the modeler’s interpretation of the application domain to the user’s interpretation for the application domain, into our definition of a semantic correct specification.

Definition 1.6. A requirements specification is *semantically correct* if every element in the specification is a representation of the user(s)’(s) view of the application domain.

The outcome of a requirements determination process expressed in some specification language, therefore, should always be a semantically correct specification.

Syntactic verification, merely deals with the compliance of a specific application specification to the modeling rules that are contained in the meta-model of the specification language. We must be aware of the possibility that a semantic incorrect specification can be syntactically correct in any given situation.

The existing research on requirements determination methods for management information systems has mainly dealt with how to represent the outcome of the requirements determination process and how one can enforce that the content of a requirement specification is syntactically correct. The steps in the requirements determination process that cover the semantic verification are missing in the existing requirements determination methods for management information systems or business information systems (Goldin and Berry, 1997:376). It is this niche in the MIS research field that we will explore in this thesis in order to find a requirements determination method in which the semantic verification is incorporated in an explicit way.

1.3.4 Eras in requirements determination

If we now apply the roles and the requirements process from figure 1.2 in a historic perspective we can say that in the early years (1950's) the roles of user, analyst and developer coincided. In the 1960's a distinction was made between the roles of 'user' and 'programmer'. In that era the role of analyst and developer coincided. " In the premethodology era [prior to 1970], systems developers used a variety of techniques to help them develop computer-based information systems....They [techniques] were typically passed on to other systems developers, often by word of mouth. These rules or techniques were typically not codified and sometimes not written down.....Systems development was considered a technical process to be undertaken by technical people. In this era, systems development was all art and no science." (Hirschheim and Klein, 1992:296-297)

In the 1970's a clear separation took place between the functional requirements and the way in which these functional requirements were coded in a specific implementation technology (Tsichritzis and Klug, 1978). The distinction between an *information analyst* and *systems developer* emerged. The application of information systems development methodologies was aimed at the creation of 'tailor-made' information systems in which the needs of the domain users served as input.

In the ERP era (1990 and onwards) the roles of the user (or domain expert), analyst and developer were becoming more iterative instead of the linear sequence in which those roles were performed in the 1970's and 1980's. Because the implementation of ERP-systems usually is linked to business process redesign (Davenport, 1998; Rolland and Prakash, 2000:180) or a business process reengineering exercise (Skok and Legge, 2002:72), the role of the user or domain expert becomes more complex. In cooperation with the ERP-analyst the domain expert has to evaluate a number of proposed ways of working that will be supported by the specific ERP system in the company (Soffer et al., 2001:183).

The roles that we have depicted in figure 1.2 have deliberately different names in figure 1.3, because an ERP analyst is not only modeling the user requirement of a proposed (or 'to-be') business process but in addition has to confront the user or domain expert with the different possible (or 'to-be') business logics that are available in the chosen ERP system. The business, therefore, is expected to select and adapt a reference model, based on available solutions with minimal changes and leaving no record of the enterprise's original requirements (Soffer et al., 2001:183). On the other hand, even when they decide to implement an ERP system some organizations (for example Reebok) still choose to customize (Light, 2001:417) and enhance the standard functionality of the ERP system (Soffer et al., 2003). We remark, that the focus of the requirements determination in this thesis is on the conceptualization of the information and decision rules that must be contained in an (ERP) application.

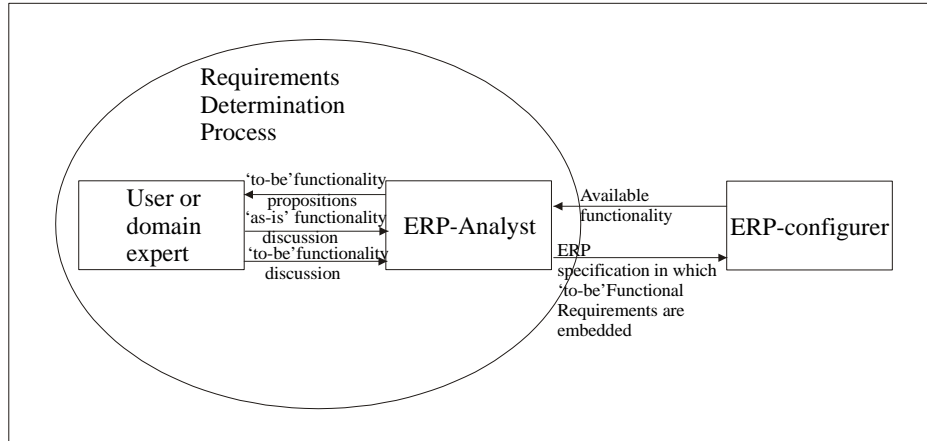


Fig. 1.3 The roles in the requirements determination process in the ERP era

Soffer et al. (2003) discuss in the context of ERP requirement-driven alignment the necessity of the construction of a modeling language that can model the entire scope of ERP options.

1.3.5 Conclusions on requirements determination as field of study

What we can conclude is that in spite of the trends in information systems development from ‘tailor-made’ towards ‘commercial-off-the-shelf’ (COTS) software implementations, the *requirement determination process* still is a significant process in the development life cycle of information systems. Moreover, the increase in complexity of the requirements determination process due to the use of ‘pre-fabricated’ software with its numerous implementation options (see the discussion on *configuration tables* in Davenport (1998)) has basically increased the need for *requirements determination methods* that have a way of modeling that can capture the complete set of user requirements and which way of working will guide the analyst in extracting all relevant business entities and business rules for a specific application domain.

In this thesis we will focus the content of a requirements specification on the conceptual aspects, in terms of the information and decision rules that underly an application (the *what* aspects). The application requirements that deal with transaction processing, workflow management, information retrieval and decision support (the *how* aspects³) will not be part of the definition of a requirements specification in this thesis.

³ In a typical ERP implementation 90-95 % of the 10 million lines of program code will be dedicated to functionality that covers transaction processing, decision support, consistency checking and data retrieval.

1.4 RESEARCH GOAL

We have found that the concept of requirements determination has been subject of an ongoing debate in the scientific literature. Over the past 40 years numerous requirements determination techniques, methodologies and approaches have emerged. Most of them, however, have proven not to be sufficient to address many of the problems we have found in the literature. Goldin and Berry (1997:376) give an overview of requirements engineering methods and tools. They conclude that the starting point for these tools and methods is a (natural language) statement of the requirements. However, none of the tools and methods they mention gives much help in how such a (natural language) requirement statement can be obtained. We have shown the gaps that exist in the existing research for the process of requirements determination and we have concluded that filling this gap in the literature is relevant to the state-of-the-art in the development of business information systems. Therefore, the goal of the research is to develop a method for requirements determination for which the way of modeling allows the analyst to capture these business entities and business rules in the application domain. This to-be developed RDM should have a way of working that contains modeling provisions that guide an analyst in eliciting the initial requirements from domain users. Finally, this method's way of controlling must contain quality preserving procedure(s) that guarantee(s) that a requirements specification that is the result of the application of this method has been validated by the user(s). This will lead to the following (main) research question in this thesis:

Does there exist a requirements determination method that is applicable in a wide range of business organizations and that can be used for specifying all domain requirements for a given business application subject area in an efficient, precise and consistent way ?

1.5 RESEARCH APPROACH AND OUTLINE OF THE THESIS

1.5.1 Research approaches

A theory explains how 'something' works. If such a theory is available we can predict how 'something' will behave under specific conditions (van der Zwaan and van Engelen, 1994).

Research in organizational studies (including the topic of this thesis: requirements determination) shows a continuous interaction between 'theory-development' and 'theory-application' (Den Hertog and Van Sluijs, 2000; Van Engelen and Van der Zwaan, 1994:93). A more detailed framework is given by Van Strien (1986:19) in which a distinction is made between an 'empirical' cycle and a 'regulative' cycle. The empirical cycle is characterized by the following stages: observation, induction, deduction, testing, evaluation. The regulative cycle can be used

as an instrument for the methodological foundation for the ‘theory-application’ in organizational studies (van Strien, 1986:19; Van Engelen and Van der Zwaan, 1994:93). In the context of this thesis we will follow the ‘design-research’-cycle as given in Van Engelen and Van der Zwaan (1994: 87-91):

- 1) *Design objective*: In this stage of the design research cycle it will be determined *what* needs to be designed. The feasibility of the product or process that needs to be designed must be questioned in this phase. Furthermore, a ‘falsification’ criterion must be given which will allow the researchers to determine whether the product or process that needs to be designed will lead to an actual improvement of the current situation or the ‘state-of-the-art’ (Van Engelen and Van der Zwaan, 1994:90).
- 2) *Design specification*: the design process must be guided by the availability of an explicit design specification for the product or process that is given in the *design objective*. The design specification must be tested on the following variables (Van Engelen and Van der Zwaan, 1994:90): consistency, realizability, completeness.
- 3) *Generation of alternative designs*: here there exist two situations. In the first situation, there exist ‘established’ conditions under which most likely a number of products or process have been developed that can be used as alternative designs for our design objective, we will call this situation the *evaluation problem* (Van Engelen and Van der Zwaan, 1994:91). In the second situation no alternative designs exist yet. We will call this situation a *development problem*, since the alternative designs have to be developed in the research.
- 4) *Selection of desired design from the set of alternative designs*: In this stage the best alternative will be chosen. A number of methodologies can be applied during this stage: optimization techniques, bounded rationality and verification method.
- 5) *Evaluation*: in this stage the researcher(s) report the findings. This evaluation must contain a justification of how the alternative designs have been gathered and/or created, which assumptions have been made and how the selection of the desired design has taken place (Van Engelen and Van der Zwaan, 1994:92).

1.5.2 Justification of the ‘design-research’ approach

In the literature overview on the topic of this thesis: *requirements determination*, a relatively large number of fields have been found (see figure 1.1) that comprise (or are linked to) requirements determination for the semantic aspects of an application. Each of the fields approaches requirements determination from a specific angle. So the theory-development surrounding the topic of requirements determination is plentiful. What we have discovered is that the ‘gap’ in knowledge of requirements determination at large is not in the ‘theory-development’ but in how these theories can be applied in a way that will prevent the creation of ‘faulty’ application systems in practice. The preliminary goal of this research is to develop a *method for requirements determination* that can be applied under a range of domain contingencies (or that has a certain ‘domain-richness’). Such a ‘to-be’ designed requirements determination method,

however, must comply with the demands for a requirements determination process that are given in the scientific literature on requirements determination. The way of modeling of such a method must at least consist of a requirements specification *language* and the way of working of the method must at least contain a set of modeling *procedures* that can tell an analyst how to create valid expressions in the modeling language that reflect some (or preferably all) perspective(s) of the semantics of the application domain at hand.

Definition 1.7. A *requirements specification language* is a set of modeling concepts and an accompanying grammar for the application of the modeling concepts in a requirements determination project.

Definition 1.8. A *requirements determination procedure* is a document that tells an analyst *how* to create a *requirements specification* in a given *requirements specification language*.

In addition to the derivation of the *requirements determination method* we will introduce a notation for expressing a *requirements specification*. This notation, however, is one out of many possible notational conventions that can be used in the requirements specification document(s).

In addition, we remark that we will apply the *perfect technology* assumption in the remainder of this thesis. This means that we will assume requirements level semantics (Eshuis et al., 2002:245) which means that we will not be influenced in our research by (current) limitations in implementation technology.

The goal of this thesis is to find or develop a *requirements determination method* in which the *way of modeling* is specified by giving the language concepts of the method's requirements specification language, the *way of working* (or *prescriptive process model*) is specified by giving the requirements determination (sub)procedure(s) of this requirements determination method. The *way of controlling* will be specified by giving explicit quality preserving steps in the modeling procedures. The way of controlling, furthermore, will consist of an overarching procedure that will be contained that exactly specifies the precedence for executing the requirements determination procedures.

1.6 STRUCTURE OF THESIS AND RESEARCH QUESTION(S)

The structure of this thesis will follow the 'design-research'-cycle as given in Van Engelen and Van der Zwaan (1994: 87-91).

In this chapter we have delineated the *design objective*, the issue of requirements determination. There exist gaps in the current state-of-the-art in requirements determination. Therefore, the goal of the research is to develop a requirements determination method that can be applied under a number of domain- and process contingencies, that we will discuss in chapter 2 and that will improve the current state of the practice in requirements determination in terms of 'falsification criteria' or 'quality criteria'.

In chapter 2 we will analyze and synthesize characteristics of the application domain into a number of dimensions, while keeping the following question in mind:

What are according to the existing requirements determination literature, the quality criteria for a requirements determination method that can be used for eliciting, verifying and specifying the complete domain requirements for a given business application subject area in a wide range of business organizations, in an efficient and formal way ? (Research sub question 1).

We will define the quality of a requirements determination method as the extent in which it is *has domain richness*, is *complete*, is *efficient*, and is *formal*. Firstly, we will synthesize a number of contingency variables that can be used to characterize application UoD's and that have an effect on the quality of the resulting requirements specification. Moreover, we will look for variables in the literature that characterize a requirements determination process itself and that can have an effect on the quality of the requirements specification that is the outcome of such a process. This will result in the definition of the *domain richness* criterion. Secondly, we will determine what perspectives and what elements within each perspective must be covered by a requirements determination method. This will lead to the definition of the *completeness* criterion. Thirdly, we will define an *efficiency* criterion, in which the extent in which the application of a given requirements determination method will use 'resources' can be measured. Finally, we will define a measure for the preciseness and consistency of a requirements specification in terms of the required formality of the requirements determination method: the *formality* criterion. We will define the *formality* as the extent in which it is *precise* and *consistent* in expressing the semantics of the UoD.

In chapter 3 we will focus the *Generation of alternative designs* from the design-research cycle as we will evaluate existing requirements determination approaches (the *evaluation* problem in Van Engelen and Van der Zwaan (1994:91)) on the criteria that we have derived in chapter 2. We will evaluate a number of popular requirements determination methods for management information systems that we have encountered in the literature. We will give an answer to the following question in chapter 3.

Why do the existing requirements determination approaches from the literature not comply with the quality criteria for assessing requirements determination methods? (Research sub question 2).

We will conclude in chapter 3 that there is no existing approach documented so far in the literature that fully complies with the four design criteria for a successful requirements determination method (RDM). This means that a requirements determination method(s) that complies with the quality criteria for a successful requirements determination method still needs to be developed. This leads us to the *development problem* according to Van Engelen and Van der Zwaan (1994) since alternative designs will have to be developed in this research.

In chapter 4 of this thesis we will develop a framework for a requirements determination method or a(n) (operationalized) design specification that contains the operationalization of the 'design specification' from chapter 2 and that takes into

account the reasons for non-compliance to these criteria by the existing approaches. This will constitute the *design criteria* (or operationalized *design specification* according) for the requirements determination method. Furthermore, we will phrase the third research sub-question based upon the results from chapters 2 and 3 here:

What are the necessary elements for the way of modeling, the way of working and the way of controlling for a requirements determination method so that this method complies with the quality criteria that we have given for the design specification? (Research sub question 3)

In chapters 5 and 6 we will define a requirements determination method that fulfills the quality criteria according to the design specification that was given in chapter 4.

We will now rephrase the main research question of this thesis:

Does there exist a requirements determination method that is applicable in a wide range of business organizations and that can be used for specifying the complete domain requirements for a given business application subject area in an efficient and formal way ?

The main research question, therefore, will lead to an investigation into the general applicability of requirements determination methods in the context of management information systems. First we will give an answer to research sub question 1, in which we will give ‘quality’-criteria. Furthermore, these criteria should provide an answer to what ‘all domain requirements’ means and what ‘in an efficient, precise and consistent way’ means in the context of the final research question. Furthermore, we will define what ‘application in a wide range of business organizations’ means. Once we have found these criteria, we will be able to search for an existing RDM that complies with these criteria (research sub question 2). If such a method does not exist, we will have to specify the elements of a newly to-be defined RDM that complies with these criteria (research sub question 3).

In figure 1.4 we have summarized the research stages in terms of the ‘design-research’ cycle and in which chapters of this thesis they will be discussed. The *design objective* is demarcated in this chapter. In chapter 2 the initial *design specification* will be synthesized from a review of the literature. In chapter 3 we will evaluate the existing approaches from the literature; this will be the stage *evaluation of the existing alternative designs* in the terminology of the design research cycle. In chapter 4 we will phrase the operationalized design specification for *the development of an alternative design (of a requirements determination method)* in terms of the way of modeling, the way of working and the way of controlling. The kernel of this thesis will consist of the development of the alternative design for a requirements determination method. This is given in chapters 5 and 6. In chapter 7 we will give conclusions and give a direction for future research in this field.

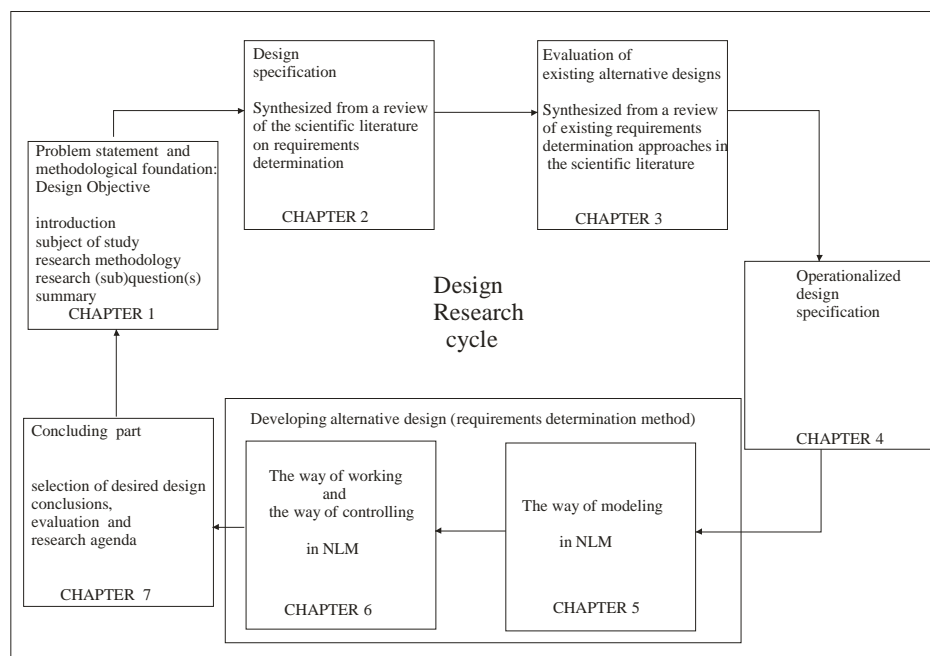


Fig. 1.4 The design research cycle and the chapters in this thesis

1.7 REFERENCES

Ackoff, R. (1967): Management Misinformation Systems. *Management Science* **14**(4)

Agarwal, R., Sinha, A., Tanniru, M. (1996): Cognitive Fit in Requirements Modeling: A Study of Object and Process Methodologies. *Journal of Management Information Systems* **13**(2): 137-162

Alvarez, R. (2002): Confessions of an information worker: a critical analysis of information requirements discourse. *Information and Organization* **12**:85-107

Barrett, A., Edwards, J. (1995): Knowledge Elicitation and Knowledge Representation in a Large Domain With Multiple Experts. *Exp. Syst. with Applications* **8**(1): 169-176.

Boehm, B. (1981): *Software engineering economics*, Prentice-Hall.

Boehm, B. (1989): *Software risk management*, IEEE computer society press

Boudreau, M. (1999): *ERP implementation and Forms of Organizational Change*. Working paper Georgia State University.

Breuker, J., Wielinga, B. (1987): Knowledge acquisition as modeling expertise; The KADS methodology. Paper presented at the 1st European workshop on Knowledge acquisition for knowledge based systems. Reading University.

Brodie, M.L., Mylopoulos, J., Schmidt, J. (eds.) (1984): On conceptual modeling, Springer Verlag, New York.

Browne, G., Ramesh, V. (2002): Improving information requirements determination: a cognitive perspective. *Information & Management* **39**:625-645

Browne, G., Rogich, M. (2001): An Empirical Investigation of User Requirements Elicitation: Comparing the Effectiveness of Prompting Techniques. *Journal of Management Information Systems* **17**(4): 223-249

Byrd, T., Cossick, K., Zmud, R. (1992): A Synthesis of Research on Requirements Analysis and Knowledge Acquisition Techniques. *MIS Quarterly* **16**(1): 117-138

Chen, P. (1976): The Entity-Relationship model: Towards a unified view of data. *ACM TODS* **1** (1) :9-36

Dag, J., Brinkkemper, S., Gervasi, V., Regnell, B. (2004): Speeding up Requirements Management in a Product Software Company: Linking Customer Wishes to Product Requirements through Linguistic Engineering. *Proceedings of Requirements Engineering Conference (RE'04)*. N.A.M. Maiden (ed.). IEEE Comp. science pr. 2004.

Davenport, T. (1998): Putting the enterprise into the enterprise system. *Harvard Business Review* **76**(4): 121-131

Davenport, T., Short, J. (1990): The new industrial engineering: Information Technology and Business Process Redesign. *Sloan management Review*. Summer : 11- 27.

Davis, G., Olson, M. (1985): *Management Information Systems, conceptual foundations, structure and development*, 2nd edition, McGraw-Hill.

Den Hertog, F., Van Sluijs, E. (2000): *Onderzoek in organisaties: een methodologische reisgids*. Tweede druk. Van Gorcum. Assen (in dutch)

Dietz, J. (1987): *Modelleren en specificeren van informatiesystemen*. Doctoral thesis Eindhoven Technical University (in Dutch)

Dullea, J., Song, I-Y., Lamprou, I. (2003): An analysis of structural validity in entity-relationship modeling. *Data & Knowledge Engineering* **47**: 167-205

Eshuis, R., Jansen, D., Wieringa, R. (2002): Requirements-Level Semantics and Model Checking of Object-Oriented Statecharts. *Requirements Eng* **7**: 243-263

Flynn, D. (1992): Information Systems Requirements: Determination and Analysis. McGraw-Hill.

Goldin, L., Berry, D. (1997): Abstfinder, a prototype natural language Text Abstraction Finder for Use in Requirements Elicitation. *Aut. Software Engineering* **4**: 375-412

Green, P., Rosemann, M. (2000): Integrated process modelling: an ontological evaluation. *Information Systems* **25**: 73-87

Halpin, T. (2001): Information Modeling and Relational Databases. Morgan Kaufmann Publishers .

Hammer, M. (1990): Reengineering work: Don't automate, obliterate. *Harvard Business Review*. july-august: 104-112.

Hevner, A., Mills, H. (1995): Box-structured requirements determination methods. *Decision Support Systems* **13**:223-239

Hirschheim, R., Klein, H. (1992): Paradigmatic Influence on IS development methodology: evolution and conceptual advances. In: *Advances in Computers*, Yovits, M. (ed.). JAI press London.

Jan, T-S., Tsai, F-L. (2002): A Systems View of the Evolution in Information Systems Development. *Systems Research and Behavioural Science* **19**: 61-75

Johannesson, P. (1995): Representation and communication-a speech act based approach to information systems design. *Information Systems* **20**(4): 291-303

Laloti, V., Loucopoulos, P. (1994):Visualisation of conceptual specifications. *Information Systems* **19**(3): 291-309

Lang, M., Duggan, J. (2001): A Tool to Support Collaborative Software Requirements Management. *Requirements Engineering* **6**:161-172

Light, B. (2001): The maintenance implications of the customization of ERP software. *Journal of software maintenance and evolution: research and practice* **13**: 415-429

Liu, K., Sun, L., Barijs, J., Dietz, J.(2003): Modelling dynamic behaviour of business organisations-extension of DEMO from a semiotic perspective. *Knowledge-Based Systems* **16**: 101-111

Loucopoulos, P. (1992): Introduction of section one. In: *Conceptual Modeling, Databases, and Case* (Loucopoulos, P., Zicari, R. (eds.)). Wiley. 1-19

Lundeberg, M. , Goldkuhl, G., Nilsson, G. (1979): A systematic approach to information systems development, *Information Systems* **4**: 1-12 and 93-118.

- Medina-Mora, R., Winograd, T., Flores, R., Flores, F. (1992): The Action Workflow Approach to Workflow Management Technology. *The Information Society* **9**: 391-404
- Molenaar, T. (2001): Siebel zet in op personeelsbeheer. *Computable* **43**: 26 oktober: p.11 (in dutch)
- Nurcan, S., Rolland, C. (2003): A multi-method for defining the organizational change. *Information and Software Technology* **45**: 61-82
- Olle, T., Sol, H., Verrijn-Stuart, A. (1982): *Information Systems Design Methodologies: a comparative review*. North-Holland
- Olle, T., Sol, H., Tully, C. (1983): *Information Systems Design Methodologies: a feature analysis*. North-Holland
- Olle, T., Sol, H., Verrijn-Stuart, A. (1986): *Information Systems Design Methodologies: improving the practice*. North-Holland
- Olle, T., Hagelstein, J., Macdonald, I., Rolland, C., Sol, H., Van Asche, F., Verrijn-Stuart, A.A. (1988a): *Information Systems Methodologies- A Framework for Understanding*, North-Holland .
- Olle, T., Verrijn-Stuart, A., Bhabuta, L. (1988b): *Computerized assistance during the information systems life cycle*, North-Holland .
- Osmundson, J., Michael, J., Machniak, M., Grossman, M. (2003): Quality Management metrics for software development. *Information & Management* **40** (8): 799-812
- Pohl, K. (1994): The three dimensions of requirements engineering: a framework and its applications. *Information Systems* **19**(3).
- Rolland, C. (1999): Requirements engineering for COTS based systems. *Information and Software Technology* **41**: 985-990
- Rolland, C., Prakash, N. (2000): Bridging the Gap Between Organisational needs and ERP Functionality. *Requirements Engineering* **5**: 180-193
- Rosemann, M., Green, P. (2002): Developing a meta model for the Bunge-Wand-Weber ontological constructs. *Information systems* **27**: 75-91.
- Sinha, A., Popken, D. (1996): Completeness and Consistency Checking of System Requirements; An Expert Agent Approach. *Expert Systems With Applications* **11**(3): 263-276
- Siriginidi, S. (2000): Enterprise resource planning in reengineering business. *Business Process Management* **6**(5): 376-391

- Skok, W., Legge, M. (2002): Evaluating Enterprise Resource Planning (ERP) Systems using an Interpretive Approach. *Knowledge and Process Management* **9** (2): 72-82
- Soffer, P., Golany, B., Dori, D., Wand, Y. (2001): Modelling Off-the-Shelf Information Systems Requirements: An Ontological Approach. *Requirements Engineering* **6**: 183-199
- Soffer, P., Golany, B., Dori, D. (2003): ERP modeling: a comprehensive approach. *Information Systems* **28** (6): 673-690
- Stock exchange kills projects to focus on Taurus. (1989): Editorial. *Computing* November; 2
- System problems leave Inland revenue with £ 20 of taxpayers' cash (2002). *Computer Weekly*. February 14.
- Teory, T., Yang, D., Fry, J. (1986): A logical design methodology for relational databases using the extended E-R model. *ACM Computing Surveys* **18**(2): 197-222
- Tsichritzis, D., Klug, A. (1978): The ANSI/X3/SPARC DBMS framework , *Information Systems* **3**: 173-191
- Umble, E., Haft, R., Umble, M. (2003): Enterprise resource planning; Implementation procedures and critical success factors. *European Journal of Operational Research* **146**: 241-257
- Van der Zwaan, A., Van Engelen, J. (1994): Bedrijfskundige methodologie 1: wetenschaps- theoretische context. *Bedrijfskunde* **66** (1): 27-35 (in dutch)
- Van Engelen, J., Van der Zwaan, A. (1994): Bedrijfskundige methodologie 2: een technisch-methodologische context. *Bedrijfskunde* **66** (2): 85-94 (in dutch)
- Van Strien, P. (1986); *Praktijk als Wetenschap*. Van Gorcum. Assen/Maastricht (in dutch)
- Verhoef, T. (1993): Effective information modelling support. Doctoral Thesis. Technical University Delft.
- Verheijen, G., van Bekkum J. (1982): NIAM: An Information Analysis Method. In: Verrijn- Stuart, A., Olle T., Sol H., (eds.): *proceedings CRIS- 1*, North-Holland Amsterdam. 537-590
- Viller, S., Bowers, J. and Rodden, T. (1999): Human factors in requirements engineering: A survey of human sciences literature relevant to the improvement of dependable systems development processes. *Interacting with Computers* **11** (6): 665-698.

- Wand, Y., Weber, R. (1993): On the ontological expressiveness of information systems analysis and design grammars. *Journal of information systems* **3**: 217-237
- Weber, R., Zhang, Y. (1996): An analytical evaluation of NIAM's grammar for conceptual schema diagrams. *Information systems journal* **6**: 147-170
- Wetherbe, J. (1991): *Executive Information Requirements : getting it right*. *MIS Quarterly* **15**(1): 51-65
- Wieringa, R. (1996): *Requirements Engineering: frameworks for understanding*, John Wiley & Sons, New York.
- Wijers, G. (1991): *Modelling support in information systems development*. Doctoral thesis. Technical University Delft.
- Wirth, N. (1976): *Programs= algorithms + data structures*, Prentice-Hall.
- Yourdon, E., Constantine, L. (1979): *Structured Design*. Prentice Hall.

CHAPTER 2

CRITERIA FOR REQUIREMENTS DETERMINATION METHODS

In this chapter we will define the design criteria for a requirements determination method (RDM). When studying the literature on criteria for requirements specification (methods) we have found a number of criteria that have been derived for software requirement specifications (IEEE Std 830, 1998), software specification techniques (Khwaja and Urban, 2002), software requirements and design specifications (Boehm, 1984), the quality of data models (Moody and Shank, 2003; Kesh, 1995), system development methodologies (Wysocki and Young, 1990; Essink and Romkema, 1989) and requirements determination methods (Hevner and Mills, 1995). The former references differ to a large extent in the level of detail and granularity of the criteria that are given. A number of authors make a distinction into criteria for the product (i.e. the requirements specification) and the criteria for the process (i.e. the requirements determination technique or procedure) (Moody and Shanks, 1995; Khwaja and Urban, 2002). In section 2.1 we will list a number of criteria for software requirements specifications, information system development methodologies and requirements determination methods as found in the literature.

2.1 CRITERIA FOR RDMs FOUND IN THE LITERATURE

In this chapter we will ask ourselves the following question:

What are according to the existing requirements determination literature, the quality criteria for a requirements determination method that can be used for eliciting, verifying and specifying the complete domain requirements for a given business application subject area in a wide range of business organizations, in an efficient and formal way ?

In table 2.1 we have given the criteria for software requirements specification, information system development methodologies and requirements determination methods as found in our literature survey.

Table 2.1. Criteria for Requirements Specification (Methods)

| <i>Author</i> | Boehm (1984) | IEEE-830 (1998) | Wysocki and Young (1990) | Essink and Romkema (1989) | Hevner and Mills (1995) |
|-----------------|--------------------------------|-------------------------|---------------------------------|---|--------------------------------|
| | <i>Software specifications</i> | <i>requirements</i> | <i>Information Developm.t</i> | <i>systems methodol.</i> | <i>Req.Determ. methods</i> |
| <i>criteria</i> | completeness | correct | efficiency | User participation | consistency |
| | consistency | unambiguous | Communications ⁴ | maintainable ⁵ | closure |
| | feasibility ⁶ | complete | control | Specification must be an expression of the real domain requirements | completeness |
| | testability ⁷ | consistent | Documentation | IS must be built according to the above specifications | clarity |
| | | ranked ⁸ | Role definition | Efficient development process | |
| | | verifiable | consistency | | |
| | | modifiable ⁹ | | | |
| | | traceable | | | |

In sections 2.3, 2.4 and 2.5 we will synthesize the variables that are given in table 2.1 into a comprehensive set of criteria that we will use to evaluate potential designs for requirements determination methods. In addition to the variables that are explicitly listed in the literature on criteria for software requirement specifications, information systems development methodologies and requirements specification (methods), we will research the literature that covers requirements at large. This type of literature looks mainly at the characteristics of the application domain. In section 2.2 we will give a review of this literature and we will synthesize a number of dimensions into the *domain richness* criterion.

⁴ Refers to milestones in the IS development process at large.

⁵ Will be covered under domain richness (section 2.2)

⁶ Is concerned with the economics of an IS development and implementation project, is outside the scope of a to-be designed RDM in this thesis.

⁷ Refers to the properties of a software implementation rather than a requirements specification

⁸ Refers to the selection process of requirements, e.g. what requirement has a higher priority.

⁹ Will be covered under domain richness (section 2.2)

2.2 THE DOMAIN RICHNESS CRITERION

The first criterion in our research on requirements determination methods is ‘domain richness’. This criterion should enable us to express to what extent any given requirements determination method can be applied in different types of application domains. The domain richness criterion, therefore, must be able to measure the extent in which a requirements determination method is applicable, under a range of contingencies for the type of application domain (section 2.2.3). In order to operationalize this criterion we will first give a literature survey of causes of systems failure caused by a faulty requirements determination method in section 2.2.1. and an overview of proposed techniques to overcome this insufficient requirements determination in section 2.2.2.

2.2.1 Reasons in the literature for systems failure caused by insufficient requirements determination

Davis and Olson (1985:474) give four major reasons why it is difficult to obtain a correct and complete set of requirements:

- 1) The constraints on humans as information processors and problem solvers
- 2) The variety and complexity of information requirements
- 3) The complex patterns of interaction among users and analysts in defining requirements
- 4) Unwillingness of some users to provide requirements (for political or behavioral reasons)

In line with the fourth reason mentioned by David and Olson, Skok and Legge (2002:80) state that one of the key findings in the 5 case studies they examined was that in the implementation of an ERP system, the staff members were often reluctant to share knowledge and information, because they experienced this as a threat to their jobs. They further conclude in their study that the language barrier that exists between users and the (ERP) consultants can be a source of major problems. The employees that were interviewed in this study felt : “ that consultants and developers do not see the impact on business processes of their actions, do not pass on their knowledge, try to run the project, communicate badly and work to their own agenda” (Skok and Legge, 2002:81).

Wetherbe (1991), concludes in the context of ‘executive’ information requirements that a number of mistakes have been made in the past: viewing systems as functional instead of cross-functional, interviewing managers individually instead of jointly, asking the wrong questions during the interview and not allowing trial-and – error in the design process

Macaulay (1996) identified five possible causes of failure in the requirements determination process: poor communication between people, lack of appropriate knowledge (or shared understanding), inappropriate, incomplete or inaccurate documentation, lack of systematic process and poor management of people or resources.

According to Land (1998:3) : “Many of the all too frequent IS failures stem from failures in the mappings somewhere along the chain. The most vulnerable link in the chain is that which attempts to map the relationship between the real world requirements-the organizational environment- and the comprehensive list of requirements which is the intended outcome of the process of systems analysis.”

Land distinguishes four categories of relationships between an information system (as the result of a systems development process) and its organizational environment:

- 1) The *unchanging* environment, in which the information requirements of the system are not changing during its lifetime.
- 2) The *turbulent* environment, in which the requirements over the expected lifetime of the system are always changing.
- 3) The *uncertain* environment, in which the requirements of the system are unknown or uncertain.
- 4) The *adaptive* environment, in which the output of the system has an influence on the environment.

Land claims that for each contingency a different design methodology should be chosen.

Galal and Paul (1999:93) challenge the ‘fixed-point stance’ towards requirements determination for a number of reasons. First requirements do change during the development of an information system. Secondly, they state that the statements in a requirements document are inherently predictive. In case of the ‘wrong’ predictions, the requirements need to be adapted. Thirdly, the requirements are context specific.

With respect to the changeability of requirements, Sutton (2000:116) concludes that: “It is becoming recognized that it is more appropriate to see requirements definition as a periodic or even continuous process that feeds other processes of delivery and review that may never end.”

Galliers and Swan (2000) introduce a two-dimensional framework for information systems development. The first dimension is the objective (formal) versus the subjective (informal) dimension. The second dimension is the unitary versus the pluralist (multiple stakeholders) dimension

Bergman et al. (2002) address the political nature of the requirements determination process and propose to reconcile the wide array of conflicting problems into a workable solution by means of *heterogeneous engineering*. Bergman et al. claim that requirements engineers must involve themselves in the politics of system design at the beginning of project design. Bergman et al. give a political requirement engineering (PRE) process model that explicitly considers the political requirements ecology in the enterprise.

Castro et al. (2002:365-366) notice a mismatch between the understanding of the operational (business) environment and the way in which the requirements analysis is aligned with this operational environment. They argue that this alignment has been predominantly based upon the programming paradigm of the day. In the 70’s the structured programming paradigm lead to the development of the structured analysis methodologies (DeMarco, 1978; Yourdon and Constantine, 1979). The object-oriented programming paradigm has led to the development of object-oriented development methodologies (Booch, 1991; Booch et al., 1999). Castro et al. (2002) introduce a modeling framework based on concepts used during early requirements analysis.

2.2.2 Proposed techniques in the literature to overcome insufficient requirements determination

Davis and Olson (1985:480) propose four strategies for determining information requirements: asking, deriving from an existing information system, synthesizing from characteristics of the utilizing system and, discovering from experimenting with an evolving information system.

Flynn (1992:137-139) gives four requirements acquisition methods: observation, analysis of existing system, analysis of desired system documentation and interview and questionnaire. Larsen and Naumann (1992) carried out an experiment in which they compared the analyst's ability to discover user requirements as a function of the knowledge representation they used: *abstract* or *concrete*. The findings of this study indicate that the more abstract representation (in this study a 'logical' DFD) is not as effective as the more concrete representation (in this study a 'physical' DFD).

Lee and Kim (1992) studied the relationship between formalization of the stages in the information systems development life cycle and the overall success of the (management) information system. They empirically demonstrated the relationship between the formalization of MIS development and MIS success.

Browne and Ramesh (2002) used the reasons that were given by Davis and Olson (1985:474) and gave some techniques that address these shortcomings. They derived three general categories for the techniques that addressed the cognitive problems: *pre-elicitation conditioning*¹⁰, *direct prompting techniques*¹¹ and *indirect prompting techniques*. Browne and Ramesh discuss the following indirect prompting techniques: *Scenario response tasks*¹², *Devil's advocacy*¹³ and *External representation techniques*. They conclude that the *devil's advocacy* technique is useful for mitigating automaticity, recall and faulty reasoning problems. Furthermore they conclude that the construction of semi-formal diagrams, for example, Entity-relationship (ER)-diagrams and data-flow diagrams (DFD) while eliciting requirements is very difficult. Browne and Ramesh conclude that informal (external) representation techniques should facilitate the interaction between analysts and users and help overcome background differences among them.

Browne and Rogich (2001:228) divide prompting techniques into *context-dependent techniques* and *context-independent techniques*. They propose that context-independent techniques are the most suited to be used by analyst in the elicitation of user knowledge in general, because analysts will often be assigned to analyze the requirements of business processes for which their substantive knowledge is limited (Browne and Rogich, 2001:231).

¹⁰ "Permits explanation of key terms, and allows analyst to create and/or influence incentive scheme for decision maker" (Browne and Ramesh, 2002:633)

¹¹ Browne and Ramesh (2002:633-635) give the following direct-prompting techniques: *directed questions* and *What-if analysis*.

¹² "Causes reflection, resulting in knowledge being used rather than simply assumed." (Browne and Ramesh, 2002:633)

¹³ "Causes users to question assumptions and generate counter-arguments, revealing knowledge that otherwise would not be evoked and improving the accuracy of reasoning and judgments." (Browne and Ramesh, 2002:633)

Bubenko and Wangler (1992) distinguish a knowledge acquisition task within the requirements engineering cycle. This task is traditionally the interviewing of end users. They, however, propose a number of different techniques for the knowledge acquisition task: Analyzing example forms and structured documents produced by end users, reverse modeling of existing databases, accepting application descriptions in natural language. Bubenko and Wangler (1992) speculate that a natural language interface to conceptual modeling tools would provide the following benefits: “ All people can express assertions in natural language; the need to learn a formal modeling language is not required....., e.g., for showing the user the conceptualization effect of stating assertions of the application world in different ways. It is probable the lexical quality (the names used to denote components of a schema) would improve due to the need for the NL-system to lexically, syntactically and semantically analyze the input.” (Bubenko and Wangler, 1992:404).

Wetherbe (1991) gives as a proposed solution to the shortcoming in (executive) information requirements determination in the past, that the systems designers must be encouraged :” to use a cross-functional, joint application design that involves input from all key decision makers in the business process.Detail requirements can then be identified through prototyping .” (Wetherbe, 1991:64-65).

Laloti and Loucopoulos (1994:291) state: “...the first step in requirements engineering namely the knowledge acquisition step has the purpose of abstracting and conceptualizing relevant parts of the application domain. The knowledge elicited during the first step is then formally specified by the use of conceptual modeling formalisms. The third step in requirements engineering is the validation, which is the process of investigating a model (in this case an IS specification) with respect to its user perceptions. The purpose of validation in the development of information systems is to ensure that a specification really reflects the user needs and statements about the application. Its importance is widely recognized by most developers but still there is a lack of formal theory for efficiently carrying out validation.”

Flynn and Warhurst (1994) empirically investigated the validation process within requirements determination and concluded that during validation, analysts perceive users as being unable to express their requirements adequately, and analysts have to employ informal realistic examples to explain the specifications to the users because the users do not feel comfortable with method notations.

Ter Hofstede et al. (1997) claim that for deriving the information from an application domain, verbalising sample forms, cases etcetera taken from the Universe of Discourse in close cooperation with a domain expert (or user) will lead to a sufficient overview of the structure and rules in the UoD can be obtained.

Burg en van de Riet (1997) advocate the idea of using natural language theories and knowledge in the construction process of a conceptual model.

Ambrosio, Metais and Meunier (1997) state that “ To improve the conceptualisation of the UoD during the schema design process, the use of linguistics is necessary ”(Ambrosio et al., 1997:112).

Kim and March (1995) give a four-phase process model for requirements determination:“ 1.*Perception*-users perceive the enterprise reality. The same enterprise reality may be perceived differently by different users (inconsistency). Any one user may perceive only a part of the reality (incompleteness).2. *Discovery*-analysts interact with users to elicit their perceptions. 3. *Modeling*-based on the information identified in the discovery phase, analysts build a formal, conceptual model (representation) of the

enterprise reality. This model serves as a communication vehicle between analysts and users. 4. *Validation*- Before concluding that the model is correct, consistent and complete, it must be validated. Validation has two aspects: comprehension and discrepancy checking. Users must comprehend or understand the meaning of the model. Then they must identify discrepancies between the model and their knowledge of reality.” (Kim and March, 1995:103).

Sutton (2000:116) discusses the aforementioned notion of ‘perception’ in the sense that ‘meaning’ implies ‘meaning to someone’ and that any meaning is constructed by an observer and therefore it can not exist objectively.

Recently, the research field of knowledge management has emerged in which not only well-structured information creation process are the subject of analysis but also the concept of ‘tacit-knowledge’ (Coughlan and Macredie, 2002:50). Polanyi classifies knowledge into *tacit* knowledge and *explicit* knowledge: “Tacit knowledge is personal, context-specific, and therefore hard to formalize and communicate. ‘Explicit’ or ‘codified’ knowledge, on the other hand, refers to knowledge that is transmittable in formal, systematic language” (Polanyi, 1966). Kim et al. (2003) studied the existing distinction into ‘tacit’ and ‘explicit’ knowledge in the literature and concluded that a revised epistemology was necessary in order to make a distinction into the concept of ‘tacit’ knowledge as defined by Polanyi (1966) (in which tacit knowledge cannot be expressed externally) and the concept of ‘tacit’ knowledge as defined by Nonaka (1994) (in which tacit knowledge is defined as knowledge that is (currently) not expressed externally). They revised the existing epistemology by replacing the old concept of ‘tacit’ knowledge by the revised concepts of *tacit* knowledge and the new concept of *implicit* knowledge: “tacit knowledge is knowledge that cannot be expressed externally and implicit knowledge is knowledge that can be expressed externally when needed, but currently exists internally” (Nonaka, 1994:3). The existence of knowledge processes that have a ‘tacit’ nature in the application domain is a characteristic of an application domain and has traditionally not been considered as an application area that can be subject for a requirements determination process.

2.2.3 Characteristics of application domains

The results of the literature review suggest a number of dimensions that determine the characteristics of the application domain and the scope of the requirements determination process. The first factor that we must take into account is how we define the scope of the requirements determination process. The information systems development paradigm that we will adhere to is the ‘functionalism’ paradigm (Hirschheim and Klein, 1989:1202-1203) in which systems development and therefore requirements determination is considered to “proceed from without, by application of formal concepts through planned intervention with rationalistic tools and methods.” (Hirschheim and Klein, 1989:1210). Within this metaphysical stance of functionalism we can characterize the domain requirements basically along at least two dimensions: *perception* and *turbulence*.

The dimension *perception* refers to the extent in which different domain users have a different perception of an underlying reality. This means when one considers a Universe of Discourse, one should always take into account from which user perspective the Universe of Discourse is considered. We can make the following dichotomy with respect to the *perception* dimension. The dimension perception can

also be related to the discussion in Bergman et al. (2002) who address the political nature of the requirements determination process and propose to reconcile the wide array of conflicting problems into a workable solution by means of *heterogeneous engineering*. This dimension also addresses the unitary versus pluralist dimension in Galliers and Swan (2000).

With respect to the dimension *turbulence* we can consider a situation in which the environment of the application domain is constant (nothing ever changes) and on the other side of the continuum an environment where there is continuous change. If we consider the turbulence of the application environment it is clear that in the case of ‘no change’ (the unchanging environment (Land, 1998)) we will pose less demands on the requirements determination methodology in terms of its ability to ‘evolve’ synchronously to the changes in the application domain, than in the case of an application subject area that is subject to a high frequency of changes in application, rules, information and procedures.

Another dimension regarding the type domain under consideration is concerned with the extent in which the domain knowledge is ‘tacit’ versus ‘explicit’. We will call this dimension the ‘tacitness’ dimension of the application subject area. The tacitness can range from fully ‘tacit’ area in which no single knowledge-creating process can be made explicit to a fully ‘explicit’ area in which every knowledge generating process can (potentially) be made explicit.

The fourth domain richness dimension is the way in which the requirements determination process is anchored. This dimension can range from a fully *abstract* anchor in which ‘open questions’ are posed to a tangible anchor in which *tangible* example forms and structured documents are used. Examples of requirements techniques that can be considered to be positioned somewhere on the ‘tangible’ side of this dichotomy are the analysis of forms and structured documents by end users (Bubenko and Wangler, 1992), others talk about ‘realistic’ examples (Flynn and Warhurst, 1994) or ‘verbalizing forms’ (Ter Hofstede et al., 1997). If we look at the abstract side of this dimension we encounter techniques like direct-prompting techniques, directed questions and what-if analysis (Browne and Ramesh, 2002).

We can now summarize the four ‘domain richness’ dimensions that characterize application domains in table 2.4.

Table 2.2 Dimensions that characterize the application domain

| Dimension | Low extreme | | High extreme |
|------------------|-----------------------|---|-------------------------|
| Perception | uniform for all users | - | Different for all users |
| Turbulence | no change | - | continuous change |
| Tacitness | fully tacit | - | fully explicit |
| Anchoring | tangible | - | abstract |

2.2.4 Definition of the domain richness criterion

We will now give a definition of the domain richness criterion for requirements determination methods. This criterion will reflect the extent in which the four dimensions can be accommodated by a single requirements determination method at the same time.

Definition 2.2. The *domain richness* of a *requirements determination method* is the extent in which this method can be applied under the full range of values for the given dimensions¹⁴.

In table 2.3 we have given the detailed definition of the domain richness criterion for each of the four dimensions given. We thereby consider the dimension *turbulence* relevant for the way of modeling, and the dimensions *perception*, *tacitness* and *anchoring* relevant for the way of working.

Table 2.3 The definition of the domain richness criterion

| | Way of Modeling | Way of Working |
|--|---|--|
| Definition of domain richness criterion | The extent in which the RDM can be applied for the full range of values for the dimension <i>turbulence</i> | The extent in which the RDM can be applied for the full range of values for the dimensions <i>perception</i> , <i>tacitness</i> and <i>anchoring</i> |

2.3 THE COMPLETENESS CRITERION

If we look at the five sets of criteria that are listed in table 2.1 we can conclude that in 4 sets of criteria the completeness of a specification is contained (Essink and Romkema call this: ‘Specification must be an expression of the real domain requirements’)

In order to arrive at a requirements specification that contains all relevant domain semantics for the specification of an application information system we first need to establish an idea of what we mean by *completeness* in the context of the way of modeling.

In this section we will operationalize the completeness criterion for a requirements determination method, e.g. *what* must be incorporated in a (semantic) requirements specification for application domains. Olle et al. (1988:41-43) distinguish three perspectives: the *data-oriented* perspective, the *process-oriented* perspective and the *behaviour-oriented* perspective. The data-oriented perspective should concentrate on the business data and must capture the domain concepts, the definition and the naming conventions for those domain concepts, the semantic relationships between the domain concepts and other ‘static’ and ‘structural’ knowledge in the enterprise. The process-oriented perspective should be able to capture the business activity and user perceivable tasks and describe what procedures exist for the creation of application facts or instances of semantic relationships. Finally, the behaviour-oriented perspective (Olle et al., 1988:43) should describe how ‘events’ can be cross-referenced to ‘elements’ in the process- and data-oriented perspectives (Olle et al., 1988:43). This means that any requirements specification should potentially consist of models that cover these three (conceptual) perspectives. Loucopoulos and Layzell (1989:264)

¹⁴ As defined in table 2.3

consider two perspectives a *state* perspective and an *action* perspective and distinguish as core of their concepts a *data* model and the types of *rules* as listed in table 2.4.

Table 2.4 Types of rules according to Loucopoulos and Layzell (1989:264)

| | State | Action |
|-------------------|--------------------|---------------------|
| Constraint | Static constraints | Dynamic constraints |
| Derivation | Static derivation | Dynamic rules |

Chakravarthy and Mishra (1994) and Campin et al. (1995a, 1995b) discuss the *Event-Condition-Action* (ECA) rules, that conceptually coincide with *dynamic rules* in the framework of Loucopoulos and Layzell. *Static constraints* are sometimes called *population constraints*, *static derivation* is also known as *derivation rules* and *dynamic constraints* are sometimes called *state transition constraints* (Halpin, 2001:298). From the requirements specification point of view we can now link the Olle et al. (1988) three perspectives framework and the constraint typology by Loucopoulos and Layzell in table 2.4.

Table 2.5 Types of rules versus perspectives (Olle et al.,1988; Loucopoulos and Layzell, 1989)

| | state | state | action |
|---------------------------|--------------|--------------------|---------------------|
| Data-oriented | Data model | Static constraints | Dynamic constraints |
| Process-oriented | | Static derivation | |
| Behaviour-oriented | | | Dynamic rules |

We can now conclude that a requirements method can be complete on two dimensions: the *number of perspectives* that are ‘covered’ by a method and secondly, the *types of rules* within every perspective that can be encoded using the requirements method. We will now give a definition of completeness in the context of a requirements determination method.

Definition 2.1. The completeness of a *requirements determination method* is the extent in which the types of rules in the data-, process- and behaviour oriented perspectives¹⁵ of an application can be captured in a requirements specification that is created using this method.

If we closely look at definition 2.1 we can operationalize this definition for the way of modeling where the *what* question is central, e.g. what modeling constructs need to be contained in a RDM. With respect to the way of working, the *how* question is relevant, e.g. how we find instances of these modeling constructs in a specific requirements determination project. The availability of (modeling) procedures is of great importance here.

In table 2.6 we have given the definition of the *completeness* criterion for the way of modeling and the way of working.

¹⁵ As given in table 2.5

Table 2.6 The definition of the completeness criterion

| | Way of modeling | Way of working |
|---|--|---|
| Definition of completeness criterion | The availability of modeling constructs for the data model, the static constraints, the static derivation, the dynamic constraints and dynamic rules | The availability of procedures for instantiating the data model, the static constraints, the static derivation, the dynamic constraints and dynamic rules |

2.4 THE EFFICIENCY CRITERION

If we look at the five sets of criteria that are listed in table 2.1 we can conclude that in 2 sets of criteria the *efficiency* of a RDM is contained. Efficiency in these sets of criteria, however, refers to the *process* dimension of requirements determination. The remaining three sets of criteria were derived for evaluating specifications from a product point of view. This means that the *efficiency* criterion is mainly concerned with the requirements (determination) process: ‘...a certain degree of guidance and direction can definitely improve efficiency.’ (Wysocki and Young, 1990:298).

The efficiency criterion that we will use for evaluating requirements determination methods is concerned with the amount of resources that is needed to create a requirements specification when such a requirements determination method is applied in a given requirements determination project. The operationalization of this criterion for the purpose of evaluating requirements determination methods must be done for the way of modeling and the way of working as well as the way of controlling. With respect to the way of modeling in general, and the requirements specification language in particular we can say that for two specification languages that have the same expressiveness (X and Y), a language X that has on average fewer modeling language constructs that serve the same purpose than a Language Y, implies that language X is more efficient than language Y. The existence of more than one modeling construct that serves the same purpose has a negative impact on the efficiency of a requirements determination method. This impact firstly, relates to cognitive aspects of the resources needed to learn a specification language and secondly, relates to the resources needed for selecting one modeling construct out of the set of alternative modeling constructs during the requirements determination process. Thirdly, the availability of ‘equivalent’ modeling constructs, and the limitations under which they can be applied might lead to modeling rework in a later stage of the project when additional information about the requirements specification becomes available. Rossi and Brinkkemper (1996) have compared 36 techniques and 11 methods on their descriptive capabilities based on the premise that: ‘generally speaking, the complexity of a method is related to the learnability and ease of use the method, even though this relationship may be complex.’ (Rossi and Brinkkemper, 1996:210).

With respect to the way of working of a requirements determination method we can say that the availability of a (set of) procedure(s) that guides an analyst in the requirements determination project will lead to an improved usage of (human) resources because it prescribes how an analyst should proceed in the process, given the knowledge he/she has elicited so far. The availability of such a procedure, therefore, will minimize the required number of analysis steps and rework that should be performed in a specific requirements determination project and will determine the efficiency of the way of modeling of requirements determination method. Such a procedure, furthermore, should contain a role definition (Wysocki and Young, 1990:300) for the analyst and must clearly make a distinction between the responsibilities of the analyst and the responsibilities of the user.

In IEEE Std. 830 (1998), a criterion called *verifiable* is given in the context of software requirements specifications: ‘A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement’ (IEEE, Std. 830, 1998).

With respect to the way of controlling we can define efficiency on two areas. Firstly, the area of *quality management*. In this philosophy, quality deficiencies must be prevented from happening, and if they do occur, they have to be ‘repaired’ by the process that is responsible for creating the deficiency. This means that the way of working of the method must contain a number of ‘quality-checking’ verification sub-procedures, in such a way that the process that is responsible for the performance of a requirements determination activity is responsible for the assurance of its quality. Secondly, the way of controlling is concerned with the *project management* of the requirements determination project. The efficiency for regarding these project management issues must be measured in terms of the three project management targets: *performance, cost and time* (Mantel et al., 2001:5)

Table 2.7 The definition of the efficiency criterion

| | Way of Modeling | Way of Working | Way of Controlling |
|---|---|---------------------------|--|
| Definition of efficiency criterion | Average number of modeling constructs in specification language that serve the same purpose | Availability of procedure | Availability of quality assurance steps. Extent in which performance, cost and time can be optimized |

Definition 2.3. The *efficiency* of a *requirements determination method* is the extent in which the efficiency criteria¹⁶ defined for the way of modeling, the way of working and way of controlling are satisfied.

¹⁶ As defined in table 2.7

2.5 THE FORMALITY CRITERION

In table 2.1 we can see that 4 out of 5 sets of criteria contain the criterion *consistency*. In 2 sets of criteria in table 2.1 some reference to precise requirements are made (correct and unambiguous). Boehm (1984) defines consistency as *internal consistency* ('items within the specification do not conflict with each other'. (Boehm, 1984:77-78)) and *external consistency* ('items in the specification do not conflict with external specifications or entities.' (Boehm, 1984:78)). In this thesis we will consider the *internal consistency* as a criterion for a requirements specification. *External consistency* is concerned with the application domain and lies outside the scope of the criteria for a RDM. A RDM, therefore, must lead to a *consistent* and *precise* requirements specification. In order to achieve requirements specifications that comply with these criteria we need a certain amount of formality in the way of modeling of the RDM. Firstly, the modeling constructs that are used for the specification of requirements in the different perspectives must be formally defined. Secondly, the way of working, must be formalized in some sort of algorithm(s) that precisely prescribe(s) how the consistent modeling constructs that were defined in the way of modeling, must be instantiated in a requirements determination project in order to obtain semantic (and syntactic) correct specifications. Such an algorithm must contain facilities to question user assumptions regarding the domain knowledge. With respect to the way of controlling we must be able to formalize the planning of activities, for example in a precedence diagram.

Boehm (1984) defines *traceability* as a sub-criterion within consistency ('items in the specification have clear antecedents in earlier specifications or statement of system objectives'. (Boehm, 1984:78)). In the IEEE recommended practice for software requirements specifications (IEEE Std. 830, 1998) traceability is defined as *backward* traceability ('this depends upon each requirement explicitly referencing its source in earlier documents.' (IEEE Std. 830, 1998:8)) and *forward* traceability ('this depends upon each requirement in the SRS having a unique name or reference number'. (IEEE Std. 830, 1998:8)) . We conclude that quality assurance steps must be embedded in (a) formal (sub) algorithm(s) including provisions that enable traceability.

Table 2.8 The definition of the formality criterion

| | Way of Modeling | Way of Working | Way of Controlling |
|--|--|-------------------------------------|---|
| Definition of formality criterion | Extent in which modeling constructs in language are formally defined | Extent in which procedure is formal | Extent in which activities can be formally planned. Extent in which quality management is contained in formal (sub)procedure Extent in which provisions that enable traceability are contained in RDM |

Definition 2.4. The *formality* of a *requirements determination method* is the extent in which the formality criteria¹⁷ defined for the *way of modeling*, the *way of working* and *way of controlling* are satisfied

2.6 CONCLUSIONS ON THE QUALITY CRITERIA FOR A RDM

In this chapter we have synthesized these criteria from the literature on information system development methodologies, software requirements specifications, and requirements specification (methods) in order to assess existing requirements determination methods from the literature in chapter 3 and to assess a newly to-be designed requirements determination method in chapters 5 and 6.

Table 2.9 Summary of the RDM criteria and definitions

| Criterion | Way of Modeling | Way of Working | Way of controlling |
|------------------------|--|---|--|
| Domain richness | The extent in which the RDM can be applied for the full range of values for the dimension <i>turbulence</i> | The extent in which the RDM can be applied for the full range of values for the dimensions <i>perception</i> , <i>tacit-ness</i> and <i>anchoring</i> | |
| Completeness | The availability of modeling constructs for the data model, the static constraints, the static derivation, the dynamic constraints and dynamic rules | The availability of procedures for instantiating the data model, the static constraints, the static derivation, the dynamic constraints and dynamic rules | |
| Efficiency | Average number of constructs in specification language that serve the same purpose | Availability of procedure | Availability of quality assurance steps. Extent in which performance, cost and time can be optimized |
| Formality | Extent in which modeling constructs in language are formally defined | Extent in which procedure is formal | Extent in which activities can be formally planned. Extent in which quality management is contained in formal (sub)procedure Extent in which provisions that enable traceability are contained in the RDM |

¹⁷ As defined in table 2.8

In sections 2.2 through 2.5 we have operationalized these criteria, which a requirements specification and a requirements determination method have to comply with. The criteria that we have defined in this chapter for the way of modeling, the way of working and the way of controlling for a requirements determination method were: *domain richness*, *completeness*, *efficiency* and *formality*. In table 2.9 we have summarized the criteria for a requirements determination method that we have derived in this chapter.

In chapter 3 we will investigate a number of ‘state-of-the-art’ requirements determination approaches and we will assess these approaches on the quality criteria that we have defined in this chapter.

In chapter 4 we will operationalize the four criteria that we have given into explicit demands for the way of modeling, the way of working and the way of controlling of a to be designed RDM.

2.7 REFERENCES

Ambrosio, A., Metais, E., Meurier, J-N. (1997): The linguistic level: Contribution for conceptual design, view integration, reuse and documentation. *Data & Knowledge Engineering* **21**: 111-129

Bergman, M., King, J., Lyytinen, K. (2002): Large-Scale Requirements Analysis Revisited; The need for Understanding the Political Ecology of Requirements Engineering. *Requirements Engineering* **7**: 152-171

Boehm, B. (1984): Verifying and validating software requirements and design specifications. *IEEE software* **1**(1): 75-84

Booch, G. (1991): *Object-Oriented Analysis and design with applications*. Benjamin Cummings, Redwood California.

Booch, G., Rumbaugh, J., Jacobson, I. (1999): *The Unified Modeling Language User Guide*. Addison-Wesley

Browne, G., Ramesh, V. (2002): Improving information requirements determination: a cognitive perspective. *Information & Management* **39**: 625-645

Browne, G., Rogich, M. (2001): An Empirical Investigation of User Requirements Elicitation; Comparing the Effectiveness of Prompting Techniques. *Journal of Management Information Systems* **17**(4): 223-249

Bubenko, A., Wangler, B. (1992): Research Direction in Conceptual Specification Development. In: *Conceptual Modeling, Databases, and Case* (Loucopoulos, P., Zicari, R. (eds.)). Wiley. 389-412

- Burg, J., Riet, R. van der (1997): The impact of linguistics on conceptual models: consistency and understandability. *Data & Knowledge Engineering* **21**: 131-146
- Campin, J., Paton, N., Williams, H. (1995a): A structured specification of an active database system .*Information and Software Technology* **37**(1): 47-61
- Campin, J., Paton, N., Williams, H. (1995b): Specifying Active Database Systems in an Object-Oriented Framework. *International Journal of Software Engineering and Knowledge Engineering*
- Castro, J., Kolp, M., Mylopoulos, J. (2002): Towards requirements-driven information systems engineering; the Tropos project. *Information Systems* **27**: 365-389
- Chakravarthy, S., Mishra, D. (1994): Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering* **14**: 1-26
- Coughlan, J., Macredie, R. (2002): Effective Communication in Requirements Elicitation; A Comparison of Methodologies. *Requirements Engineering* **7**: 46-60
- Davis, G., Olson, M. (1985): *Management Information Systems, conceptual foundations, structure and development*, 2nd edition, McGraw-Hill.
- DeMarco, T. (1978): *Structured Analysis and System Specification*, Yourdon Press.
- Essink, L., Romkema, H. (1989): *Ontwerpen van informatiesystemen*. Academic-service. (in dutch)
- Flynn, D., Warhurst, R. (1994): An empirical study of the validation process within requirements determination. *Information Systems Journal* **4**: 185-212
- Galal, G., Paul, R. (1999): A qualitative Scenario Approach to Managing Evolving Requirements. *Requirements Engineering* **4**: 82-102.
- Galliers, R., Swan, J. (2000): There's more to information systems development than structured approaches: information requirements analysis as a socially mediated process. *Requirements Engineering* **5**(2): 74-82
- Halpin, T. (2001): *Information Modeling and Relational Databases*, Morgan Kaufmann Publishers
- Hevner, A., Mills, H. (1995): Box-structured requirements determination methods. *Decision Support Systems* **13**: 223-239
- Hirschheim, R., Klein, H. (1992): Paradigmatic Influence on IS development methodology: evolution and conceptual advances. In: *Advances in Computers*, Yovits, M. (ed.). JAI press London.

- IEEE Std 830 (1998): IEEE Recommended practice for software requirements specifications. IEEE New-York
- Kesh, S. (1995): Evaluating the quality of entity relationship models. *Information and Software Technology* **37**(12): 681-689.
- Khwaja, A., Urban, J. (2002): A synthesis of evaluation criteria for software specifications and specification techniques. *Journal of Software Engineering and Knowledge Engineering* **12**(5): 581-599
- Kim, Y-G., March, S. (1995): Comparing Data Modeling Formalisms. *Communications of the ACM* **38**(6): 103-115.
- Kim, T-G., Yu, S-H., Lee, J-W. (2003): Knowledge strategy planning: methodology and case. *Expert Systems with Applications* **24**(3): 295-307
- Lalioti, V., Loucopoulos, P. (1994): Visualisation of conceptual specifications. *Information Systems* **19**(3): 291-309
- Land, F. (1998): A Contingency Based Approach to Requirements Elicitation and Systems Development. *Journal of Systems and Software* **40**: 3-6
- Larsen ,T, Naumann, J. (1992): An experimental comparison of abstract and concrete representations in systems analysis, *Information and Management*, **22** (1): 29-40
- Lee, J., Kim, S-H. (1992): The relationship between procedural formalization in MIS development and MIS success. *Information & Management* **22**: 89-111
- Loucopoulos, P., Layzell, P. (1989) : Improving information system development and evolution using a rule-based paradigm. *Software Engineering Journal*, BCS/IEE. **4**(5): 259-267.
- Macaulay, L. (1996): *Requirements Engineering*. Springer. London.
- Mantel, S., Meredith, J., Shafer, S., Sutton, M. (2001): *Project management in practice*. Wiley & Son.
- Moody, D., Shanks, G. (2003): Improving the quality of data models: empirical validation of a quality management framework. *Information Systems* **28**: 619-650
- Nonaka, I. (1994): A dynamic theory of organizational knowledge creation. *Organization Science* **5** (1): 14-37
- Olle, T., Hagelstein, J., Macdonald, I., Rolland, C., Sol, H., Van Asche, F., Verrijn-Stuart, A.A. (1988): *Information Systems Methodologies- A Framework for Understanding*, North-Holland .
- Polanyi, M. (1966): *The tacit dimension*, Routledge & Kegan Paul ltd. London

- Rossi, M., Brinkkemper, S. (1996): Complexity metrics for systems development methods and techniques. *Information Systems* **21**(2): 209-227
- Skok, W., Legge, M. (2002): Evaluating Enterprise Resource Planning (ERP) Systems using an Interpretive Approach. *Knowledge and Process Management* **9** (2): 72-82
- Sutton, D. (2000): Linguistic Problems with Requirements and Knowledge Elicitation. *Requirements Engineering* **5**: 114-124
- Ter Hofstede, A., Proper, H., Weide, T. van der (1997): Exploiting fact verbalisation in conceptual information modelling. *Information Systems* **22**: 349-385
- Wetherbe, J. (1991): Executive Information Requirements : getting it right. *MIS Quarterly* **15**(1): 51-65
- Wysocki, R., Young, J. (1990): *Information systems management principles in action*. John Wiley and Sons.
- Yourdon, E., Constantine, L. (1979): *Structured Design*. Prentice Hall.

CHAPTER 3

EVALUATION OF EXISTING RDM DESIGN ALTERNATIVES

3.1 INTRODUCTION

In this chapter of this thesis we will give an answer to the following research sub question that was stated in chapter 1:

Why do the existing requirements determination approaches from the literature not comply with the quality criteria for assessing requirements determination methods?

In order to answer this question we will first provide an overview of a number of (families) of approaches that are used in the process of requirements determination and that are documented in the information systems body of literature. Secondly, we will further discuss those approaches that at least contain a modeling language to express the data-oriented perspective of a requirements specification. After we have discussed these ‘families’ of requirements determination approaches we will compare them on the criteria that we have derived in chapter 2. We will analyze a member of each family of approaches and we will discuss the deficiencies of these approaches that need improvement in order to obtain requirements specifications that are precise and consistent, and that fulfill the completeness, domain richness, efficiency and formality criteria. The requirements modeling problems that we will encounter while discussing these approaches will be used when we are going to formulate the operationalized design specification of this thesis in chapter 4.

3.2 A SURVEY OF APPROACHES FOR REQUIREMENTS DETERMINATION FROM THE LITERATURE

Traditionally two ‘families’ of approaches can be distinguished in the field of requirements determination: the *data-oriented* approaches and the *process-oriented* approaches (Bubenko and Wangler, 1992:393). In addition to these two groups of approaches, hybrid approaches have emerged that are both data-oriented and process-oriented (Vessey and Conger, 1994:102). In the mid-eighties the object-oriented approach emerged in which *static* and *dynamic* features of an enterprise area should be considered together in objects (Parsons and Wand, 1997:109). In the nineties a business process engineering approach was introduced that provides facilities for the creation of a requirements definition (Scheer, 1998).

3.2.1 Data oriented approaches in requirements determination

The data-oriented approaches to requirements determination can be divided into the following families of semantic data modeling: the ER or Extended ER (Entity-Relationship) approach and the Fact Oriented Modeling approach (NIAM, ORM) (Kim and March, 1995:103). Peckham and Maryanski (1988) reported on a survey of a number of semantic data models. A summary of their findings is given in table 3.1.

Table 3.1 Main findings of Peckham and Maryanski survey (Peckham and Maryanski, 1988:181)

| Data model | Relationship representation | Derivation/inheritance | Relationship semantics | Dynamic modeling |
|------------|----------------------------------|---|------------------------|---------------------------------------|
| E-R | Independent and tables | No | User selectable | No |
| TAXIS | Entity (classes) | Inheritance | Predefined | Transaction modeling, object oriented |
| SDM | Independent and entity (classes) | Elaborate and varied | User defined | No |
| Functional | Functions | Functional | User defined | No |
| RM/T | Independent | Inheritance | Predefined | No |
| SAM | Independent | Summation over classes/ inheritance | Predefined | Object oriented |
| Event | Attributes | No | Predefined | Transaction modeling |
| SHM+ | Attributes, entities, separate | Inheritance over Generalization and Association hierarchies | Predefined | Transaction modeling |

In addition to the E-R and fact-oriented approaches, the Semantic Data Model (SDM) by Hammer and McLeod (1981) has had an influence the evolution of the dialects within the ER and Fact-oriented approaches and it has had an influence on the design of object-oriented modeling languages. However, in the survey that we will present in this thesis we will limit ourselves to the ER and Fact-Oriented families of approaches in data modeling.

The Entity-Relationship family of approaches

In 1976 Chen published the first article on Entity Relationship Modeling (Chen, 1976). The basic modeling constructs for capturing the data structure in the ER model are *entitie(s)(sets)*, *attributes* and *relationship(s) (sets)*. The *entities* are the objects in an application domain about which information is collected; *attributes* represent intrinsic properties of entities whose value does not depend upon other entities in the model. Relationships represent interconnections among entities (Pitrik, 1996:115). In addition Chen introduced (maximum) cardinalities that can express some of an application area's business rules. In the course of time Chen's original Entity-Relationship model has been extended EER (Teorey et al., 1986); ERT (Theodoulidis et al., 1991); EDM

(Scheer and Hars, 1992); NER (Silva and Carlson, 1995); ER+ (Kolp and Zimanyi, 2000); MEER (Balaban and Shoval, 2002)) in which additional conceptual abstractions have been incorporated (Chan et al., 1998:117): optional relationships (Teorey et al., 1986), subtyping (Teorey et al., 1986), aggregation, generalization/specialization (Gogolla and Hohenstein, 1991; Teorey et al., 1986), time and complex objects (Theodoulidis, 1991), update methods (Balaban and Shoval, 2002). A main characteristic according to Akoka and Comyn-Wattiau of ER models is that these models are mainly oriented towards the data-oriented perspective and leave the process- and behaviour-oriented perspectives undefined (Akoka and Comyn-Wattiau, 1996:88). At the beginning of the 21st century the family of (E)ER approaches remains the most popular approach used in practice (Rauh and Stickel, 1996:135; Shoval and Shiran, 1997:298) and in curricula of universities (in case we consider the static aspects of UML class diagrams as a flavour of the (E)ER family of approaches (Balaban and Shoval, 2002:245; Steimann, 2000:88)). Saiedian surveyed a number of (E)ER models in the literature (Saiedian, 1997) and concluded that in the course of time, the expressiveness of these models has increased and that the recent extensions incorporate object-oriented features, e.g. *methods*, *messages* and *operations*. We will make a distinction into ‘conventional’ EER approaches and ‘object-oriented’ EER approaches. The latter group will be considered object-oriented approaches and is analyzed in section 3.2.3.

The fact oriented family of approaches

From the pioneering work of Abrial (1974) on the Semantic Binary Relationship Model (SBRM), followed by the object-role model (Falkenberg, 1976a, 1976b) the fact-oriented approach became a relatively popular requirements specification approach when the object-role models were expressed in a ‘circle-box’ notation and accompanied by a modeling methodology (ENALIM) (Nijssen, 1977). The ENALIM methodology provided the foundation for (binary) NIAM (Verheijen and van Bekkum, 1982) and the Binary Relationship Model (Van Griethuysen, 1982). In the late 1980’s binary NIAM evolved into N-ary fact oriented information modeling (Halpin and Orlowska, 1992; Leung and Nijssen, 1988; Nijssen and Halpin, 1989) and the acronym NIAM became a shortcut for *natural language information analysis method* (Halpin, 1996). The data model in the fact-oriented approaches basically consists of a set of *fact types* and *entity types* that are connected through *roles*. A fact type is a semantic relationship consisting of N roles between (at most) N (different) entity types and/or label types. Every entity type can be involved in a number of roles within any number of fact types. The content of the fact base at any time is subject to the set of *population state* and *state transition constraints* that can be defined on the information structure diagram (ISD, see Verheijen and van Bekkum, (1982)). Prabhakaran and Falkenberg (1988:98) give an overview of other fact oriented modeling approaches, amongst them: CSL (Breutmann et al., 1979); CIAM (Gustafsson et al., 1982); DADES/RM/RA (Olive, 1982), REMORA (Rolland and Richard, 1982).

Conclusions for the data oriented approaches in requirements determination

The data oriented approaches in requirements determination are numerous. The two families of approaches that appear to be the most influential are the entity-relationship family of approaches and the fact-oriented family of approaches (Kim and March, 1995). The commonality of all members of these two families of approaches lies in the perspective of a subject area that they intend to model: the data-oriented perspective, this implies that these approaches have hardly any facilities for modeling the process-oriented and behaviour-oriented perspectives of an application domain.

3.2.2 Process oriented approaches in requirements determination

The process-oriented approaches that we will discuss in this thesis have their origins in the mid-seventies and have in common that they all are based on the notion that application systems can be modeled as a set of functions that interact and that some form of (functional) decomposition is required. We will discuss three (sub) approaches within the process-oriented family: SADT, the structured analysis and structured design (SA/SD) school and ISAC.

SADT

SADT was developed by Douglas T. Ross (Maarssen and McGowan, 1986). A SADT model is a series of data flow diagrams that consists of *activities* that transform *input* data onto *output* data. *Control* governs the way in which the *transformation* takes place and mechanisms show the means by which an activity is performed (Maarssen and McGowan, 1986). A transformation can be decomposed into a sub transformation on a lower level of abstraction.

Structured analysis and structured design (SA/SD) school

The structured analysis and structured design school is based upon work of Yourdon and Constantine (1979), Gane and Sarson (1979) and DeMarco (1978). SA/SD looks upon enterprises as functions that process data. The main diagramming technique in SA/SD is the data flow diagram. Data Flow diagrams are designed to show the functionality of an application. A data flow diagram (DFD) consists of a collection of processes, flows, stores, terminators. The processes in a DFD constitute the activities of the application that is being represented. Each *process* is considered to transform the *incoming flows* of data and material into outgoing flows of data (and material). *Stores* are containers for data or material that is carried in the flow. *Terminators* represent actors or processes external to the system that is under consideration. A transformation can be decomposed into a sub transformation on a lower level of abstraction. Opdahl and Sindre (1994:231-234) point at the omissions in the application of DFD's for real-world modeling. They conclude that decomposition of DFD's is problematic with respect to flows. They specifically critique the real world meaning of high-level flows.

Ward (1986) extended the data-flow diagram by concepts that represent control and timing.

ISAC

ISAC (Lundeberg et al., 1979; Lundeberg, 1982) was developed in the 1970's at Stockholm University and the methodology was created for a much broader coverage of stages in the information systems development life cycle than the requirements determination stage. The ISAC methodology consists of the following stages: change analysis, activity study, information analysis, data system design (Ruys, 1983). Falkenberg et al. (1983:188) concluded that ISAC is very strong in the earliest phases of systems design i.e. change analysis and activity analysis, however the data-model that is the outcome of ISAC is cumbersome and cannot guarantee integrity. Hanani and Shoval (1986:249) conclude that a major gap exists between the products of information analysis and the design of the data system. Furthermore, ISAC does not have facilities for specifying static and dynamic constraints.

Conclusions for the Process oriented approaches in requirements determination

Floyd (1986) compared SADT, the SA/SD school and ISAC and concluded that only SADT and ISAC seemed suitable for requirements determination. Deng and Fuhr (1995:107) claim that structured analysis and design techniques cannot allow a simple modification to a module without a complete redesign of the system.

Henderson-Sellers and Edwards (1990:145) summarize the findings of Meyer regarding the flaws in top-down system design as is implemented in all process-oriented approaches that we have discussed in this chapter:

1. top down systems design does not take account of evolutionary changes,
2. in top down systems design a system is characterized by a single function,
3. top down design neglects the data structure aspect very often,
4. top down design does not encourage reuse.

3.2.3 Object oriented approaches in requirements determination

The object-oriented analysis concepts have their roots in object-oriented programming (Cox, 1986) and object-oriented software construction (Meyer, 1988) blended with ideas from semantic data modeling and knowledge representation (Mylopoulos et al., 1999).

Classic OO approaches in requirements determination

The OO-paradigm has been applied in corporations during the last decade (Johnson and Hardgrave, 1999:5) in methodologies for requirements determination and information systems analysis and design: object-oriented modeling and design (OMT) (Rumbaugh et al., 1991), object-oriented systems design (OOD) (Yourdon, 1994), object-oriented information engineering; analysis, design and implementation (Montgomery, 1994). Mistic and Graf (2004) empirically studied the use of different system analysis approaches by systems analysts and concluded that the percentage of respondents that are using object-oriented approaches for analysis and design grew from 0 % in 1994 to 35 % in 2001. Davis (1995) gives a critical view on the application of OO programming concepts for requirements specification and information systems design.

The OO paradigm considers an object as:” an identifiable thing that remembers its own state [1, 6], and that can respond to requests for operations with respect to this state [5].” (Parsons and Wand, 1997:106). Furthermore the OO-paradigm considers an object to have an unchangeable *identity* (Brown, 1991:20)), it *encapsulates* data and behaviour and it is *persistent* (Parsons and Wand, 1997:106). One of the first OO approaches that were specifically designed for requirements determination was OMT (Rumbaugh et al., 1991).

The Unified Modeling Language (UML)

In 1997 a standard emerged (OMG, 2002) to streamline the multitude of OO-approaches: The Unified Modeling Language (UML). An application’s data model can be expressed in an object-oriented class diagram. The static constraints and the static derivation rules can be defined in the static structure part of a class diagram, for example in a UML class diagram this can be done by using *association end* and *attribute* multiplicities and the Object Constraint Language (OCL). Furthermore, the dynamic constraints and dynamic rules of a domain application can partly be encoded as methods from object classes and in miscellaneous models that have come into existence like for example *use cases* and *state charts*.

Conclusions for the Object oriented approaches in requirements determination

We can conclude that the object-oriented approaches have evolved from the application of the OO-paradigm in programming languages in the seventies and eighties towards OO approaches that are considered suitable for requirements determination. The object-oriented approaches provide facilities for the specification of the data- as well as the process- and behaviour-oriented perspectives in requirements determination.

3.2.4 The Business Process Engineering approach: ARIS

From the late eighties until the mid-nineties the Business Process (Re)engineering was at its peak. Around that time product-software suppliers started to implement their IT solutions on a wide scale in (mainly) large organizations. Scheer has developed an Architectural framework for integrated information systems (ARIS), that can analyze, model and navigate business processes (Scheer, 1994:607). ARIS acknowledges the existence of a semantic requirements definition (Scheer, 1998:14). To express the data view of such a semantic requirements definition, ARIS uses an (E)ER model¹⁸. For representing the functional and control view, ARIS uses a number of modeling techniques, like for example flow-chart techniques, Petri-nets, activity-diagrams and OMT object-diagrams, object-flow diagrams.

¹⁸ ARIS consists of a number of requirements models that are basically covered by the modeling facilities in the other three families of approaches.

3.3 THE SUITABILITY OF EXISTING APPROACH FAMILIES FOR REQUIREMENTS DETERMINATION

In this paragraph we will evaluate the family of approaches that we have introduced in paragraph 3.2 on the first criterion that we have derived in chapter 2. In table 3.2 we have compared these three ‘families’ of approaches with respect to the underlying *way of thinking*, the *way of modeling* and their *way of working* (Wijers, 1991: 17-23).

Table 3.2 Comparison families of approaches found in the literature

| | Data-oriented | Process-oriented | Object-oriented | Business Process Engineering |
|------------------------|---|-----------------------------------|--|---|
| Way of thinking | Business data | Functions that interact | Objects that encapsulate data and behaviour | Business processes |
| Way of modeling | Information models | DFd’s, A-schemes | Class diagrams, use cases, activity diagrams, State charts | Process chains, OMT class diagrams, ER models, Petri-nets |
| Way of working | Analysis of textual description of application domain | Top-down functional decomposition | Identifying objects in application domain | Translating business application knowledge into DP -suitable structures |

The completeness criterion that was given in chapter 2, implies the capability of a requirements determination approach to specify at least the *data model* from the data-oriented perspective. The data model is necessary in order to be able to describe the content of the process- and behaviour-oriented perspectives in a meaningful way. The literature study has revealed that the process-oriented approaches do not provide sufficient modeling constructs that would allow an analyst to create a requirements specification that contains a data model. We will now analyze three families of approaches for requirements determination that have facilities for a *data model*: The Entity-Relationship approach, the fact-oriented approach and the object-oriented approach (OO). In sections 3.4 through 3.6 we will evaluate a specific member of each of these families (of approaches) on the modeling deficiencies that exist for these approach instances and the remaining criteria that we have given in chapter 2.

3.4 THE EXTENDED (OR ENHANCED) ENTITY-RELATIONSHIP APPROACH

The entity-relationship approach was introduced in a seminal article of Peter Chen (1976). We will consider an ER extension as it can be found in McFadden et al. (1999:85-159) that is a main stream contemporary text book on database management. In addition we will reference those approaches from the ER literature that provide solutions for some of the modeling deficiencies that we will encounter in the McFadden approach.

3.4.1 Deficiencies in the (E)ER way of modeling

In this section we will discuss a number of problems that are related to requirements specifications that use EER as a specification language. These problems are rooted in the definition of the EER modeling constructs and in the ways in which these modeling constructs can be applied in a requirements determination process.

Ambiguities regarding the modeling of N-ary relationships

In most of the examples that are used in the articles, books or instruction manuals that give the definitions of the (E)ER modeling constructs and examples of how these modelling constructs can be applied, no explicit coverage of how to model n-ary ($N > 2$) relationships is provided: 'Higher degree relationships are possible, but they are rarely encountered in practice, ...' (McFadden et al., 1999:101). Although in some versions of the EER dialect (Connolly et al., 1996: 174-175) a number of examples of N-ary ($N > 2$) relationships are shown, McFadden et al. (1999) do not give illustrative examples of 'pure' N-ary relationships, they adapt the 'pure' N-ary relationship into either: an *associative entity* or *gerund* (McFadden et al., 1999:99-100)¹⁹ having one or more *relationship attribute* or a (N-1) ary relationship having a relationship attribute (McFadden et al., 1999:102). Furthermore, they give the conditions under which a semantic relationship can be encoded as a *gerund*. However, these conditions, presume knowledge on the cardinality constraints and the properties of the integrated EER schema, and therefore can not be applied to model the initial requirements.

The application of the N-1 relation/attribute modeling construct leads to severe problems whenever the participating 'concept type' that must be modeled as an attribute is also involved in other semantic relationships (see the discussion on the instability of EER models further on). Teorey et al. (1986:201) claim that certain relationships of a degree higher than 2 might exist in a UoD and are 'awkward' (or incorrect) when represented in a binary form and they explicitly state that: 'a ternary relationship cannot be reduced to equivalent binary relationships if the relation used to represent it is in 4NF.' (Teorey et al., 1986:202).

¹⁹ "An *associative entity* (or *gerund*) is an entity type that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between those entity instances." (McFadden et al., 1999:99-100)

No facility for naming conventions of attributes

In the (E)ER dialect in McFadden et al. (1999) there does not exist a modeling provision for the naming convention of some attributes. In figure 3.1a and figure 3.1b it is illustrated how the same piece of domain semantics must either be modeled as the ER diagram in figure 3.1a or as the ER diagram in figure 3.1b. This means that whatever option is chosen, essential domain semantics will be lost.

After analyzing a number of examples in McFadden et al. (1999:105-110) that illustrate the application of the attribute construct. This analysis reveals that a *key attribute* stands in all cases for a name class; other *attributes* can actually stand for *name classes* or *concept types*. Furthermore, it is not possible to record both the *concept type* and the *name class* in the same attribute if that is required (see for example figure 3.1 in which address is a concept type and address_ID is a name class). Most (E)ER dialects do not have guidelines on when to interpret the attribute as a name class or a concept name. A noteworthy exception is the Extended-Entity-Relationship model as defined by Engels et al. (1992) in which data types (that are user-definable) for attributes can be incorporated into the model.

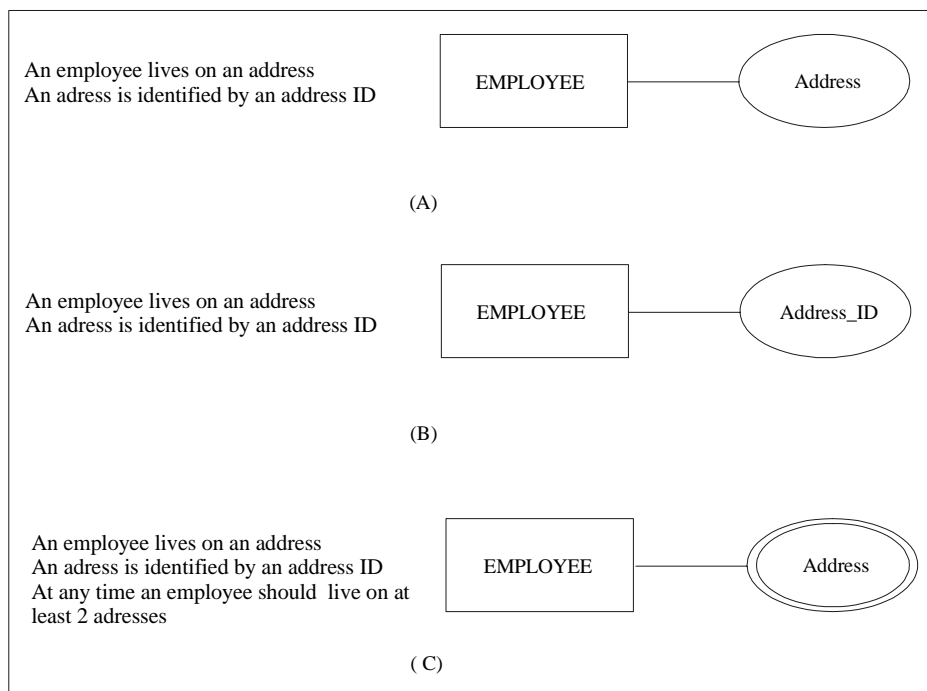


Fig. 3.1 Domain semantics and representation in EER model I

Ambiguous definition of relationship cardinalities for a ternary or higher order relationships

In the EER specification language only a small number of business rules can be modeled as static constraints. Only those business rules that can be expressed as *minimum* or *maximum cardinalities* or that can be expressed as the *multi-valued* qualification of an attribute (see figure 3.1c) can be modeled. The EER approach contains the concepts of *minimum cardinality* and *maximum cardinality*. In McFadden et al. (1999:106) the ‘look across, look across’ type of cardinality constraints is used (Dullea et al., 2003) at least for binary relationships. The application of these minimal cardinalities in ‘pure’ N-ary relationships remains unclear. A number of interpretations exist for the cardinalities that refer to a ‘pure’ ternary or higher order relationships (Halpin, 2001a). To avoid this ambiguity all ternary or higher order relationships must be converted into associative entities or gerunds (McFadden et al., 1999: 105) or binary relationships with *relationship attributes*.

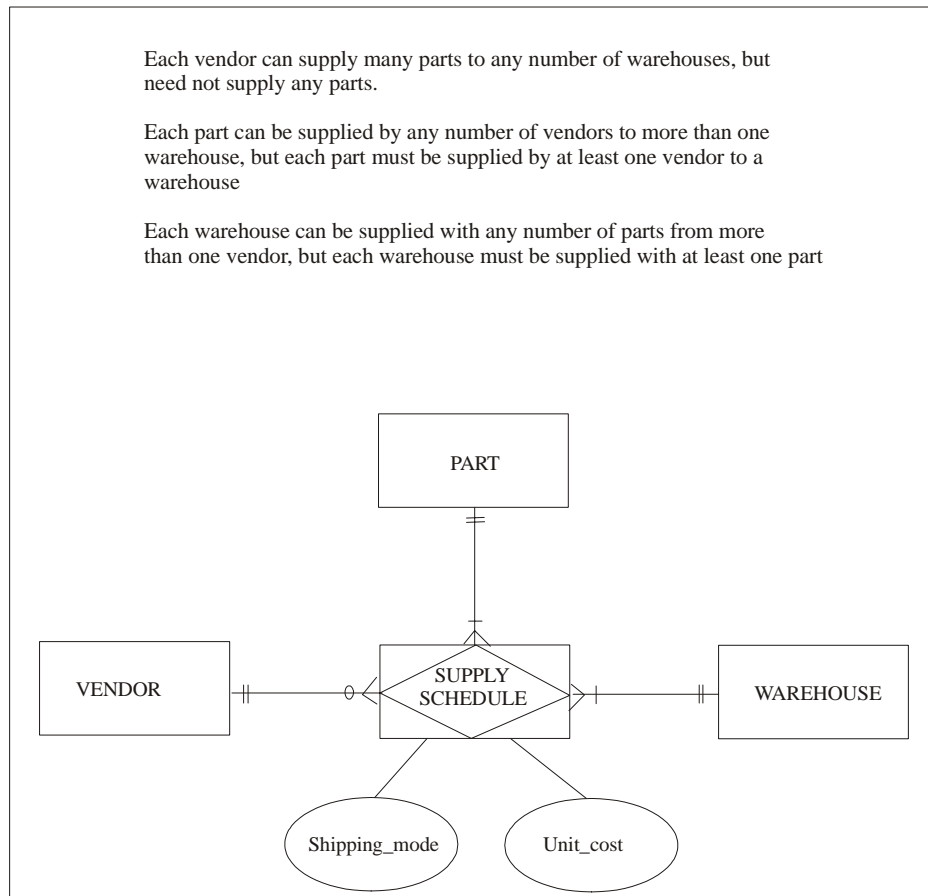


Fig. 3.2 Domain semantics and representation in EER model II (taken from figure 3.17 in McFadden et al., 1999:108)

We can see from figure 3.2 (taken from McFadden et al., 1999;108) that the relationship cardinalities in these situations represent very complicated combinations of business rules. It is not possible to represent the following simple atomic business rules as (straightforward combination) of relationship cardinalities:

A part must be supplied to at least one warehouse
A warehouse must be shipped from at least one vendor

McAllister (1998) gives an approach to check consistency of cardinalities in N-ary ER relationships by using cardinality tables. However, the number of cardinality constraints that should be analyzed as a function of the arity (N) of the relationship will increase exponentially (e.g. from 2 when N=2 to 180 for n=5).

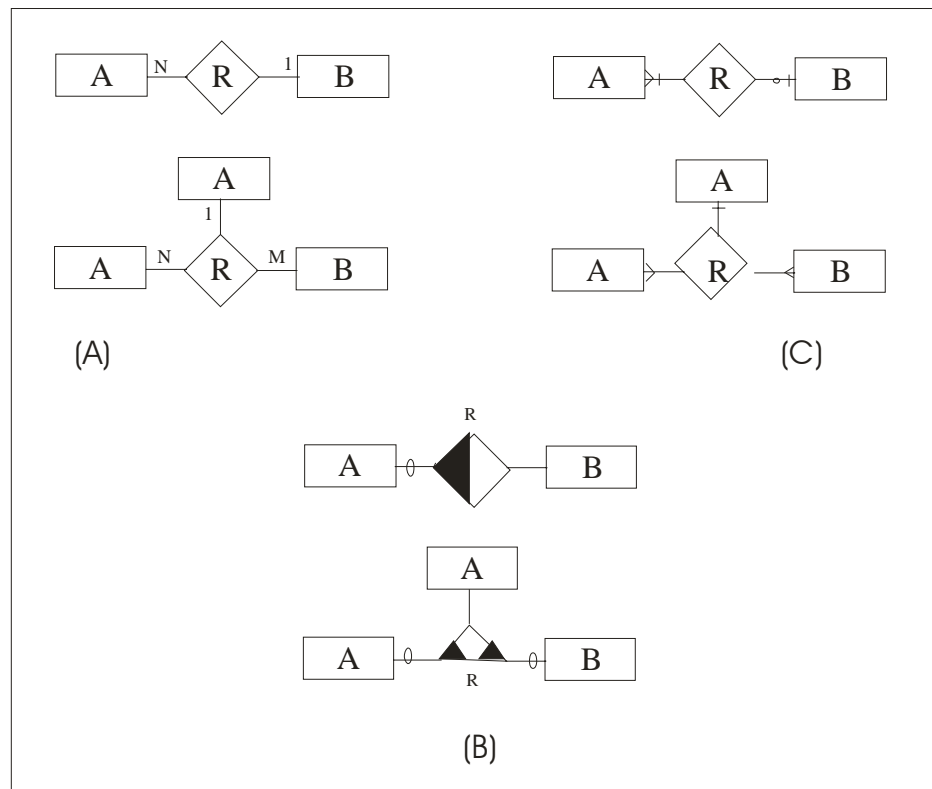


Fig. 3.3 (a) ER, (b) XER and (c) EER cardinality constraints (taken from Liddle et al. (1993) and McFadden et al. (1999))

In an overview article on cardinality constraints in semantic data models, Liddle et al. (1993) give formal definitions for relationship cardinalities in a number of ER dialects (amongst other data models) in which they show that the definition of *maximum cardinality* in the original Chen's ER dialect and the Extended ER (EER) by Teorey et al. (1986) are the same (see figure 3.3a and b) and in which the minimum cardinality

that is added to the original ER model by Teorey et al. denotes the participation status of an entity (instance) from an entity set. In this notational convention a circle that is placed on the connecting line indicates an ‘optional’ participation for the connected entities in this relationship set. For mandatory participation the line that connects the entity type to the relationship set has no special marking. We can conclude that in EER the ‘look here, look across’ type of cardinality constraints is used (Dullea et al., 2003). In the EER dialect that we have analysed in McFadden et al. (1999), the ‘look across, look across’ application of cardinality can only be found in binary relationships, no instances of N-ary ($N > 2$) relationships are given that illustrate the applicability of the minimum cardinality. We refer to Liddle et al. (1993: 239, 246) for the formal semantics of Chen’s ER and Teorey et al.’s XER cardinality constraints.

Instability of EER models because of the existence of the attribute and relationships as information bearing constructs.

The (E)ER approach shows some problems in terms of capturing evolving requirements when a binary relationship is modeled initially as an attribute of an entity type (see figure 3.4). When this modeling decision has been made in the initial stage of a project, this can lead to remodeling when additional domain semantics need to be incorporated into the application’s (E)ER-model (Bots et al., 1990; Halpin, 1996; Storey, 1991:52). In figure 3.1 we have modeled *address* as an attribute of the entity type EMPLOYEE. If we now want to model the relationship between an *address* and a *zip code* we will have a problem because simply adding a relationship will make it impossible to use the relationship cardinalities for modeling the domain semantics that every address needs to have a zip code (see figure 3.4a). In order to be able to model the domain semantics explicitly in the EER model by using cardinalities we need to remodel the original entity/attribute diagram from figure 3.1a into the relationship in the upper part of figure 3.4b. A noteworthy exception in the family of EER dialects is ERT (Theodoulidis et al., 1991) in which it is not possible to model domain knowledge as attributes. In ERT all domain semantics must be encoded as entities and/or as relationships between entities. Lim and Chiang (2000) give an overview on schema-level relationships in (E)ER diagrams.

Incomplete recording of domain semantics when encoded as relationship or attribute.

We remark that in EER it is not possible to exactly denote the sequence in which the name(s) of entity types and the name of the relationship must be read (Halpin, 2001b:315) in order to derive the correct phrasing of the semantic relationship (e.g. *address lives at employee* or *employee lives at address*). In case domain semantics are modeled as an attribute of an entity type (see for example figure 3.4 a), the ER approach does not enforce an analyst to record the verbs or predicate of such a semantic relationship. Let us assume that the upper diagram from figure 3.4a represents the following domain semantics:

Employee lives at address

We will now extend our example by a new requirement:

Employee was born at address

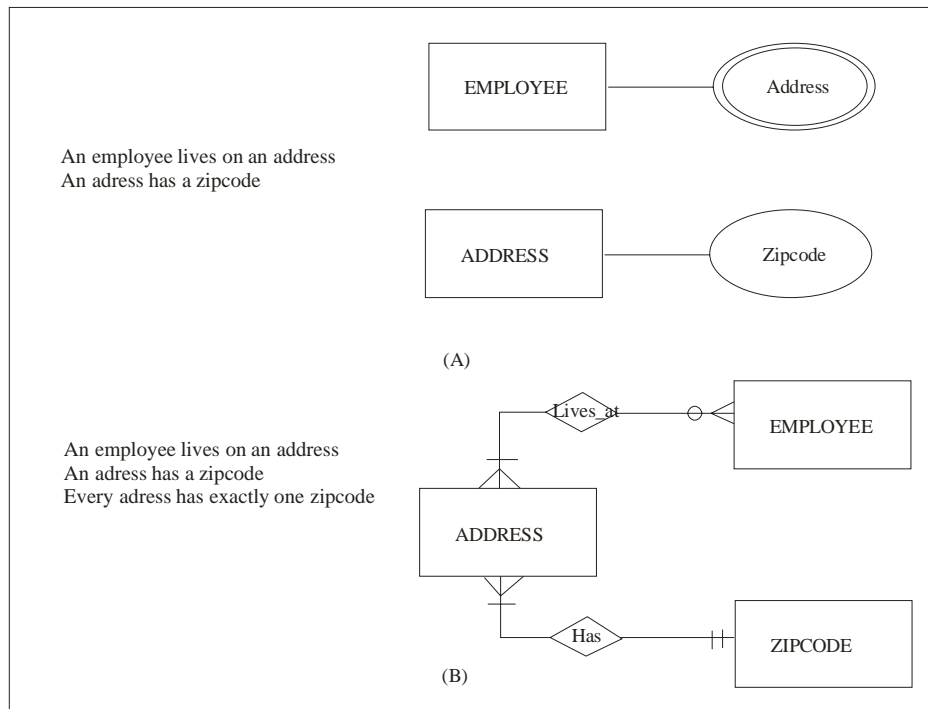


Fig. 3.4 Domain semantics and representation in EER model III

In the best situation we need to remodel the old requirements either by adding the verb on the attribute name or by creating two binary relationships in the adapted model in which the verbs can be explicitly recorded. However, in the latter situation the model will be subject to the ‘verbalization’ problems in ER diagrams that we have discussed earlier. The main problem, however, with the lacking of a verb that is recorded in an ER model, is in the interpretation of the relationship when time has passed, e.g. for application maintenance purposes.

3.4.2 Deficiencies in the (E)ER way of working

Most (E)ER family members lack a procedure that exactly specifies how an analyst can derive a semantically correct ER-model in the requirements determination process. The guidelines that Chen (1976) proposes to support the design of an ER schema are not accurate enough to be able to explain how an ER model must be created (Rolland et al., 1995:338). With respect to the static constraints we remark that apart from some participation and cardinality constraints and disjoint constraints associated with super types and subtypes (McFadden et al., 1999:145) the (E)ER approach does not provide

us with modeling facilities to do so. The same holds for the dynamic constraints and the dynamic rules.

In the EER dialect that we have analyzed (McFadden et al., 1999) no explicit procedure is given on how to apply the EER modeling concepts in a requirements determination process. Especially, the condition under which a N-ary semantic relationship must be modeled either as a ‘gerund’ or a ‘pure’ N-ary relationship are missing in the (E)ER dialects that we have studied in our survey. However, there exist some EER dialects in which modeling steps are given. We will now give a summary of three EER modeling procedures: MOODD (Silva and Carlson, 1995), EER (Teorey et al., 1986) and Storey’s EER dialect (Storey, 1991).

Modeling steps in MOODD

In MOODD a rudimentary outline of a requirements determination procedure is given (RSL) that specifies how sentences from a user requirements specification can be translated onto a Nested Entity-Relationship (NER) diagram (Silva and Carlson, 1995:163):

- Step (i) Check for synonyms and homonyms
- Step (ii) Use a glossary to ensure uniform use of words
- Step (iii) Group sentences describing the static properties of the same subjects into O-paragraphs
- Step (iv) Group sentences describing the dynamic properties of the same subject into BR (business rules) paragraphs
- Step (v) For each O-paragraph, analyze each sentence converting it to the corresponding NER object.
- Step (vi) For each BR-paragraph, analyze each sentence converting into the corresponding UPM expression

Teorey’s modeling steps

Teorey et al. (1986) give a logical design methodology for the creation of relational database schemas. The first step of this methodology, however, is directed towards the EER modeling of requirements and consists of the following sub steps:

- Step 1.1 Classify entities and attributes
- Step 1.2 Identify the generalization hierarchies and subset Hierarchies
- Step 1.3 Define Relationships
- Step 1.4 Integrate multiple views of entities, attributes, and Relationships

Teorey et al. give guidelines for classifying entities and (multi-valued) attributes but these guidelines assume knowledge of the final schema: “For example, in the above store and city example, if there is some descriptive information such as STATE and POPULATION for cities, then CITY should be classified as an entity. If only CITY-NAME is needed to identify a city, the CITY should be classified as an attribute” (Teorey et al., 1986:204). This means that such a procedure can never be applied in capturing the initial requirements of a domain user because in that stage global knowledge of the schema is not known (see also Bollen, 2002b).

Storey's modeling steps

In another EER dialect, Storey (1991) gives a procedure that covers not only the creation of a requirements specification in EER but also provides steps that result in the definition of normalized relational tables, that can serve as an input to a DDL of a database implementation. We will summarize the steps from Storey's procedure that refer to the requirements specification stage in the analysis and design process:

Step 1: Identify entities

Step 2: Identify relationships

Step 3: Check for design problems and eventually go back to step 1

Storey only specifies *what* potentially must be done in a requirements determination process, e.g. during the first step she advises to make the distinction among entity types, attributes and relationships. However, no explicit rules are given that can guide an analyst in making those modeling decisions in a specific requirements determination process.

3.5 OBJECT-ROLE MODELING (ORM)

3.5.1 Deficiencies in the ORM way of modeling

In section 3.4.1. we have illustrated some of the modeling deficiencies that exist in the EER-approach. A number of these deficiencies have been addressed in the definition of the modeling constructs in Object Role Modeling. The 'state-of-the-art' in fact oriented modeling (Halpin, 2001b); however, still has a number of modeling deficiencies that deserve attention.

Semantics of naming conventions in ORM

If we consider the example from figure 3.4 in which we have given a natural language statement of the domain requirements, the naming convention for an address in ORM is the *address name* (simple reference scheme) or a compound reference scheme that consists of three values: *street name*, *house number* and *city name* (e.g. see the discussion on signification in (Falkenberg, 1976a)). However, we have assumed that in this UoD, the addresses are restricted to one country. In case a postal service organization decides to expand its activities by taking over a foreign postal service it becomes clear that what used to be a valid signification within the country of origin now has become an *invalid* or *incomplete* signification. To avoid these problems from happening when requirements evolve it is a good practice to model these explicit semantics of naming conventions in the requirements specification at all times. An example of an explicit naming convention for an address within the Netherlands would be:

An address within the Netherlands can be identified among the union of addresses within the Netherlands by the combination of street name, house number and city name.

The existence of different referencing modes

In ORM three different ways of modeling naming conventions or reference schemes exist (Bollen, 2002b). The 1-1 referencing mode is depicted graphically by adding the name of the reference mode in parentheses to the name of the entity type that has to be referenced (Halpin, 2001b:81). We have given the ORM model for the following domain requirements in figure 3.5:

An employee works for a department within the ABC company
An employee can be identified by an employee ID
A department can be identified by a department name
An employee can work for one department at most

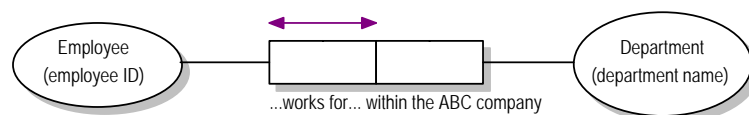


Fig. 3.5 Domain semantics and representation in ORM model I

The second way of modeling naming conventions is the case of a compound referencing scheme in which an entity of a given entity type can only be identified when using 2 or more values. In figure 3.6 we have illustrated such a compound referencing scheme (Halpin, 2001b: 192-195) for our (running) example from figures 3.2 and 3.5 in which we have changed domain semantics that allow us to identify an employee by the combination of *first name* and *last name*.

An employee works for a department within the ABC company
An employee has exactly one first name
An employee has exactly one last name
An employee can be identified by a combination of first name and last name
A department can be identified by a department name
An employee can work for one department at most

If we compare the ORM models from figures 3.5 and 3.6 we can see that the distinction between a simple and a compound reference scheme has a big impact on the resulting Object-Role Model. All other domain semantics in the example of figures 3.5 and 3.6 are identical, however, in the example from figure 3.6 we have a model that contains 3 fact types in comparison with the model from figure 3.5 in which we have only one fact type.

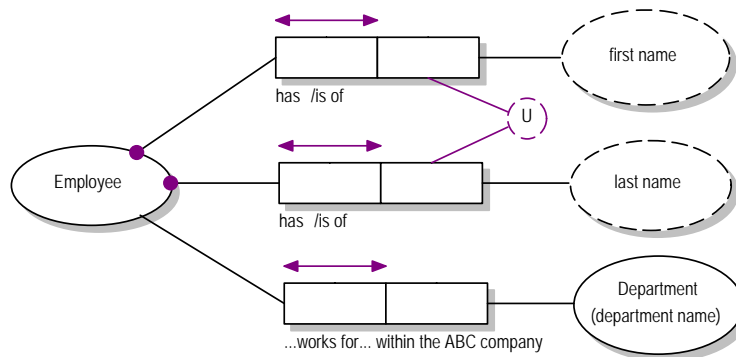


Fig. 3.6 Domain semantics and representation in ORM model II

The third way of modeling naming conventions is called objectification (Halpin, 2001b:85) in which a nested object type is modeled as a fact type in which the constituting entity types and/or name types of the objectification are given (see figure 3.7).

An employee works for a department within the ABC company
An employee can be identified by a combination of first name and last name
A department can be identified by a department name
An employee can work for one department at most

The resulting ORM diagram is given in figure 3.7.

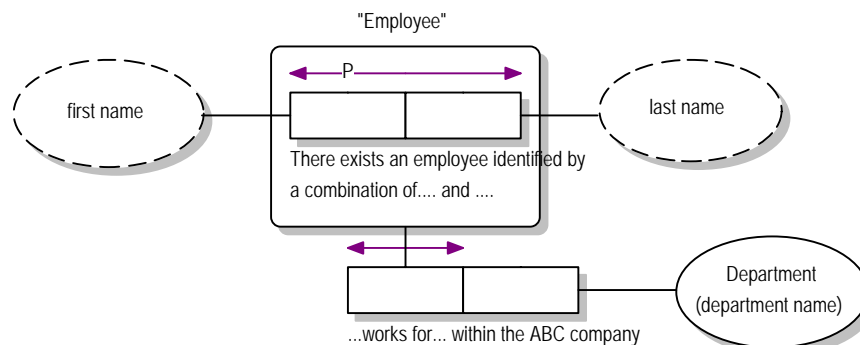


Fig. 3.7 Domain semantics and representation in ORM model III

Missing naming convention(s) for roles and/or fact types in ORM

In ORM we are *not* required to specify a role name every time a fact type is defined. This can lead to confusing situations in case the same entity type plays two or more

roles within a single fact type. Consider, for example, the following application domain semantics.

A Person introduces a person to a person
A person can be identified by a person name
A person can only be introduced once to another person

The resulting ORM diagram is given in figure 3.8.

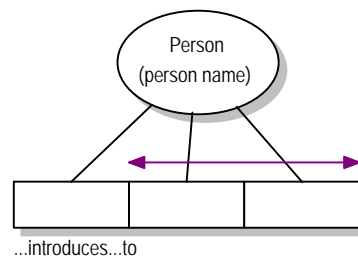


Fig. 3.8 Domain semantics and representation in ORM model IV

As we already discussed in chapter 1 of this thesis, we consider the notation legend of a requirements specification of minor importance in comparison with the modeling language concepts. If we now consider the ORM example in figure 3.8 we miss the naming conventions for the major modeling concepts in ORM: *fact types* and *roles* that would allow us to communicate the modeling results without having to use a specific notational convention, e.g. we must be able to record the modeling results in figure 3.8 in the following way:

There is a fact type that contains roles person1, person2 and person3.
Fact type template of this fact type reads as follows: <Person1>
introduces <person2> to <person3>.
Role 'person1' is played by the entity type 'Person'.
Role 'person2' is played by the entity type 'Person'.
Role 'person3' is played by the entity type 'Person'.
The name class 'Person name' is a reference type for the entity type 'Person'
There is a uniqueness constraint defined on roles 'Person2' and 'Person3'.

This means that ORM at least a simple naming convention should exist that will allow an analyst to uniquely identify a role among the union of roles or a compound reference scheme in which a role can be identified by a combination of a fact type name and the (locally unique) role name.

3.5.2 Deficiencies in the ORM way of working

With respect to the availability of a modeling procedure that guides an analyst in creating semantically correct ORM models we remark that ORM has a conceptual schema design procedure (Halpin, 2001b; Halpin and Orlowska, 1992).

Halpin's conceptual schema design procedure

In ORM a conceptual schema design procedure is defined (Halpin and Orlowska, 1992; Halpin, 2001b). This procedure consists of 7 steps:

- Step 1: From examples to elementary facts
- Step 2: Draw fact types and populate
- Step 3: Trim schema; Note basic derivations
- Step 4: Uniqueness constraints, arity check
- Step 5: Mandatory roles and logical derivation check
- Step 6: Value, Set and Subtype Constraints
- Step 7: Other constraints; Final checks

However, a close examination of this procedure in Halpin (2001b) and Halpin and Orlowska (1992) reveals that the procedure basically tells an analyst *what* to do next but does not exactly specify *how* such an activity must be carried out in a requirements determination process. With respect to steps 4, 5, 6 and 7 we must remark that ORM does not give a precise algorithm or procedure the application of which guarantees that the instances of those static and dynamic constraint types will be found in the requirements determination process.

3.6 THE UNIFIED MODELING LANGUAGE (UML)

The Unified Modeling language has its ancestors in a number of object-oriented modeling approaches (OMT (Rumbaugh et al., 1991); OOAD (Booch, 1994); OOSE (Jacobson et al., 1992)). The UML started out as a collaboration between the designers of the latter OO-methods (Kobryn, 1999:30). The UML is “a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system...it is intended for use with all development methods, lifecycle stages, application domains, and media.” (Rumbaugh et al., 1999:3). In UML the class diagram represents the data-oriented perspective of an application domain (Otero and Dolado, 2004). Bollen (2002c) has analyzed the diagrams types within UML that jointly cover the description of the information requirements as given in the criteria from chapter 2, and has found modeling problems that occur in the application of UML. He concludes that out of the 9 diagram types that are currently defined in UML (class diagrams, object diagrams, use-case diagrams, sequence diagrams, collaboration diagrams, state charts, activity diagrams, component diagrams and deployment diagrams) only the use-case diagram, class diagram, activity diagram and (advanced) state chart diagram are necessary to fulfill the completeness criterion in

section 2.1 of this thesis. Otero and Dolado (2004) conclude that 4 types of diagrams are needed to specify the behaviour-oriented aspects of systems: sequence, collaboration, state and activity diagrams. Dori (2002:83) claims that “The tight interdependence of structure and behavior mandates that these two major system aspects be addressed concurrently. This task is, however, counter-intuitive and extremely difficult if structure and behavior are forced into two (let alone nine) separate diagram types.”

Related to the problem of too many diagram types is the lack of consistency when it comes to modeling for example a state transition constraint as a state chart that constrains the states of the object that are specified in an object class diagram.

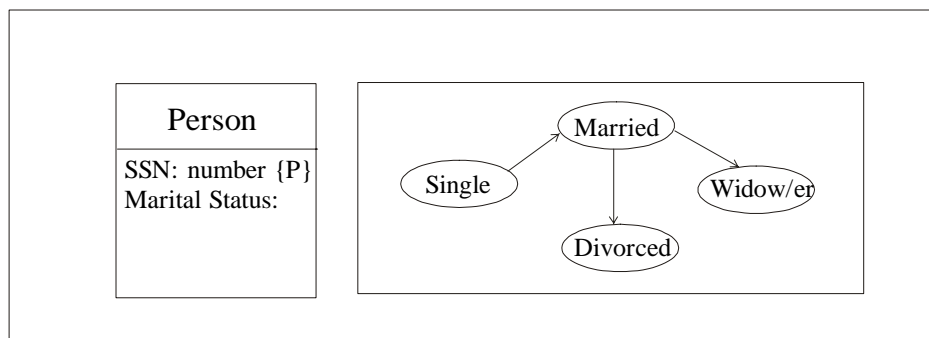


Fig. 3.9 Lack of coherence in UML class diagram and UML state chart diagram

From figure 3.9 we see that the right-hand diagram is intended to serve as a way to encode a transition rule or dynamic constraint on the subsequent values of the attribute *marital status* of the object class *Person*. UML does not give guidelines how to consistently model that the state in the state chart refer to a particular attribute of the accompanying class diagram.

In section 3.6.1 we will focus on the deficiencies in the UML specification language that are connected to the modeling constructs for the UML class diagrams

3.6.1 Deficiencies in the UML way of modeling

Although the static aspects of class diagrams share most of the modeling problems that were encountered when we analyzed the (E)ER modeling approach, UML has addressed some of them. For example, the naming conventions for attributes are implemented in the UML class diagrams as attribute types. However, there are additional modeling complications that must be taken into account when evaluating the modeling constructs in UML class diagrams and that can be fully contributed to the properties of the object-oriented paradigm, most notably the *object ID* and the interaction between the concepts of *generalization/specialization* and *class inheritance*.

The object ID in the Unified Modeling Language

In addition to the declaration of the object’s class, the declaration of the attributes and methods that an object inherits, the OO paradigm states that each object instance has a

‘unique’ identity: “Each object has its own unique identity. Most object-oriented languages automatically generate implicit identifiers with which to reference objects” (Rumbaugh et al., 1991:24). “Object identifiers must uniquely identify as many objects as may ever coexist in the system at any one time” (Cox, 1986:54).

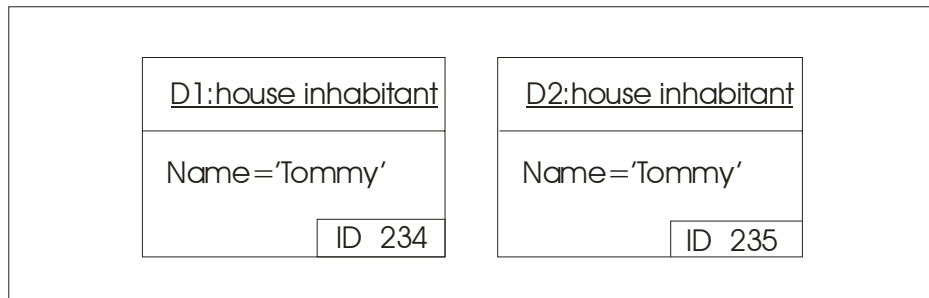


Fig. 3.10 Two different object instances of the class House Inhabitant

In UML the following definition for the object ID is given: “Each object has its own unique identity and may be referenced by a unique handle that identifies it and provides access to it.”(Rumbaugh et al., 1999:360). This concept of ‘globally’ unique object ID’s to identify objects within a specific application system allows us to make a precise distinction between two *different* objects that have the *same* state and behaviour (Dittrich, 1990:16). The existence of these object IDs allows us to refer to a house inhabitant with object ID 234 having the name Tommy and a house inhabitant with object ID 235 having the name Tommy as two different objects (see figure 3.10). It is impossible to empower users in the application domain to use ‘abstract’ object IDs as naming conventions (Halpin, 2001b:353). The best way to encode a domain-based naming convention for the concepts that are modeled as object classes is as a (combination of) class attributes. UML, however, does not provide a standard graphic notation for such a constraint. Halpin and Bloesch (1999:12) define a primary identifier constraint (‘{P}’) on the combination of attributes that can be used to identify an instance of an object class using application-based naming conventions. This means that in UML state constraints need to be applied in order to facilitate the implementation of domain-based naming conventions.

The interaction between the concepts of generalization/specialization and class inheritance.

The OO paradigm uses the same ‘is-a’ relationship for denoting specialization and generalization. In the OO-paradigm: “Generalization and specialization are two different viewpoints of the same relationship, viewed from the super class or from the subclasses. The word generalization derives from the fact that the super class generalizes the subclasses. Specialization refers to the fact that the subclasses refine or specialize the super class.” (Rumbaugh et al., 1991:42). The concept that is used in the OO paradigm for modeling generalizations is the abstract class concept in combination with the ‘is-a’ relationship.

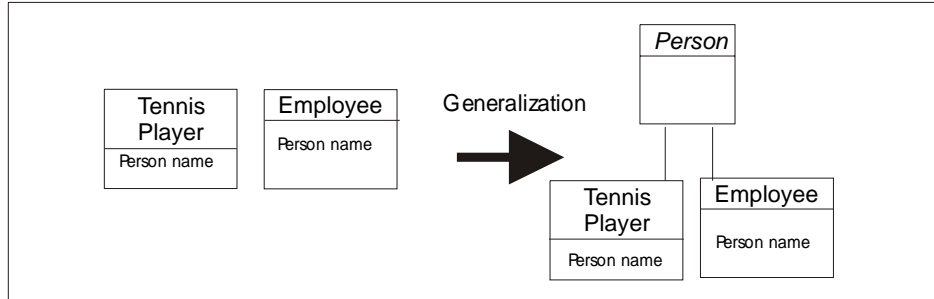


Fig. 3.11 Generalization transformation using abstract class constructs

The abstract class can be extensionally defined as the *union* of extensions of the subclasses at any point in time.

$$\text{Person} := \text{Tennis player} \cup \text{Employee}$$

One of the significant concepts in the object-oriented paradigm is the concept of inheritance. Rumbaugh et al. (1991:42) give the following description of inheritance: "...inheritance refers to the mechanism of sharing attributes and operations using the generalization relationship." Other definitions found in the literature are: "Inheritance is a code-sharing mechanism. It allows reuse of behaviour of a class in the definition of new classes. Subclasses of a class inherit the data structure and the operations of their parent class (also called a super class) and may add new operations and new instance variables." (Tkach and Puttick, 1994:21). "Inheritance is a tool for organizing, building and using reusable classes" (Cox, 1986:69). For an in-depth discussion on different types of inheritance see Rahayu et al. (2000).

In this case the class *hierarchy* is determined by clustering characteristics of the class *attributes* and *methods*. "Although many of the classes do not represent physical objects, they are conceptual entities which can be stated in the terminology of the problem domain." (Korson and McGregor, 1990:46). "The availability of an inheritance relation enables the designer to "push higher" and to identify commonality among abstractions and to produce higher level abstractions, from this commonality." (Korson and McGregor, 1990:53). Bollen (2002d) gives an example of how the aforementioned 'pushing higher' process interferes with the specialization/generalization concepts in the data perspective.

The application of the OO concept of inheritance can lead to the creation of abstractions in an object class hierarchy that do not represent things, entities or concepts in a Universe of Discourse. This type of abstraction should be modeled as an abstract object class or the conditions under which it can be modeled as a non-abstract object class should be explicitly given in a methodology for the OO-modeler. Snoeck and Dedene (1996:179-180) offer some guidelines for specializations/generalizations in object-oriented conceptual modeling.

Association end multiplicities in N-ary relationships in UML class diagrams

As we indicated earlier the static aspects of the UML class diagrams are based to a large extent on the (Extended) Entity-Relationship model. However, when the association (end) cardinalities for ternary (or N-ary in general) relationships are discussed, the defining UML literature gives specific definitions. On page 61 of the UML notation guide (Rumbaugh et al., 1999) we find the following definition of *association end multiplicity*: “Multiplicity for N-ary associations may be specified but is less obvious than binary multiplicity. The multiplicity on a role represents the potential number of instance tuples in the association when the other N-1 values are fixed.” On page 348 of the UML language reference manual (Booch et al., 1999) we find the following definition of *association end multiplicity*: “In a n-ary association, the multiplicity is defined with respect to the other n-1 ends. For example, given a ternary association among classes (A, B, C) then the multiplicity of the C end states how many C objects may appear in association with a particular pair of A and B objects. If the multiplicity of this association is (many, many, one), then for each possible (A,B) pair, there is a unique value of C. For a given (B,C) pair, there may be many A values, however, and many values of A,B and C may participate in the association.” “If the multiplicity of this association is (many, many, one)..... For a given (B,C) pair, there may be many A values, however.” This means that the upper multiplicity of many (*) defined on the association end that is connected to object class A implies there can exist many links in the object diagram for every possible (B,C) pair. It is not clear whether a lower or implied lower multiplicity for 0 in a n-ary association in UML specifies whether an object in the object class that is connected to the association end can exist independently of the association or not. In Bollen (2002a) an example is given of the ambiguity for the definition of the *lower association end multiplicity* in UML. Because of this ambiguity or ‘fuzziness’ in the definition of the lower association end multiplicity the expressiveness of this type of graphical constraint type in UML is rendered insignificant.

Verbalization of sentences for N-ary associations

The precise verbalization of the semantics of a n-ary association in UML is not possible (Halpin and Bloesch, 1998).

Default existence of object classes

In UML the modeling of semantic relationships as associations between object classes implies that instances of these object classes can exist on their own (Bollen 2002b), UML does not give guidance on how to suppress these non-existing domain semantics.

Association class and qualifier as naming conventions

Next to the identification attribute(s) that can be used to model the domain naming conventions in UML, there exist a number of alternative referencing modes: the *association class* and the *association qualifier* (Bollen, 2002b). However, the choice of a specific referencing mode can only be justified if additional domain semantics have been analyzed.

3.6.2 Deficiencies in the UML way of working

UML is a modeling language without a modeling process or procedure (Liang, 2003:83). Liang (2003) gives a procedure for mapping use cases into classes of a class diagram. In UML a requirements determination procedure is lacking (Bollen, 2002c) that specifies how the UML can be used to model the domain requirements in terms of the data model, static constraints, dynamic constraints, static derivation rules and dynamic rules. Bollen (2002c:24) proposes an outline of a modeling procedure that can be used for applying the necessary UML modeling concepts in order to model those domain semantics that are necessary for the requirements determination (see section 2.1). In the defining UML literature (Booch et al., 1999; OMG, 2002; Rumbaugh et al., 1999), however, such a rudimentary procedure outline is missing and therefore, the consistent application of UML modeling constructs can never be guaranteed.

Juristo et al. (1999:140) give an overview of research that indicates that there are no rigorous criteria for identifying the components of OO conceptual models other than procedures that contain steps that tell an analyst *what* to do, instead of *how*. See for an example Nanduri and Rugaber (1996) who took one of the predecessors to UML as their OO methodology: OMT (see Rumbaugh et al., 1991).

Rumbaugh's modeling steps

The modeling procedure that is recommended by Rumbaugh et al. (1991) and summarized in Nanduri and Rugaber (1996:10) contains the following steps:

- Step 1: Identify objects and classes (nouns)
- Step 2: Identify associations between objects (verb phrases)
- Step 3: Identify attributes of objects and associations (adjectives)
- Step 4: Identify operations (verbs and adjectives)
- Step 5: Organize and simplify object classes using inheritance
- Step 6: Iterate and refine the model

3.7 CONCLUSIONS ON THE WAY OF -MODELING, -WORKING AND - CONTROLLING FOR THE REQUIREMENTS DETERMINATION APPROACHES FROM THE LITERATURE

We now have analyzed three members of the three most prominent families (ER, fact orientation, object orientation) of requirements approaches from the literature. While analyzing specific instances of these three approaches (McFadden's EER, ORM and UML) we have discovered modeling deficiencies in each of them. Although a number of deficiencies that we, for example, have found in approach A might have been addressed in approach B, the conclusion so far is that each of these three approaches contains some deficiencies. In this chapter we will therefore summarize the extent to which any deficiency that is found in a single approach is addressed or is not addressed in at least one of the other methodologies. In addition we will indicate the extent in

which the three specific modeling approaches comply with the other criteria for requirements determination methods that we have given in chapter 2 of this thesis.

3.7.1 Overall modeling deficiencies

The (E)ER and ORM approaches basically allow an analyst to incorporate all application semantics (static and dynamic if applicable) that can be modeled by the approach into one diagram type e.g. an ER-schema or an ORM information model or information grammar. In UML there exist a multitude of diagram types in which it remains unclear what diagram types must be used for the modeling of the application system's dynamic features (Dori, 2002). However, for the main purpose of the research in this thesis we have already stated that the notations that are used by the different RDM's are of secondary importance.

3.7.2 Modeling deficiencies regarding the data model for the way of modeling

In this section we will compare the deficiencies as we have found them in the three approaches which mainly are concerned with the data model, e.g. the definition and naming of domain concepts and their semantic relationships.

Modeling facilities for n-ary relationships

ORM provides modeling support for N-ary and binary relationships. In ORM a binary relationship is a special case of a N-ary. The definitions of uniqueness and mandatory role constraints are orthogonal to the arity of the fact type(s) in the information structure diagram in ORM (this means that 'look here, look here' variant is applied for all arities).

It is possible, however, to model N-ary relationships in the EER approach and the UML class diagrams. However, only a few EER dialects, explicitly point at the necessity of a N-ary relationship concept (e.g. Teorey et al., 1986: 202; Thalheim, 2000:40). The main difference between the modeling facilities for N-ary relationship in EER and UML on one side and ORM on the other is in the dependency that exist between the application of cardinality constraints (or association end multiplicities) and the modeling of relationships/associations because some common business rules can not be expressed easily at all times in EER and UML because of ambiguities in the definition of participation cardinalities.

The existence of multiple information bearing constructs

In (E)ER and UML at least two information bearing constructs are available, in EER these are the *attribute* and *relationship*, in UML this is the *class attribute* and *association*.

In ORM the *fact type* is the only information bearing construct. An exception within the plethora of EER/OO approaches for requirements determination approaches in terms of the number of information bearing constructs, is Embley's et al. OSA (Object oriented Systems Analysis) approach (Embley et al., 1992) in which the

declarative information is represented in the Object Relationship Model. In this model the single information bearing construct is the *relationship*.

Facilities to capture precise domain semantics of naming conventions

UML and ORM provide facilities for capturing (at least) the names of the name classes.

Most (E)ER dialects lack a facility for recording name classes for concepts that are modeled as attributes at all times. All three approaches lack a way of explicitly recording the context in which the names of a name class are valid for referencing entities or concepts of a given type.

The co-existence of different referencing modes including object ID's

In ORM three ways of referencing entities exist and in UML entities can be referenced using a combination of attribute (names)²⁰ or as an association qualifier or as an association class.

In EER entities that need to be referenced by names for a name class can only be modeled as entity types in which attributes or composite attributes can be applied (McFadden et al., 1999: 219). In most EER dialects gerunds can be defined which is similar to association class construct in UML.

Facilities for specification of how to communicate semantic relationships in data models

In EER and UML there's no facility for verbalizing the relationships that are modeled as N-ary relationships in a precise and unambiguous way. In UML verbalization into sentences is only possible for binaries associations that use an optional marker (Halpin and Bloesch, 1999:8). In ORM these facilities exist for all semantic relationships in an application domain.

Naming conventions for elements/concepts in data models

If we want to communicate the content of the data model in a way that is 'diagram-free' we miss naming conventions for the roles and/or fact types in ORM. Furthermore, the optionality of the *role* concept in EER and UML can lead to additional application model verbalization problems. This severely impacts the traceability of the requirements documents in ORM, EER and UML.

The facility to capture the precise generalization/specialization semantics.

In most extended ER approaches (Balaban and Shoval, 2002; Kolp and Zimanyi, 2000; Scheer and Hars, 1992; Silva and Carlson, 1995; Teorey et al. 1986; Theodoulidis et al., 1991) and ORM (Halpin, 2001b) modeling constructs are defined that enable the

²⁰ Such a combination of attribute names, however, needs a static constraint that specifies that thi(e)s(e) attribute(s) can serve as a reference type. This means that in UML, domain based naming conventions are encoded as static constraints.

analyst to model the specialization/generalizations relationships that exist in the application domain.

In UML, however, it is possible to create ‘inheritance’ trees in which the generalizability of methods determines a specialization/generalization hierarchy other than is justified by the domain ontology.

3.7.3 Modeling deficiencies regarding the static constraints for the way of modeling

Extent in which business rules can be modeled as static constraints

The business rules that can be modeled as static constraints in EER reflect those domain semantics that can be encoded as cardinalities in binary relationships. The ER+ dialect in addition contains a subset constraint (Kolp and Zimanyi, 2000), Rochfeld and Negros (1992) define a range of inter-relationship constraints in their ER dialect; *inclusive FIC*, *exclusive FIC*, *simultaneity constraint*.

In UML this is extended to include attribute multiplicities. Furthermore UML has the facility to model some types of *exclusion* and *subset* constraints. Furthermore UML has the object constraint language (OCL) that enables it to model a wide range of domain semantics.

ORM offers the most pre-defined graphical static constraint types for encoding business rules.

Interpretation of cardinality constraints/association end multiplicities

The minimum relationship cardinalities and/or association end multiplicities in many EER dialects and in UML, especially for $N > 2$ are not or at best ill-defined. In EER a number of interpretations exist for cardinality constraints. Dullae et al. (2003) give two archetypes of interpretations (‘look here, look across’(LELA) and ‘look across, look across (LALA)). In the EER flavor that we have analyzed in this thesis (McFadden et al, 1999: 85-165) we have the LALA variety for binary relationships. However for N-ary relationships the cardinality semantics totally change. McFadden et al. use two ways for encoding N-ary domain semantics: a N-ary relationship as an associative entity of arity ($\leq N$) having 1 or more relationship attributes or as a binary having at least 1 relationship attribute. However, for those application areas in which it is not possible to identify a concept in the application domain as gerund and in which it is not possible to use relationship attributes, the interpretation of the minimum cardinalities for such a ‘pure’ N-ary (EER) relationship remains ambiguous and the fact that the ER approach is used for the creation of requirements specifications, does not give any guidance in how to interpret cardinalities (see figure 3.3).

The multiplicity constraints on association ends defined in UML specify any range of occurrence frequencies applied to a single role for binaries (for N-aries, such a range indicates what occurrence numbers are possible when the other $n-1$ classes have a fixed value). ORM partitions this multiplicity concept into the orthogonal constraint types: *mandatory role constraints* and *frequency constraints* (Halpin and Bloesch, 1999:11).

Default existence constraints

In UML entities or objects are allowed to ‘exist’ independently of the relationships they are involved in (Bollen, 2002b:133).

In EER entity types are strong by default (McFadden et al., 1999: 92-93) which means that they are allowed to exist independently of the relationships they are involved in.

In ORM entity types are *not* allowed to exist independently by default. Bollen (2002b) concludes that in UML and for the same reason in EER when a (binary or higher order) semantic relationship is modeled, unary relationships that declare the existence of entities or objects are modeled at the same time. This means in practice that to be able to model such a (binary or higher order) relationship (on its own) the analyst has to declare in EER that the entity type that is not allowed to exist independently is assigned the status weak (Kolp and Zimani, 2000: 1059; Tsichritzis and Lochovsky, 1982:182) and in UML a textual constraint must be attached that states that each instance should at least participate in one of the relationships (Bollen, 2002b:133). We note that in an evolving requirements specification this implies that such a constraint must again be specified whenever a new relationship in which the object class participates is added to the EER diagram or UML class diagram.

3.7.4 Modeling deficiencies regarding the dynamic constraints for the way of modeling

In this paragraph we will compare the EER model, the UML and ORM on the facilities that they provide for modeling *dynamic constraints*. We will use a number of subclasses of dynamic constraints that can be found in De Brock (2000). De Brock makes a distinction into subclasses of dynamic constraints. Prabhakaran and Falkenberg (1988) give modelling constructs for transition oriented constraints (TOC) in NIAM.

Cumulativity of tuples, key attribute value combinations, attribute value combinations

In the terminology of the application information base this cumulativity requirement expresses that every fact that has been entered into the application’s information base should stay in the application’s information base. In EER no provision for such a domain rule exist, in UML the changeability qualification can be defined on an attribute or association end of binary associations (Halpin, 2001b:393) and be assigned the value *add Only* (Rumbaugh et al., 1999: 166, 184). In ORM changeability constraints are not supported (Halpin, 2001b:395).

Non-decreasing attribute values and non-decreasing number of tuples

These constraint types can not be specified in EER and ORM.

Integrity constraints on initial values

UML may assign initial values to attributes, EER and ORM (Halpin, 2001b:390) do not support this.

Life cycles

UML supports this in the form of *state charts* and ORM uses a *state transition fact type* in which the graphs in the life cycle can be captured as data (Halpin, 2001b:299). EER does not support this type of constraint.

Changing Life cycles

UML supports this in the form of state charts, but a change in life cycle implies remodeling. ORM uses a state transition fact type in which the graphs in the life cycle can be captured as data (Halpin, 2001b:299) and therefore changes in the life cycle can be implemented on an information base level. EER does not support this type of constraint.

3.7.5 Modeling deficiencies regarding the static derivation (rules) for the way of modeling

In the specific EER dialect that we have studied (McFadden et al., 1999) only provisions are given for static derivation (rules) that refer to derived attributes. Furthermore these derived attributes are restricted to those that can be derived from other attributes (McFadden et al., 1999:95). It remains unclear whether derived attributes that partly need relationships instances as an input should be signified. In most cases, however, no modeling constructs in EER are given that allow us to model a precise specification of a derivation rule. Rauh and Stickel (1996) give an extension to the ER approach called ERM_{ded}, which contains modeling constructs for derivation rules.

In ORM, derivation rules are written as text below the diagram (Halpin, 2001b, 97). We note that derivation rules should contain explicit references to roles in the information structure diagram. We note however that the data structure of a derived fact type is not required to be contained in the diagram (Halpin, 2001b:99) but if it is, it must be distinguished from the base diagram by an asterix (Halpin, 2001b:100).

In UML a static derivation (rule) is modeled as a *derived element* (i.e. a *derived attribute* or a *derived association*) (Rumbaugh et al., 1999:254-255). We note that in UML, the derived attribute or association is included in the class diagram and the derivation rule is specified and included in the class diagram. Furthermore, UML allows us to specify (the more complicated) derivation rules (in terms of the number of classes and relationships involved in an activity diagram (Bollen, 2002c:23))

3.7.6 Modeling deficiencies regarding the dynamic rules for the way of modeling

The EER dialect in McFadden et al. (1999) does not provide facilities for the modeling of dynamic rules. Gorman and Choobineh (1991) and Silva and Carlson (1995) do provide an object-oriented extension to ER that facilitates the modeling of dynamic rules.

In ORM (Halpin, 2001b; Halpin and Orłowska, 1992) no facilities are given for modeling dynamic rules. Prabhakaran and Falkenberg (1988:100) introduce trigger-precedent-consequent triplets in combination with a NIAM conceptual schema.

UML provides modeling facilities for the encoding of event-condition action constraints. Bollen (2002c) states that an advanced state chart in UML can be used to model the event-condition action constraints (Rumbaugh et al., 1999: 447-448).

3.7.7 Modeling deficiencies regarding the way of working and way of controlling

In the EER and OO families of requirements modeling languages, some authors have tried to define a modeling procedure; however these procedures basically specify *what* an analyst should do rather than prescribing *how* these steps must be performed. In combination with the choices that are inherent to the multitude of information bearing constructs in EER and UML, these procedures are prone to a ‘deadly embrace’ in terms of the knowledge on the end result that must be available before the initial requirements can be modeled.

Another deficiency in many EER dialects is that the requirements specification that is expressed in such an EER diagram is not complete in terms of domain semantics. In some approaches subsequent steps are given that should transform the requirements specification into an implementation schema, e.g. a relational schema. This, however, means that in this transformational stage from a requirement specification into a design specification domain, still domain knowledge needs to be ‘injected’ to determine the appropriate functional dependencies (see Teorey et al. (1986) and Ram (1995)).

In ORM *all* semantics regarding functional dependencies are incorporated in the information model or conceptual schema. In the fact-oriented approach, the conceptual schema design procedures in Halpin and Orlowska (1992) and Halpin (2001b), however, do *not* specify *how* the instances of the pre-defined constraint types can be instantiated at all times.

3.7.8 Summary of Modeling deficiencies in the EER, ORM and UML approaches

In table 3.3 we have summarized the deficiencies from the three approaches studied. A ‘+’ denotes that an approach does not have this language or procedure deficiency. A ‘0’ means that an approach has this deficiency to some extent. A ‘-’ means that an approach has this deficiency to the highest extent.

Table 3.3 Summary of the comparison of EER, ORM and UML approaches on modeling deficiencies.

| Modeling deficiencies | | EER | ORM | UML |
|----------------------------|--|-----|-----|-----|
| REQUIREMENTS | LANGUAGE/PROC. Deficiency | | | |
| Data Model | Facilities for n-ary relationships | 0 | + | 0 |
| | Facilities for sem. of naming conventions | - | 0 | 0 |
| | Existence of multiple information bearing constructs | 0 | + | - |
| | Facilities for naming conventions of modeling concepts | - | - | 0 |
| | Co-existence of different reference modes | 0 | - | - |
| | Facilities for capturing verbs in data models | - | + | 0 |
| | The facility to capture the precise generalization/specialization semantics | + | + | 0 |
| Static constraints | Extent in which static constraints can be modeled | 0 | + | + |
| | Interpretation of cardinalities/aem's | - | + | - |
| | Default existence constraints | - | + | - |
| Dynamic constraints | Cumulative of value combinations | - | - | + |
| | Non-decreasing values | - | - | - |
| | Integrity constraints on initial values | - | - | + |
| | Life cycles | - | + | + |
| | Changing life cycles | - | + | 0 |
| Static Derivation | | 0 | + | + |
| Dynamic Rules | | - | - | + |

3.8 THE SUITABILITY OF EXISTING APPROACHES FOR REQUIREMENTS DETERMINATION WITH RESPECT TO THE COMPLETENESS-, DOMAIN RICHNESS, EFFICIENCY AND FORMALITY CRITERIA

In chapter 2 we have defined four criteria that can be considered relevant in the context of requirements determination. In this section we will summarize the findings from the literature survey on existing requirements determination approaches with respect to these four criteria: *domain richness*, *completeness*, *efficiency*, and *formality*.

With respect to the *domain richness* criterion we remark that this criterion contains a number of dimensions. The dimension *perception* refers to the extent in which different domain users have a different perception of an underlying reality. This means that the application of a requirements determination method must lead to a requirements specification that reflects the (possibly) different perceptions of an underlying reality by different user groups. It is possible to reflect these difference perceptions by using the EER, UML and ORM approaches, whenever they are embedded in a procedure that enables an analyst to integrate the different views from

different user groups on the ‘underlying reality’ by integrating the sub-schemas of these users into a final ‘overall’ requirements specification in which the different perceptions are made explicit. So far, the EER, UML and ORM approaches that we have discussed in this chapter do not give provisions for this, however, Hayne and Ram (1995:100-101) report on a design checking tools (EasyER, GAMBIT and DDEW) for (E)ER models.

The dimension *turbulence* characterizes the extent in which an application domain is subject to changes in the business data and business rules. While discussing the characteristics of the *data model* and the *static constraints* in the EER, UML and ORM approaches we have remarked that there is an interaction between the definition of set of modeling constructs and the extent in which a specification has to be remodeled when requirements are added to the model or change in general. We concluded that the EER and UML approaches are most prone to remodeling because of the multitude of information bearing constructs (Halpin and Bloesch, 1999:8). ORM addresses those issues mentioned but has a problem with a multitude of naming conventions which might lead to unstable models.

With respect to the dimension *tacitness*, we can say that the EER, UML and ORM approaches basically have the assumption that users will be able to express their initial requirements in natural language, e.g. in a way that the data model, (static and dynamic) constraints and static derivation and dynamic rules can be written down in a requirements document. This restricts the applicability of these approaches to those domains that exclusively contain explicit knowledge. However, we think that a requirements determination methodology must be able to capture (at least some of) those tacit business rules that are implicit but that can be made explicit in the terminology of Kim et al. (2003).

With respect to the dimension *anchoring* we can say that the requirements determination process in which we use EER and UML models for our specification language are in principle not limited to any specific range on the anchoring scale. ORM is anchored in familiar examples or data use cases (Halpin, 2001b:60) and it requires the domain expert to come up with these real examples and therefore is applicable for those domains that are on the ‘tangible’ side of the anchoring scale (see chapter 2).

A brief conclusion regarding the suitability of the three approaches that we have studied in this chapter of this thesis is that the EER and ORM approach do not comply to the *completeness* criterion for the way of modeling that was defined in chapter 2 and that contains a description of what type of ‘domain knowledge’ in essence must be incorporated in a requirements specification. Furthermore, there exists a large difference between the families of approaches and even between members within a given family in terms of the extent in which the application domain semantics can be expressed in the *data model*, and as *static* or *dynamic constraints*, *static derivation rules* or *dynamic constraints*. With respect to the completeness criterion for the way of working we can conclude that ORM is the only approach that provides some assurance that all relevant semantic relationships in the data model and some types of static constraints will be detected in the application UoD. This means that there still is an opportunity to improve the requirements determination approaches we have surveyed in this chapter in terms of the *completeness* aspects that were given in section 2.1.

With respect to the *efficiency* criterion for the way of modeling we must remark that in EER and UML in a number of cases remodeling is necessary not

because domain semantics have changed, but because the attribute modeling construct has been applied in the initial requirements specification. With respect to the efficiency in the way of working we concluded that in some species of the family of EER approaches modeling procedures do exist. However, they rather tell an analyst what to do next than to specify how he/she must do it. In ORM a CSDP (conceptual schema design procedure) is given that gives more guidance on how an analyst must apply the modeling concepts than in the EER counterparts. However, with respect to the derivation of static constraints ORM does *not* give a procedure that specifies how an analyst can find *all* instances of such a constraint type in a given UoD. In the defining literature of UML no (rudimentary) procedure is provided that tells an analyst how to detect instances of constraints (in a dialogue with a domain expert). With respect to the efficiency in the way of controlling we must conclude that none of the approaches (EER, ORM and UML) provides quality assurance steps and the EER and UML approaches do not provide an activity structure that gives handles for optimizing performance, cost and time.

With respect to the *formality* criterion for the way of modeling we can conclude that in many (E)ER approaches and in the UML, it is not possible to apply a consistent definition for minimum cardinalities or multiplicities across all types of semantic relationships. In UML it is not clear how the modelling concepts that are used in the 9 different diagram types are related on the level of an application requirements specification. In ORM an inconsistency is found with respect to naming conventions. Furthermore, we remark that the optionality or non-existence of some modeling constructs in all three approaches that we've studied might lead to imprecise and inconsistent requirements specifications. The non-required naming conventions for model elements in EER, UML and ORM can lead to traceability problems.

With respect to the *formality* criterion for the way of working we can conclude that for EER and UML the formality of the procedure is not relevant because there hardly exists any procedure. With respect to the CSDP in ORM we remark that those segments of the CSDP that can be considered prescriptive documents are at most semi-formal. In EER, UML and ORM no procedure exists that allows an analyst to question the assumptions on which the utterance of the domain semantics is based.

Table 3.4 Comparison EER, ORM and UML approaches on completeness, domain richness, efficiency and formality criteria for the way of modeling, way of working and way of controlling

| | | EER | | | | ORM | | | | UML | |
|-----------------|-------------|-------------|-------------|--|-------------|-------------|-------------|--|-------------|-------------|-------------|
| | W o M | W O W | W o C | | W o M | W o W | W o C | | W o M | W o W | W o C |
| Completeness | - | - | n. a. | | 0 | 0 | n.a. | | + | - | n.a. |
| Domain Richness | - | + | n. a. | | 0 | - | n.a. | | - | 0 | n.a. |
| Efficiency | - | 0 | - | | 0 | 0 | - | | - | - | - |
| Formality | - | - | - | | - | - | + | | - | - | - |

The position of these approaches is basically that the domain requirements that are uttered by the user are encoded in the model 1-on-1. ORM claims in steps 3, 4 and 7 to perform checks on sample populations; however, it does not give guidelines on how to perform these checks in a dialogue with the responsible domain user.

With respect to the formality criterion for the way of controlling we must conclude that there exist no formal quality assurance algorithms in EER, UML and ORM. Finally we can conclude that ORM is the only approach that has facilities for formally planning a requirements determination project in terms of the stages in the conceptual schema design procedure (CSDP).

In table 3.4 we have summarized the deficiencies from the three approaches studied. A '+' denotes that an approach does fully comply with this criterion for the given aspect of the RDM. A '0' means that an approach complies to some extent. A '-' means that an approach does not comply at all.

After studying the existing literature on RDM's we can conclude that no single approach fulfills the criteria that were given in chapter 2. To put it even stronger: even a compilation of approaches in which the best features of a number of approaches will be combined, will not comply with the four quality criteria from chapter 2. In chapter 4 we will use the flaws, inconsistencies and omissions that we have diagnosed in the state-of-the-art in RDM's when we operationalize the quality criteria from chapter 2 into an operationalized design specification for a to-be designed RDM.

3.9 REFERENCES

- Abrial, J. (1974): Data Semantics. In: Klimbie, J., Koffeman, K. (eds.): Data Base Management, North Holland, Amsterdam. 1-59
- Akoka, J., Comyn-Wattiau, I. (1996): Entity-relationship and object-oriented model automatic clustering. *Data & Knowledge Engineering* **20**: 87-117
- Balaban, M., Shoval, P. (2002): MEER-An EER model enhanced with structure methods. *Information Systems* **27**: 245-275
- Bollen, P. (2002a): Using Object-Role modeling for capturing user requirements expressed as UML class diagrams., In: Callaos, N., Hernandez-Encinas, L., Yetim, F. (eds.) *Proceedings 6th world conference on Systemics, Cybernetics and Informatics*, Orlando, Florida, volume I, Information systems development I, p. 305-310.
- Bollen, P. (2002b): A formal Transformation from Object Role Models to UML class Diagrams. In : Halpin, T., Siau, K., Krogstie, J. (eds.) *Proceedings of the 7th CAISE/IFIP-WG8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*. Toronto, Canada p. 132-143.

- Bollen, P. (2002c): De totale samenhang tussen de diagramsoorten in UML. Meteor research memo RM 02/048. Faculty of Economics and Business Administration. University of Maastricht.(in Dutch)
- Bollen, P. (2002d): Using the OO paradigm for conceptual modeling: the need for a methodology', In: M. Hunter and K. Dhanda (Eds.) *proceedings ISONeworld 2002*, Las Vegas, U.S.A.
- Booch, G., (1994): Object-oriented analysis and design with applications, 2nd edition, Benjamin-Cummings.
- Booch, G., Rumbaugh, J., Jacobson, I. (1999): The unified modeling language user guide, Addison-Wesley
- Bots, J., van Heck, E., van Swede, V., Simons, J. (1990): Bestuurlijke Informatiekunde; Een praktisch studie- en handboek voor de mondige gebruiker van informatiesystemen. Cap Gemini publishing/Pandata b.v. (in dutch)
- Breutmann, B., Falkenberg, E., Mauer, R. (1979): CSL- A language for Defining Conceptual Schemas. In: Database Architecture. Bracchi, G. et al. (eds.). North-Holland
- Brown, A.W. (1991), Object-oriented databases: their applications to software engineering, McGraw-Hill.
- Bubenko, J., Wangler, B. (1992): Research Directions in conceptual specification developments. In: Conceptual Modeling, Databases, and Case (Loucopoulos, P., Zicari, R. (eds.)). Wiley. 389-412
- Chan, H., Poo, D., Woon, C. (1998): An object-oriented implementation of an entity relationship model. *Journal of Systems and Software* **41**: 117-225
- Chen, P. (1976): The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database systems* **1** (1): 9-36
- Connolly, T., Begg, C., Strachan, A. (1996): Database Systems: a practical approach to design, implementation and management. Addison-Wesley
- Cox, B.J., (1986): Object-oriented programming. Addison-Wesley.
- Davis, A. (1995): Object-Oriented Requirements to Object-Oriented design: An Easy Transition ?. *Journal of Systems and Software* **30**: 151-159.
- De Brock, B. (2000): A General Treatment of Dynamic Integrity Constraints. *Data & Knowledge Engineering* **32**: 223-246
- DeMarco, T. (1978): Structured analysis and System Specification. Prentice-Hall.

- Deng, P-S., Fuhr, C. (1995): Using an object-oriented approach to the development of a relational database application system. *Information & Management* **26**: 107-121
- Dittrich, K. (1990): Object-oriented database systems: the next miles of the marathon. *Information Systems* **15**(1): 161-167
- Dori, D. (2002): Why significant UML change is unlikely. *Communications of the ACM* **45**(11): 82-85
- Dullea, J., Song, I-Y., Lamprou, I. (2003): An analysis of structural validity in entity-relationship modelling. *Data & Knowledge Engineering* **47**: 167-205.
- Embley, D., Kurtz, B., Woodfield, S. (1992): *Object-Oriented Systems Analysis: A Model Driven Approach*. Prentice-Hall
- Engels, G., Gogolla, M., Hohenstein, U., Hulsmann, K., Lohr-Richter, P., Saake, G., Ehrich, H-D. (1992): Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering* **9**: 157-204
- Falkenberg, E. (1976a): Concepts for Modelling Information. In: Nijssen, G. (ed.), *Modelling in Database Management Systems*, North-Holland, Amsterdam ,pp.95-109
- Falkenberg, E. (1976b): Significations: the key to unify data base management. *Information Systems* **2**: 19-28
- Falkenberg, E., Nijssen, G., Adams, A., Bradley, L., Bugeia, P., Campbell, A., Carkeet, M. Lehmann, G., Shoesmith, A. (1983): Feature Analysis of ACM/PCM, CIAM, ISAC and NIAM. In: Olle, T., Sol, H. and Tully, C. (eds.) *Information systems design methodologies: a feature analysis*. IFIP. North-Holland. 169-190.
- Floyd, C. (1986): A comparative evaluation of system development methods. In: Olle, T., Sol, H., Verrijn-Stuart, A. *Information Systems Design Methodologies: improving the practice*. North-Holland
- Gane, C., Sarson, T. (1979): *Structured systems-analysis: tools and techniques*. Englewood Cliffs, NJ, Prentice-Hall.
- Gogolla, M., Hohenstein, U. (1991): Towards a semantic view of an Extended Entity-Relationship Model. *ACM transactions on database systems* **16** (3): 369-416
- Griethuysen, J. van, (ed.). (1982): *Concepts and terminology for the Conceptual Schema and the Information Base*, Report of ISO TC97/SC5/WG3.
- Gustafsson, M., Karlsson, T., Bubenko, J. (1982): A declarative approach to conceptual information modeling. In: Verrijn-Stuart, A., Olle T., Sol H., (eds.): *Proceedings of IFIP TC-8 Conference on Comparative Review of Information Systems Methodologie (CRIS-1)*, North- Holland Amsterdam.

Halpin, T. (1996): Business Rule and Object Role modeling. Database Programming & Design. October.

Halpin, T. (2001a): Augmenting UML with Fact-orientation , in:workshop proceedings: UML: a critical evaluation and suggested future, HICCS-34 conference.

Halpin, T. (2001b): Information Modeling and Relational Databases, Morgan Kaufmann Publishers

Halpin, T., Bloesch, A. (1998): A comparison of UML and ORM for data modelling. In Proceedings of the 3th CAISE/IFIP-WG8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design. Pisa. Italy.

Halpin, T., Bloesch, A. (1999): Data modeling in UML and ORM: a comparison. Journal of Database Management **10**(4): 4-13

Halpin, T., Orlowska, M.(1992): Fact-oriented Modelling for Data Analysis. Journal of Information Systems **2**: 97-118

Hammer, M., McLeod, D. (1981): Database Description with SDM: A Semantic Database Model, ACM Transactions on Database Systems **6**(3): 351-386

Hanani, M., Shoval, P. (1986): A combined methodology for information systems analysis and design based on ISAC and NIAM. Information Systems **11**(3): 245-253.

Hayne, S., Ram, S. (1995): Group Data Base Design: Addressing the View Modeling Problem. Journal of Systems and Software **28**: 97-116

Henderson-Sellers, B., Edwards, J.M. (1990): The object-oriented system's life cycle. Communications of the ACM **33**(9):143-159

Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G. (1992): Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley.

Juristo, N., Morant, J., Moreno, A. (1999): A formal approach for generating OO specifications from natural language. Journal of systems and software **48**: 139-153.

Kim, Y-G., March, S. (1995): Comparing Data Modeling Formalisms. Communications of the ACM **38**(6): 103-115

Kim, T-G., Yu, S-H., Lee, J-W. (2003): Knowledge strategy planning: methodology and case. Expert Systems with Applications **24**(3): 295-307

Kobryn, C. (1999): UML 2001: A standardization odyssey. Communications of the ACM **42**(10): 29-37

- Kolp, M., Zimanyi, E. (2000): Enhanced ER to relational mapping and interrelational normalization. *Information and Software Technology* **42**: 1057-1073
- Korson, T., McGregor, J., (1990): Understanding object-oriented: a unifying paradigm. *Communications of the ACM* **33**(9): 41-60.
- Leung, C., Nijssen, G. (1988): Relational Database design using the NIAM conceptual schema. *Information Systems* **13**: 219-227
- Liang, Y. (2003): From uses cases to classes: a way of building object model with UML. *Information and Software Technology* **45**: 83-93
- Liddle, S., Embley, D., Woodfield, S. (1993): Cardinality constraints in semantic data models. *Data & Knowledge Engineering* **11**: 235-270.
- Lim, E-P., Chiang, R. (2000): The integration of relationship instances from heterogeneous databases. *Decision Support Systems* **29**: 153-167
- Lundeberg, M., Goldkuhl, G., Nilsson, G. (1979): A systematic approach to information systems development, *Information Systems* **4**: 1-12, 93-118.
- Lundeberg, M. (1982): The ISAC Approach to Specification of Information Systems and its Application to the Organization of an IFIP Working Conference. In: Olle et al. (eds.), *Information System Design Methodologies- a comparative review*, North-Holland, pp 173-234.
- Maarssen, L., McGowan, G. (1986): *Structured Analysis and Design Technique (SADT)*. Methodieken voor informatiesysteemontwikkeling, 1983. NGI (in dutch)
- McAllister, A. (1998): Complete rules for n-ary relationship cardinality constraints. *Data & Knowledge Engineering* **27**: 255-288
- McFadden, F., Hoffer, J., Prescott, M. (1999): *Modern Database management*, 5th edition, Addison-Wesley.
- Meyer, B. (1988): *Object-oriented Software Construction*. Prentice Hall
- Misic, M., Graf, D. (2004): Systems analyst activities and skills in the new millennium. *Journal of systems and software* **71**: 31-36.
- Montgomery, S. (1994): *Object-oriented information engineering: analysis, design and implementation*, Academic press limited
- Mylopoulos, J., Chung, L., Yu, E. (1999): From object-oriented to goal-oriented requirements analysis. *Communications of the ACM* **42**(1): 31-37
- Nanduri, S., Rugaber, S. (1996): Requirements Validation via Automated Natural Language Parsing. *Journal of Management Information Systems* **12** (3): 9-19

Nijssen, G. (1977): On the Gross management for the next generation database management systems. In: Gilchrist,B. (ed.): Information Processing 77 IFIP, North-Holland, Amsterdam, pp.327-335

Nijssen, G., Halpin, T. (1989): Conceptual schema and relational database design: A fact based approach, Prentice-Hall, Englewood Cliffs.

Olive, A. (1982): DADES- a methodology for specification and design of information systems. In: Verrijn-Stuart,A., Olle T., Sol H., (eds.): Proceedings of IFIP TC-8 Conference on Comparative Review of Information Systems Methodologie (CRIS-1), North- Holland Amsterdam.

OMG, (2002). UML Specification v. 2.0.

Opdahl,A., Sindre, G. (1994): A taxonomy for real world modelling concepts. Information Systems **19**: 229-241.

Otero, M., Dolado, J. (2004): Evaluation of the comprehension of the dynamic modeling in UML. Information and Software Technology **46**(1): 35-53.

Parsons, J., Wand, Y. (1997): Using Objects for Systems Analysis. Communications of the ACM **40**(12):104-110

Peckham, J., Maryanski, F. (1988): Semantic Data Models. ACM Computing Surveys **20**(3):153-182

Pitrik, R. (1996): Analyzing the Notions of Attribute, Aggregate, Part and member in Data/Knowledge Modeling. Journal of Systems and Software **33**: 113-122

Prabhakaran,N., Falkenberg, E. (1988): Representation of Dynamic Features in a Conceptual Schema. Australian Computer Journal **20**(3): 98-104

Rahayu, J., Chang, E., Dillon, T., Taniar, D. (2000): A methodology for transforming inheritance relationships in an object-oriented conceptual model to relational tables. Information and Software Technology **42**: 571-592

Ram, S. (1995): Deriving functional dependencies from the entity-relationship model. Communications of the ACM **38**(9): 95-106

Rauh, O., Stickel, E. (1996): Modeling deductive information systems using ERMded. Decision Support Systems **18**: 135-143

Rochfeld, A., Negros, P. (1992): Relationship of relationships and other inter-relationship links in E-R model. Data & Knowledge Engineering **9**: 205-221

- Rolland, C., Richard, C. (1982): The Remora methodology for information systems design and management. In: Verrijn-Stuart, A., Olle T., Sol H., (eds.): Proceedings of IFIP TC-8 Conference on Comparative Review of Information Systems Methodologie (CRIS-1), North- Holland Amsterdam.
- Rolland, C., Souveyet, C., Moreno, M. (1995): An approach for defining ways-of-working. *Information Systems* **20**(4): 337-355
- Rumbaugh, J., Jacobson, I., Booch, G. (1999): The Unified Modeling Language reference manual, Addison-Wesley.
- Rumbaugh, J., Blaha, M., Premeriani, W., Eddy, F. & Lorensen, W. (1991): Object-oriented modeling and design, Prentice-Hall.
- Ruys, H. (1983): De ISAC-methodiek. Methodieken voor informatiesysteemontwikkeling, NGI (in dutch)
- Saiedian, H. (1997): An evaluation of extended entity-relationship models. *Information and Software Technology* **39**: 449-462.
- Scheer, A-W., Hars. A. (1992): Extending Data Modeling to cover the whole Enterprise. *Communications of the ACM* **35**(9): 166-172
- Scheer, A-W. (1994): ARIS toolset: A software product is born. *Information Systems* **19**(8): 607-624
- Scheer A-W. (1998): Business Process Engineering: reference models for industrial enterprises. Springer, Berlin.
- Shoval, P., Shiran, S. (1997): Entity-relationship and object-oriented data modeling- an experimental comparison of design quality. *Data & Knowledge Engineering* **21**: 297-315
- Silva, M., Carlson, G. (1995): MOODD, a method for object-oriented database design. *Data & Knowledge Engineering* **17**: 159-181
- Snoeck, M., Dedene, G. (1996): Generalization/specialization and role in object oriented conceptual modelling. *Data & Knowledge Engineering* **19**: 171-195
- Steimann, F. (2000): On the representation of roles in object-oriented and conceptual modeling. *Data & Knowledge Engineering* **35**: 83-106
- Storey, V. (1991): Relational database design based on the Entity-Relationship model. *Data & Knowledge Engineering* **7**: 47-83
- Teorey, T., Yang, D., Fry, J. (1986): A logical design methodology for relational databases using the extended E-R model. *ACM Computing Surveys* **18**(2): 197-222

- Thalheim, B. (2000): Entity-Relationship Modeling: Foundations of database technology. Springer verlag.
- Theodoulidis, C., Loucopoulos, P., Wangler, B. (1991): A conceptual modelling formalism for temporal database applications. *Information Systems* **16**(4): 401-416
- Tkach D. & Puttick, R., (1994), Object technology in application development, The Benjamin-Cummings publishing company.
- Tsichritzis, D., Lochovsky, F. (1982): Data Models. Prentice-Hall.
- Verheijen, G., van Bekkum J. (1982). NIAM: An Information Analysis Method. In: Verrijn-Stuart, A., Olle T., Sol H., (eds.): Proceedings of IFIP TC-8 Conference on Comparative Review of Information Systems Methodologie (CRIS-1), North-Holland Amsterdam, 537-590
- Vessey, I., Conger, S. (1994): Requirements Specification: Learning Object, Process, and Data methodologies. *Communications of the ACM* **37**(5): 102-113
- Ward, P. (1986): The transformation schema: An Extension of the Data Flow Diagram to Represent Control and Timing. *IEEE Transactions on Software Engineering* **12**(2) : 198-210
- Wijers, G. (1991): Modelling support in information systems development. Ph.D thesis, Technical University Delft.
- Yourdon, E., (1994), Object-oriented systems design: an integrated approach, Prentice-Hall.
- Yourdon, E., Constantine, L. (1979): Structured Design. Prentice-Hall.

CHAPTER 4

OPERATIONALIZED DESIGN SPECIFICATION

4.1 INTRODUCTION

The evaluation of existing design alternatives in chapter 3 has lead to the conclusion that no existing requirements determination method complies with the quality criteria for a requirements determination method that were given in chapter 2. This leads us to the *development problem* according to Van Engelen and Van der Zwaan (1994) since alternative designs will have to be developed in this research. In this chapter we will give an answer to the third research-(sub) question that we have given in chapter 1:

What are the necessary elements for the way of modeling, the way of working and the way of controlling for a requirements determination method so that this method complies with the quality criteria that we have given for the design specification?

In this chapter we will develop a specification for the to-be designed RDM in chapters 5 and 6 that takes into account the reasons for non-compliance with the criteria from chapter 2 for many of the existing approaches. This will constitute the *design criteria* (or *design specification* according to Van Engelen and Van der Zwaan (1994)).

We will draw conclusions from the derived criteria in chapter 2 and the literature survey on the state of the art in requirements determination approaches from chapter 3 and determine the explicit demands or requirements for the way of modeling, the way of working and the way of controlling for a 'to-be designed' RDM according to the *domain richness*, *completeness*, *efficiency*, and *formality* criteria. These operationalized 'design criteria' will be used to evaluate the way of modeling of the 'to be designed RDM chapter 5 and the way of working and the way of controlling of this to-be designed RDM in chapter 6.

4.2 RDM DEMANDS FOR THE WAY OF MODELING

4.2.1 RDM demands for completeness in the way of modeling

In chapter 2 we have given a general definition of the completeness criterion for the way of modeling of a RDM. In this chapter we will refine this definition to cater for modeling deficiencies that we have encountered while studying a number of existing approaches.

An information bearing construct must be applicable for all possible types of semantic relationships that can exist in an application domain. This means that such an information bearing construct in principle must facilitate the encoding of N-ary

relationships ($N \geq 1$). The encoding of binary relationships in such a situation will be a special case for which N equals 2. The literature survey from chapter 3 reveals that in most of the requirements specification languages that we have analyzed it is not possible to capture the abstracted natural language phrasing of a N -ary semantic relationships in a complete, precise and consistent way (preliminary RMD 1').

RMD 1': The information bearing modeling construct in the to be designed RDM must be able to express the complete, precise and consistent communication semantics of any N -ary semantic relationship.

Furthermore, the existence of non-domain based naming conventions, for example, a 'global' unique *object ID* in UML is in general not suitable as a naming convention to be used in application requirements specifications. Furthermore, there must exist one modeling construct for naming conventions that must be able to capture all domain semantics regarding the context in which the naming convention is valid. This leads us to the definition of RMD 2:

RMD 2: The modeling construct(s) for naming conventions must allow for one domain-based naming convention and must be able to capture the semantics regarding the context in which the naming convention is valid.

In the literature survey from chapter 3 we have found that in the (E)ER and OO approaches there is generally no compulsory role modeling construct defined. This can lead to severe problems when the contents of a requirement specification document must be communicated in a different way than in the diagrammatic or symbolic format. To denote a specific involvement of a given object type in a semantic relationship, especially when such an object or entity type plays more than one role in a semantic relationship, a compulsory role construct and an appropriate naming convention for roles must be contained in the requirements specification language.

RMD 3: The to be designed requirements method must contain a role construct and an explicit naming convention for roles.

From our literature survey in chapter 3 we can conclude that in the state-of-the-art in requirements specification languages, a difference exist between the modeling capabilities for business rules between on the one hand EER languages and on the other hand UML and ORM. The common denominator in terms of the types of business rules that can be encoded as constraints in each of those approaches can be considered those types that can be modeled as minimum or maximum cardinalities of relationships. This means that a to be designed requirements determination method must provide at least modeling facilities to express those business rules that are encoded as relationships cardinalities in traditional EER and OO approaches to be complete in terms of the 'state-of-the-art' in the way of modeling of existing approaches.

RMD 4: The static constraint types in the to-be designed requirements method must at least contain those types that enable us to encode those business rules that can be encoded by relationship cardinalities in EER and UML.

We, finally, note that the operationalization of the completeness criteria should lead to demands regarding the static derivation (rules), the dynamic constraints and dynamic rules. These demands, however, will be implied by ‘stronger’ RMD’s in section 4.2.4.

4.2.2 RDM demands for the domain richness in the way of modeling

With respect to the dimension *turbulence* of the domain richness criterion we can say that the to-be designed requirements RDM must accommodate the whole range of values that potentially can characterize an application domain. In case an application domain is stable, the requirements determination method will not need facilities to cope with changing requirements, however, if these facilities are available it does not mean that the method should not be applicable in stable environments. In case of turbulent application domains, the requirements determination method must have facilities that allow an analyst to easily adapt the requirements specification document to the evolving requirements. Ideally a 1-on-1 relation between a domain requirement and a requirements specification segment should exist. This means that the way of modeling of the to-be designed RDM must facilitate this 1-on-1 addition or deletion of a specific domain requirement and would imply that no unnecessary remodeling efforts need to be undertaken when the application business logic evolves.

RMD 5: A requirements specification that is the result of the application of the to-be designed requirements determination method must be able to adapt to an evolving application logic without unnecessary remodeling.

4.2.3 RDM demands for the efficiency in the way of modeling

In this section we will refine the efficiency criterion for the way of modeling from chapter 2 into design criteria for the number of modeling constructs for the data model and the robustness of constraint definitions that will be defined in the way of modeling of the RDM.

In the survey on the existing requirements determination approaches in chapter 3 we have shown how the availability of more than 1 information bearing construct in a requirements specification language can lead to rework in the requirements determination process, either in the ‘short-run’ when the initial requirements specification needs to be adapted or in the ‘long-run’ when an evolving requirement leads to an adaptation of the requirements specification that has a bigger impact than is implied by the evolving requirement. The existence of multiple information bearing constructs can lead to unstable models. This leads us to the requirement (RMD) that a to-be designed requirements method must contain 1 information bearing modeling construct. We will, accordingly, redefine RDM 1’ into RDM 1 to cater for this:

RMD 1: A to be designed RDM must contain 1 information bearing modeling construct. This construct must be able to express the complete, precise and consistent communication semantics of any N^{21} -ary semantic relationship.

²¹ $N \geq 1$

Another requirement for the to-be designed requirements determination method is the status of the default existence of application objects or entities. We will require that the domain semantics that declare the existence of entities or objects on their own (e.g. without participating in a semantic relationship) should be encoded explicitly and therefore in the default situation the existence of objects or entities on their own must not be implied, thereby preventing the modeling rework. Hence, this will have a positive effect on the efficiency of the to be designed requirements determination method.

RMD 6: The definition of an application object or entity in the to be designed requirements method must not imply that it can exist on its own by default.

In the literature survey we encountered inconsistencies in some requirements specification languages with respect to (some types of) state constraints. Especially, the *minimum cardinalities* in some (E)ER approaches and *minimum multiplicities* in the UML are not defined in a consistent way. Their definition changes when the arity of the semantic relationship changes from 2 to N ($N > 2$) or the definition is only valid for $N=2$. In the specification language of the to be designed requirements method, therefore, we demand that a constraint of a constraint type X that is defined on a semantic relationship of arity N must have the same generic definition as a constraint of a constraint type X that is defined on a semantic relationship having arity M (where $N \neq M$). This leads to the following RMD:

RMD 7: The definition of the static constraint types in the to-be designed requirements method must be the same for all arities of the semantic relationships in the data model and must contain an explicit reference to the elements in the data model.

4.2.4 RDM demands for the formality in the way of modeling

In this section we will refine the formality criterion from section 2.4 for the way of modeling into the preciseness of the specifications that can be created using the to-be designed RDM. Subsequently we will refine the formality criterion for the way of modeling into a number of consistency requirements between the definition of the different modeling constructs for the different elements in a requirements specification. Furthermore, the to be designed requirements determination method needs to allow the different dynamic constraint types to be encoded. This means that modeling constructs must be defined that allow us to specify operations on the application's data base in terms of evaluating the current data base, but also in terms of evaluating a projected 'to-be' state.

RMD 8: The definition of the dynamic constraint types in the to-be designed requirements method must enable us to explicitly refer to the (actual and projected states of the) application's data base.

From the literature survey in we concluded that not all requirements specification languages have facilities to express derivation rules in a precise way. Not only should

the to be designed RDM specify what semantic relationships are derivable, but the specification language of the to be designed RDM should also contain modeling constructs that allow an analyst to precisely denote, how a static derivation takes place in terms of the knowledge that is contained in (other parts) of the data model. This requirement (RMD 9) therefore assumes that an effective naming convention is in place to reference the elements in the data model (see RMD 3 for the naming convention for roles)

RMD 9: The definition of static derivation (rule) in the to be designed requirements method must contain an explicit reference to the elements in the data model that serve as an input for the static derivation (rule) and it must contain a precise specification on how these input elements lead to the result of the static derivation (rule).

The literature survey in chapter 3 revealed that most requirements determination approaches that we have studied do not provide facilities to model dynamic constraint types. In the Unified Modeling Language (UML) a (number of) diagram type(s) exist that intentionally express these types of constraints. In the UML, the coherence between the different diagram types is unclear and the consistency between the model elements that are featured within one or more diagram types must be seriously questioned. For the *to-be designed RDM*, therefore we need to be able to define the dynamic constraints in a consistent way in which the elements in the data structure and the derivation rules that are involved in a dynamic constraint must be specified precisely. In an event-condition-action triplet (ECA), an event is something that ‘happens’ within the application subject area. Such an event can be caused by a change in state of the application information base (internal event) or by something that happens in the application area outside the information base (external event). This means that the following demands must be met in terms of the definition of *internal* and *external* events (RMD 10).

RMD 10: An internal event in the to-be designed RDM must be defined as the insertion or deletion of a specific piece of domain knowledge into or from the application’s data base. An external event in the to-be designed RDM must be defined as something that happens in the application domain and that can lead to the insertion or deletion of a specific piece of domain knowledge into or from the application’s data base or the execution of a static derivation rule (eventually) under some condition on the content of the application’s data base.

Furthermore, if we want to enforce the condition under which an event will lead to the execution of a derivation rule (action alternative 1) and or the insertion/deletion of information into or from the application’s data base (action alternative 2) we must be able to express such a condition within the event-condition-action rule as a proposition on the application data base that any point in time must evaluate to *true* or *false*.

RMD 11: A condition in the to-be designed requirements method must be defined as a proposition on the application’s information base that must yield the value true or false when evaluated at any point in time.

4.3 RDM DEMANDS FOR THE WAY OF WORKING

4.3.1 RDM demands for completeness in the way of working

From the literature survey of the EER, UML and ORM approaches we concluded that a mere definition of a fact or constraint type modeling construct, does not guarantee that all fact types, naming conventions or all instances of such a constraint type will be ‘found’ or ‘expressed’ by domain experts all by themselves at all times. The to-be designed requirements determination method, therefore, must give guidance to an analyst in deriving (the verbs of) semantic relationships, naming conventions, specializations/generalizations and all instances of a specific constraint type that exist in the application area, for example, in a dialogue with the domain user(s). On the other hand we will focus the general completeness criterion for the way of working to an ambition level that is equal to the maximum level of completeness that we can achieve by applying the existing modeling requirements specification alternatives. This means that with respect to the way of working for the constraint types that we must incorporate into a to be designed RDM we will limit ourselves to those constraint types that can be modeled in (E)ER and UML. This leads to requirement RMD 12.

RMD 12: The definition of the modeling constructs for the data model in the to-be designed requirements method must be accompanied by some kind of guidance on how all instances of these modelling constructs can be found in an application subject area. The definition of the state constraint types in the to-be designed requirements method must be accompanied by some kind of guidance on how such instances of a constraint type can be found in an application subject area.

4.3.2 RDM demands for the domain richness in the way of working

With respect to the dimension *perception* we have concluded that a to be designed requirements determination method must provide facilities to incorporate the different views or perceptions of different user (groups) into one requirements specification document. This means that the to-be designed requirements determination method needs facilities for the integration of multiple views (of the underlying reality).

RMD 13: A view integration sub-procedure must be defined in the to-be designed requirements method in which it is specified how an analyst must carry out the integration of views on the application domain by user (groups) that have a different perception on the ‘underlying’ reality.

The requirements determination method that will be designed also needs to facilitate the elicitation of tacit knowledge that is held by the domain user (dimension *tacitness*). It is the implicit knowledge that can be made explicit but that can not be uttered ‘out of the blue’ by the application domain users. It is this knowledge that can be potentially encoded as a (part of a) data model and the accompanying constraints by eliciting it from domain users by using one or more ‘knowledge’ elicitation procedures.

RMD 14: The to-be designed requirements determination method must provide facilities for transforming implicit tacit knowledge into explicit knowledge.

With respect to the dimension *anchoring* we note that the to be designed requirements determination method must be able to accommodate the whole spectrum of starting situations on the anchoring scale ranging from an abstract starting point on one side to a tangible anchor on the other side.

RMD 15: The to be designed requirements determination method must accommodate every possible starting point in the requirements determination process ranging from abstract to tangible; ranging from natural language description to documents that can only be understood by domain users.

4.3.3 RDM demands for the efficiency in the way of working

We have concluded in chapter 3 that the requirements modeling facilities that exist in the approaches that we have found in the literature merely tell an analyst *what* to do next instead of specifying *how* these ‘steps’ must be performed in the requirements determination process itself. The necessity of (a) precise modeling procedure(s) for instantiating static constraints was already shown. In addition, preceding modeling steps are needed that specify how to derive the application’s data structure. In addition to the existing ‘modeling’ procedures in the (E)ER, ORM and UML approaches that we discovered in chapter 3, we need (a) modeling procedure(s) that not only specifies what must be done, but also *how* it must be done in the most efficient way, thereby minimizing the required number of analysis steps. The way of working for the to be designed RDM must also be adaptable in the sense that dependent upon context in which a requirements determination project is carried out, the most efficient way can be implemented for that project.

RMD 16’: A modeling procedure must be defined in the to-be designed requirements method in which it is specified how an analyst must carry out the modeling steps in the most efficient way.

4.3.4 RDM demands for formality in the way of working

As we already discovered in chapter 3, most of the ‘modeling procedures’ that exist in the ‘state-of-the art’ in requirements determination methods, only tell an analyst what to do. However, in order to guarantee that different analyst will arrive at the same (or equivalent) requirements specification in any given project we need not only a procedure that tells an analyst how to create a requirement specification, but this procedure must be formally defined, for example as an algorithm. This means that we will replace the implied RMD 16’ with the following RDM:

RMD 16: Formal modeling procedure(s) must be defined in the to-be designed requirements method in which it is precisely specified how an analyst must carry out a modeling step in the most efficient way.

4.4 RDM DEMANDS FOR THE WAY OF CONTROLLING

4.4.1 RDM demands for efficiency in the way of controlling

The way of controlling criterion is concerned with issues like project- and quality management and therefore must be built-in in the requirements determination method. In each relevant requirements determination step the user must be confronted with the in-between results (or milestone document) in the terminology that he/she understands so he/she can question his/her assumptions and validate the interpretation of the information that he/she supplied to the analyst

RMD 17': The way of working in the to-be designed RDM must have explicit quality assuring sub-procedures for the activities of the work breakdown structure and checks that enables an analyst to validate the information that is supplied by the user and that confronts a domain user with his/her assumptions and enables a user to validate the information that is supplied to the analyst.

4.4.2 RDM demands for formality in the way of controlling

This formality in the way of controlling concerned with the extent in which the activities that are defined within the RDM's way of working can be formally planned. This means that we need a work-breakdown structure (Mantel et al., 2001) in the definition of the way of working in the to-be designed RDM that will allow us to formally plan the activities in a requirements determination project.

RMD 18: The way of working in the to-be designed requirements determination method must have a work breakdown structure that allows to formally plan the activities in a requirements determination project.

The formality in the way of controlling, furthermore, is concerned with the quality assurance steps in the RDM's way of working, it means that these quality assurance sub-procedures must be an integral part of the processes that create the intermediate RDM 'products' and hence we can replace RMD 17' by RMD 17 :

RMD 17: The way of working in the to-be designed RDM must have explicit formal quality assuring sub-procedures for the activities of the work breakdown structure and formal checks that enables an analyst to validate the information that is supplied by the user and that confronts a domain user with his/her assumptions and enables a user to validate the information that is supplied to the analyst.

Finally, the RDM must provide facilities in the way of modeling and in the way of working that enable traceability.

RMD 19: The way of modeling and the way of working in the to-be designed RDM must have provisions that enable traceability.

4.5 CONCLUSIONS

We have derived in this chapter 19 demands that must be fulfilled by a requirements determination method that needs to be designed. These demands can be considered to be operationalizations of the four (groups of) criteria that were given in chapter 2 in which remedies for the major flaws regarding these criteria, from the existing approaches that we have studied in chapter 3, are incorporated. Analyzing the research literature on this topic shows many times that scholars also postulate criteria like: *must be able to timestamp domain data* or *must contain a modeling construct for modelling time*. We note that the latter examples of criteria in our view belong to the domain characteristics, and, therefore will end up in a specific requirements specification as part of a data model and/or one or more constraint instances.

We will use these requirements method demands (RMD's) to evaluate the to be designed RDM in chapters 5 and 6 of this thesis.

4.6 REFERENCES

Mantel, S., Meredith, J., Shafer, S., Sutton, M. (2001): Project management in practice. Wiley and Sons

Developing an alternative design for a requirements determination method

In the second part of this thesis (chapters 5 and 6) we will develop an alternative design for a requirements determination method. In chapter 5 we will focus on the *way of modeling* by describing *what* should be contained in a requirements determination method. In chapter 6 we will focus on the *way of working* and *way of controlling* by precisely describing how the model elements in the requirements specification language can be instantiated in a specific application subject area and how the overall requirements determination process must be controlled when using this method.

CHAPTER 5²²

THE WAY OF MODELING IN NATURAL LANGUAGE MODELING

5.1 INTRODUCTION

In this chapter we will define the way of modeling of a requirements determination method that we will call Natural Language Modeling (NLM). The information modeling constructs in NLM are based upon the axiom that all verbalizable information (computer screens, reports, note-books, traffic signs and so forth) can be translated into *declarative natural language sentences* (Nijssen, 1989:158). It means that it is not a ‘real’ or ‘constructed’ world that is subject to modeling, but that it is the communication about such a ‘real’ or ‘constructed’ world (van der Lek et al., 1992). In this chapter we will show that such a communication oriented way of requirements modeling will lead to a smaller number of necessary modeling constructs than in requirements determination methods that we have found in the literature.

The applicability of NLM is not limited to requirements determination but it can also be applied for the semantic analysis of ‘static’ knowledge domains (Nijssen and Bollen, 1995). Another area in which NLM can be applied is meta data semantics for the world wide web (Grönbaek et al. 2000; Broekstra et al. 2002; Resource Description Framework 2004) and ontologies for the world wide web (Davies, Duke and Stonkus, 2002; Davies, Weeks and Krohn, 2004). NLM, furthermore provides facilities for implementing natural language processing for querying large data bases (Conlon et al., 2004)

The modeling constructs of Natural Language Modeling (NLM) are an evolution of the modeling primitives that are rooted in the *semantic binary model* (Abrial, 1974), the *object-role model* (Falkenberg, 1976a, 1976b), *NIAM* (Verheijen and van Bekkum, 1982) and *Fact Oriented Information Modeling* (Halpin, 1995; Halpin and Orlowska, 1992). Amongst other semantic modeling approaches that have gained popularity, are Sowa’s conceptual graphs (Sowa, 1984).

5.1.1 Organization of chapter 5

In this section we will provide some guidelines for the reader of this thesis on how to read this chapter. In sections 5.2 through 5.6 we will introduce two primitives and the basic modeling constructs (roles, intentions, extensions, fact types, naming conventions and compound referencing schemes) for our to be designed requirements specification language. The application of these primitives and the modeling constructs based hereupon, in a requirements determination process will lead to an instance of a basic

22 An earlier version of this chapter was published as ‘Natural Language Modeling and Application Ontologies.’, Bollen, P., Proceedings ISoneworld 2003, Las Vegas, U.S.A. (2003).

information model that is defined in section 5.7. The discussion of these basic modeling constructs is illustrated by a ‘running’ example of a university enrolment UoD. The domain semantics for the UoD of this example are paraphrased in two locations in this chapter. The first part of our running example is given in section 5.2. In this first part of the running example the university enrolment of a single (Vandover) University is considered. The second part of our running example in section 5.6.2 paraphrases the domain semantics of an integrated university enrolment UoD in which two universities have merged and have streamlined their enrolment activities. This second part of our example description also includes the dynamic aspects of this UoD. In section 5.8 we will give the NLM constraint modeling constructs. We advise the readers that are not familiar with constraints to read appendix A, before reading section 5.8. Readers who are familiar with the constraint modeling construct can skip appendix A, when reading this thesis. We will illustrate in sections 5.6 through 5.9 how we can formalize the paraphrased domain semantics from the UoD of the (integrated) University enrolment example into an instance of NLM requirements specification in section 5.9. In addition we will explicitly show in section 5.9 how the textual description of our example maps onto the instances of the modeling constructs from this chapter and appendix A. In section 5.10 conclusions will be drawn regarding the extent in which the way of modeling in NLM fulfills the operationalized design criteria for a requirements determination method that were derived in chapter 4 of this thesis.

5.2 NAMES

We will first introduce the running example for this chapter. In section 5.6.2. we will give a further description of this running example in which Vandover University has merged with Ohoa University and in which the dynamic aspects of this (integrated) UoD will be given.

5.2.1 Example 5.1: University Enrolment part 1

The University Enrolment example will be used to illustrate the modeling concepts throughout this chapter of the thesis. The first part of the University Enrolment example will consist of the old situation that deals with the Vandover University. In the second part of this chapter we will introduce the new situation for the University Enrolment example in which a merger has taken place between the *Vandover* and *Ohoa* universities. The University of Vandover offers a number of majors in education. Students can choose between majors in *Science*, *History* and *Economics*.


| | | |
|---|------------------|--------------|
|  Vandover University Enrollment | | |
| Student id | last name | major |
| 1234 | Thorpe | Science |
| 5678 | Jones | Economics |
| 9123 | Thorpe | History |

Fig. 5.1 Example Vandover University Enrolment (example 5.1).

In figure 5.1 an example is given of a university enrolment document (example 5.1). In this example the Vandover University wants to record information about the major for each of its students. It is assumed that the student ID can be used to identify a specific student among the union of students that are (and have been) enrolled in the Vandover University and that a major name can be used as identifier for a specific major among the set of majors that are offered by the Vandover University.

5.2.2 The name primitive

A name in human communication is used to refer to a concept or a thing in a real or constructed world (Senko, 1976).

Primitive 1. A *name* is a sequence of words in a given language that is agreed upon to refer to at least one concept or thing in a real or constructed world.

Examples of names: *Jake Jones*, *567893AB*, *General electric*,
We will now define the archetype.

Definition 5.1. The *archetype* is the union of all names.

Let A be the archetype.

Let n_i be a name.

$$A = \cup_i n_i$$

The choice of names used in communication is constrained by the reference requirement for effective communication. For example, the university registration office will use a *student ID*, for referring to an individual student. The use of names from the name class *last name* in the university registration subject area for referring to individual students, however, will not lead to effective communication because in some cases two or more students may be referenced by one name instance from this name class. This is one of the reasons why not all names (or name classes) can be used for referencing entities, things or concepts in a specific part of a real or constructed world.

On the other hand 'knowledge workers' that are involved in activities in an application subject area have knowledge on the reference characteristic of the potential name classes for the different groups of 'things' and concepts in their real or

constructed world. This means that they should be able to tell an analyst whether a name from a specific name class can be used to identify a thing or concept among the union of things or concepts (in a specific part of a real or constructed world).

5.3 THE NATURAL LANGUAGE AXIOM

In every (business) organization examples of communication can be found. These examples can be materialized as a computer screen, a world wide web page, a computer report or even a formatted telephone conversation. Although the outward appearance of these examples might be of a different nature every time, their content can be expressed using natural language. We will refer to this class of examples of communication²³ as *verbalizable information* (Nijssen, 1989).

Primitive 2 (Natural language axiom). All verbalizable information can be expressed as declarative natural language sentences.

The application of the natural language axiom on the example of communication from figure 5.1 will result in declarative sentence instances 1.1 through 1.6.

The student 1234 majors in Science.....(sentence 1.1)
The student 5678 majors in Economics.....(sentence 1.2)
The student 9123 majors in History.....(sentence 1.3)
The student 1234 has last name Thorpe.....(sentence 1.4)
The student 5678 has last name Jones.....(sentence 1.5)
The student 9123 has last name Thorpe.....(sentence 1.6)

We can conclude from the literature that for the initial stage in the requirements determination process, the use of natural language (Bubenko and Wangler, 1992) is recommended (Ambrosio et al., 1997). Henderson-Sellers and Edwards (1990:194) state that a requirements definition must be expressed in the language of the user, so the analyst and user can agree upon the content.

5.4 ROLES

If we analyze sentences 1.1 through 1.6 that have resulted from verbalizing the university enrolment example in figure 5.1 we can divide them into two groups according to the type of sentence predicate (*..majors...*, respectively *..has last name..*). If we focus on the first group we can derive two sentence group templates in which we have denoted the predicate as text and the variable parts as text between '<>' brackets:

²³ Sometimes the concept of 'form' is used to refer to any structured document (e.g. See Choobineh and Venkatraman (1992: 270).

Student <enrolled student> majors in major <chosen major>.
Student <enrolled student> has chosen the major <chosen major>.

We will refer to these variable parts as *roles*.

Definition 5.2. A role is a variable part in one or more sentence group templates.

Let nm be a name slot in a sentence group SG
Let s_1, s_2 be sentence instances of sentence group SG
 $R = \{ nm | \exists s_1, s_2 [nm_{s_1} \neq nm_{s_2}] \}$
A role $r \in R$

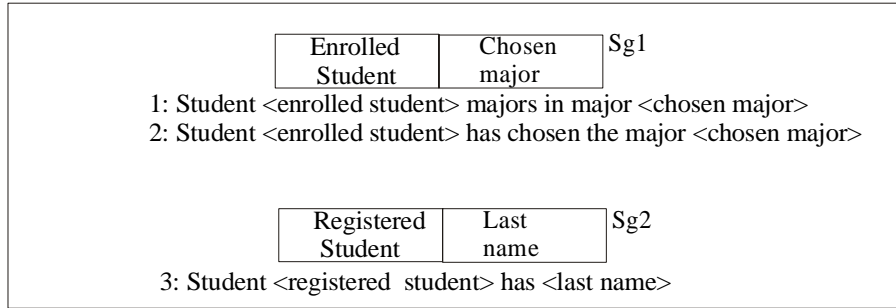


Fig. 5.2 Roles and sentence group template for university enrolment example.

Figure 5.2 shows a graphical representation of the two sentence groups in the University Enrolment example. Each role is graphically represented by a ‘box’, e.g. enrolled student. Each sentence group is represented by a combination of role boxes. Sentence group *SG1* is represented by the combination of role boxes *enrolled student* and *chosen major*. Sentence group *SG2* is represented by the combination of role boxes *registered student* and *last name*. For each sentence group one or more sentence group templates are positioned underneath the combination of role boxes that belong to the sentence group. In the diagram of figure 5.2 sentence group templates 1 and 2 belong to sentence group *SG1*. Sentence group template 3 belongs to sentence group *SG2*. The remaining parts of a sentence group template we will call *verb*-parts.

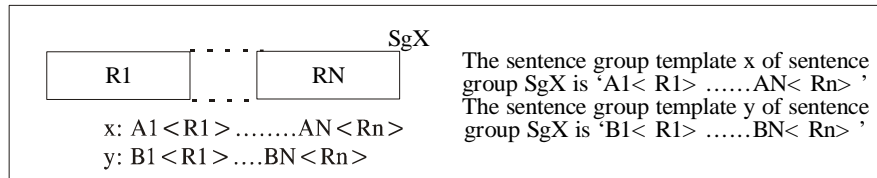


Fig. 5.3 Example legend for sentence groups²⁴.

²⁴ We note that a similar legend applies for fact types. In section 6.3 we will define the difference between sentence groups and fact types.

In figure 5.3 we have given the legend for the verbalization of the graphical representations of the modeling results as for example given in figure 5.2.

5.5 INTENTION AND EXTENSION

If we inspect figure 5.2 we see that a sentence group template can reveal additional information about the type of things that can be ‘inserted’ into a *role* variable. For example, the word ‘student’ specifies what type of thing (or concept) is allowed to play the role ‘enrolled student’ but also what type of thing (or concept) is allowed to play the role ‘registered student’. We will call the ‘student’ part in the sentence groups in figure 5.2 the intention of the roles ‘enrolled student’ and ‘registered student’.

Definition 5.3. An *intention* is the meaning or the definition of a concept in a real or abstract world.

Let X be a concept.
Let DX be the definition of the concept X
Let $Int(X)$ be an intention
 $Int(X) = DX$

We can for example give the definition of the concept *Student*:

A student is a person that studies at a University.

The set of names of things or concepts to which such a definition of an intention applies within a *specific* application subject area at a specific point in time is called the *extension* of the intention.

Definition 5.4. The *extension of an intention* is the set of names of the things or concepts to which the definition of the intention applies²⁵.

Let X be a concept. Let DX be the definition of the concept X
Let A be the set of names of things or concepts to which definition DX applies
Let $Ext(X)$ be an extension
 $Ext(X) = A$

We can now give an example extension for the intention *Student*: {1234, 5678, 9123}. In the remainder of this paper we will use the intention concept to denote the *type* of thing or concept to which a *specific* thing or concept belongs. For every application area we should document the intentions, their definitions and synonyms in a list. We

²⁵ We will call a specific concept or a ‘thing’ that is referenced by one element of the intention’s extension, an *instance* of the intention.

will call this list the *application concept repository (ACR)*. In the following we have given the ACR for part 1 of our Vandover University Enrollment example.

| Concept | Synonym | Definition |
|-------------------|-----------------------|--|
| <i>Student</i> | | <i>a person that studies at Vandover University</i> |
| <i>Student ID</i> | | <i>a name class</i> |
| <i>Major</i> | <i>Specialization</i> | <i>course program offered to students by Vandover University</i> |
| <i>Major name</i> | | <i>a name class</i> |
| <i>Last name</i> | | <i>a name class</i> |

Such a list of concepts and their definitions should contain a definition for each intention in the UoD. The definition of an intention should specify how the knowledge forming the intention (definiendum) is to be constructed from the knowledge given in the definition itself and in the defining concepts (definiens). A defining concept (definiens) should either be an intention (from the list) or it should be defined in common business ontology (in our example *course program* and *university*) or it must be a trivial and generally known concept (in our example *person*).

5.6 FACT TYPES

In this section we will take the meaning of a sentence as opposed to its format as a starting point. The construct for modeling the meaning of a sentence is a fact instance. A fact instance is expressed as a declarative natural language sentence instance of a one of its corresponding sentence group template(s). It is possible that the extensions of two different sentence group templates refer to the same fact. For example we can say that there exists a fact that a student is enrolled in a major (at the University of Vandover). Two sentence instances (from different sentence group templates) for communicating this fact instance can be:

Student 1234 has chosen the major Science.
Student 1234 majors in major Science.

We need to make a distinction into the concept of fact and the concepts that we use to represent a fact. A specific fact instance can be represented as one or more sentence instances from one or more sentence group templates. The sets of roles that are referred to in these sentence group templates should be identical. We can now conclude that a fact type is a set of roles that can be represented by one or more sentence group template(s) in which these roles are contained.

Definition 5.5. A *fact type* is a set of roles belonging to a sentence group.

Let r_i be a role that belongs to a given group of sentence templates SG
Let FT_j be a fact type.
 $FT_j = \{ r_i \mid r_i \text{ is a variable part of } SG_j \in SG \}$

Example 5.2: STUDENTMAJOR:={enrolled student, chosen major}
STUDENTNAME:={registered student, last name}

Because every fact type has at least one accompanying sentence group (template) that contains the precise domain semantics as verbalized by the domain users it is not required to use 'semantic' role names at all times. The only requirement for a role name is that it is unique within a fact type (provided that we use unique names for a fact type).

A second distinction between sentence groups is fact type lies in the notion of atomization. In chapter 6 we will discuss this issue in greater detail. For now it is sufficient to say that a fact type must be elementary, atomic or irreducible in the context of a given UoD. A sentence group on the other hand might be non-atomic or (still) reducible.

5.6.1 Naming convention fact types

In this section we will further formalize the outcome of the process of the selection of a name class for referring to things in a real or abstract world. The outcome of such a naming process will result in the utterance of sentences, examples of which are given below:

1234 is a name from the student ID name class that can be used to identify a student within the set of students at Vandover University.....(sentence 2.1)

5678 is a name from the student ID name class that can be used to identify a student within the set of students at Vandover University.....(sentence 2.2)

Science is a name from the major name name class that can be used to identify a major within the set of majors at Vandover University.....(sentence 2.3)

Economics is a name from the major name name class that can be used to identify a major within the set of majors at vandover University(sentence 2.4)

Sentences 2.1 through 2.4 express that a certain *name* belongs to a certain *name class* and that instances of the name classes *student ID*, respectively *major name*, can be used to identify an instance of a *student*, respectively an instance of a *major*. We can give for example the definition of the concept *Student ID*:

Student ID is a name class.

The 'intention' of the names in the extension of a naming convention fact type is a name class and NOT a type of thing, entity or concept in the real world. We will, therefore, refer to facts 2.1, 2.2, 2.3 and 2.4 as *naming convention facts* and to the corresponding abstracted fact type as *naming convention fact type*.

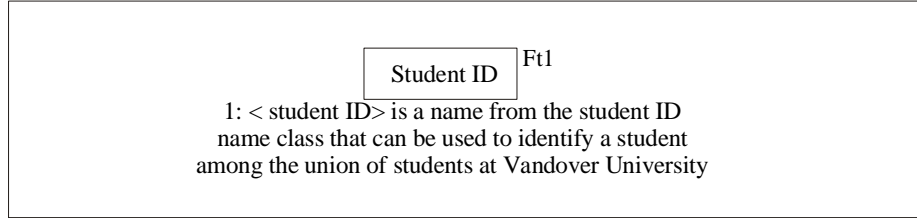


Fig. 5.4 Naming convention fact type for student at Vandover University

Definition 5.6. Facts of a *naming convention fact type* declare that a name from a name class can be used to identify a thing or a concept in a selected portion of a real or abstract world.

Let nc be a name class.

Let A be the archetype: $nc \subset A$

Let $DOM(T)$ be the concepts in the selected portion of the real world.

It now holds: $\forall c, d \in T; t, v \in nc$ [t is a name for $c \wedge v$ is a name for d]:

$c = d \Leftrightarrow t = v$

In business domains, however, not all things or concepts can be identified by using a single name from a given name class or a simple domain. In general, a name can have an internal structure of itself, therefore, we need to generalize the concept of a (simple) naming convention fact type to a referencing scheme in which names are composed of values from multiple domains. We will call those referencing schemes, *compound referencing schemes* and they will be discussed in section 5.6.3. First we will give an extension of the university enrolment example in section 5.6.2.

5.6.2 Example 5.1: university enrolment part2

We assume that Vandover university has merged with Ohao university. In order to streamline the enrolment operations of the two universities in the new situation it is decided to centralize them.

A student within the merged Ohoadover university federation can no longer be identified by the existing *student ID*, because a given *student ID* can refer to a student in the former Ohoa university and to a different student in the (former) Vandover university. However, to capitalize on the existing naming conventions, management has decided to add the qualification O (for Ohao) or V (for Vandover) to the existing *student ID*. We will call this qualification: *university code*.

Furthermore, there is a possibility that a student is registered as freshmen at the Ohao University and at the Vandover University at the time of the merger. In this case it is decided that the student is considered to study only at the Ohao University for naming purposes.

In the new situation the Ohoa and Vandover universities still offer their own freshmen year and the majors that existed before the merger. This means that after the merger students that apply for enrolment will be assigned *student ID*'s within the university that enrolls them for their freshmen year. For our application it suffices to

know that student can be identified by the combination of a University code and a locally unique student ID.

Furthermore, we assume that the assignment of a student's university ID, which takes place outside our example UoD, will always lead to the recording of a student's last name.

In this example we assume that the University system is closed in the sense that a major can only be selected by students that are currently doing (or have been doing) a freshmen year at Ohoa University and/or Vandover University. A student can apply for at most one major at a time. The majors that are offered after the merger are a simple union of the existing majors at the 'old' universities: *science*, *history* and *economics* at Vandover University and *medicine* and *law* at Ohao university.

In the integrated Ohoa and Vandover enrolment system it is decided to record all courses and the credits that have been obtained by them during their student's freshmen year at Ohoa or Vandover.


|  Ohoadover University Enrollment | | | |
|---|------------|-----------|-------------|
| Total number of students enrolled | | | 156 |
| University | Student ID | Last Name | Major |
| V | 1234 | Thorpe | Science |
| | | | Accounting |
| | | | Finance |
| | | | Marketing |
| | | | Mathematics |
| | | | <u>8</u> |
| | | | 26 |
| O | 5678 | Smith | Law |
| | | | Macro econ. |
| | | | Micro econ. |
| | | | Finance |
| | | | <u>8</u> |
| | | | 24 |
| V | 5678 | Jones | |
| | | | Accounting |
| | | | <u>5</u> |
| | | | 5 |

Fig. 5.5 Example integrated Ohoadover enrolment system

The definite enrolment in the major of their choice depends upon the number of credits that a student has earned in his/her freshman year. If the total number of (approved) credits for these freshmen courses is 24 or more and the specific required freshmen courses for the major of their choice are contained in their credited freshmen courses than a definite enrolment, for the student will always be recorded.

In the following the specific required freshmen courses for each major are given:

| Major | Required course | Required minimum # of credits |
|------------------|-----------------------------|--------------------------------------|
| <i>Science</i> | <i>Mathematics</i> | 8 |
| <i>History</i> | <i>Language and culture</i> | 5 |
| <i>Economics</i> | <i>Macro economics</i> | 8 |
| <i>Medicine</i> | <i>Biology</i> | 5 |
| <i>Law</i> | <i>Finance</i> | 5 |

A ‘real-life’ user example of the integrated Ohoadover enrolment system is given in figure 5.5.

The example of figure 5.5 contains *Majors* that can be identified by a major name among the union of majors at Vandover and Ohoa University. We also have *Courses* that can be identified by a *course name* among the union of courses at Vandover and Ohoa University. *Course credits* or *the total number of course credits* for a student or *the total number of enrolled students* is expressed by a *natural number*. At any point in time a student can have at most one total number of credits. The total number of credits for a student is the arithmetic total of all individual credits for credited courses. If a student does not have any credits assigned for at least one course the total number of course credits will not be shown.

We will now give a description of additional semantics from the Ohoadover enrolment UoD. First of all, the responsibilities for exams, grading and the assignment of study credits is not considered to be part of this UoD. We will therefore consider this to take place outside the UoD.

There will be some kind of message coming from another part of the university system that acknowledges at a certain point in time that *student X has been credited Y credits for course Z*. As soon as such a message is received by the enrolment clerk the information is entered into the enrolment system.

After the new course credits have been entered into the enrolment system the total number of credits for that student will be recalculated. Furthermore, students are allowed to switch majors before graduation. In that case the requirements regarding the content of their freshmen course and credits needed for this new major will be checked again. In addition the management of Ohoadover has decided that not all enrolment switches are allowed. At this time the following restriction is applied: A student can not major in *Economics* after he/she has majored in *Science*.

After a student has successfully finished his/her current major (this is decided outside the scope of our example UoD) he/she will be removed from the University Enrollment system.

Every time a student is enrolled or graduated the enrolment system will recalculate the total number enrolled students. The total number of enrolled student at any point in time is calculated by inspecting which of the registered students are currently enrolled for a major.

5.6.3 Compound referencing schemes

In the first part of the (Vandover) university enrolment example the intention ‘student’ has a ‘simple’ referencing scheme, namely the single role ‘enrolled student’ or the single role ‘registered student’. In many cases, however, a simple referencing scheme will not be sufficient for referencing instances of a given intention within a specific portion of a ‘real’ or ‘constructed’ world. In those cases we need *compound* referencing schemes.

In NLM we will apply compound referencing scheme in the same way as the simple referencing schemes. To illustrate this we will first adapt our example UoD and give additional domain knowledge that also covers the dynamic aspects of the UoD.

| | | | |
|-----------------|------------|--------------|------|
| University code | Student ID | Chosen major | Ft10 |
|-----------------|------------|--------------|------|

1:Student [identified by the combination of <university code> and <student ID>]

Fig. 5.6 Fact types and sentence group templates with compound referencing scheme for student from the university enrolment example part 2

The sentence group templates and the corresponding fact types in which such a compound referencing scheme is implemented are given in figure 5.6²⁶. We have

²⁶ See for an earlier discussion on aggregation: Smith and Smith (1977).

introduced the [] ('square brackets') symbol for capturing the definition of the compound referencing scheme (see figure 5.6). The case of a simple referencing scheme in NLM is a special case of the compound reference scheme in which the brackets and description within (except for the role name used in the reference) are left out. In addition we need to adapt the naming convention fact types for the constituting intentions of the compound reference scheme. For example the naming convention fact type for student should be adapted to reflect the application subject area in which it can be used to identify a specific student. In this case a student can be identified by his/her student ID within a specific University (Ohao or Vandover).

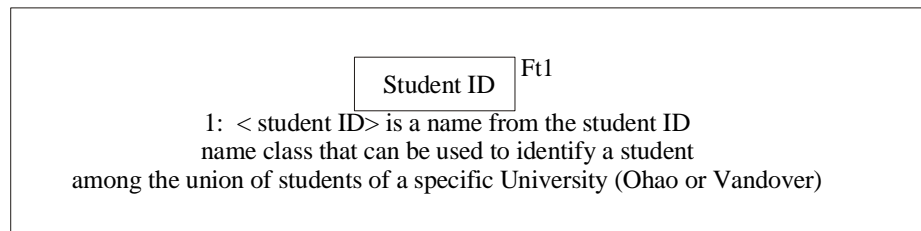


Fig. 5.7 Naming convention fact type for student in the integrated UoD.

In the University Enrollment example we have shown how the evolvement of a Universe of Discourse can lead to the existence of compound referencing schemes. In other situations a reference scheme that is a concatenation of multiple value domains can be considered a simple referencing scheme. An example of the latter is the postal code in the Netherlands that consist of two groups of values: 4 numerical characters (0-9) and 2 non-numerical (a-z) characters. In most UoD's the postal code is considered to be a simple referencing scheme for a postal area. However, in the UoD of the national Dutch postal code administration the first 2 characters refer to a municipality in the Netherlands, the last two digits refer to a neighbourhood within that municipality. Finally, the non-numerical characters refer to a specific 'side' of the street within that neighbourhood (within that municipality). In this example we can consider the intention *postal area* as an aggregation (Smith and Smith, 1977) in which the 'aggregated intention' postal area is composed of the 'basic intentions' *city*, *neighbourhood in that city* and *part of the street in this neighbourhood of that city*. In the fact type template of this example we therefore can incorporate the 'basic intentions' as follows:

Postal Area [identified by the combination of **City** <city code>, the **neighbourhood** <neighbourhood code> in the city and the **part of the street** <part of street code> in that neighbourhood] contains the number of addresses <natural number>.

Furthermore, the naming convention fact types for the basic intentions in this compound referencing scheme are necessary, in order to precisely understand the difference between the 'global' and 'local' concepts of neighbourhood, respectively part of street:

<city code> is a name from the **city code name class** that can be used to identify a **city** among the union of cities within the Netherlands. <neighbourhood code> is a name from the **neighbourhood code name class** that can be used to identify a **neighbourhood** among the union of neighbourhoods within city in which it resides. <part of the street code> is a name from the **part of the street code name class** that can be used to identify a specific **part of the street** among the union of part of the streets cities within a given neighbourhood in a given city.

We can conclude that the compound referencing scheme that we have introduced in this chapter of this thesis is applicable for all possible ‘aggregations’ ranging from *value-domain* aggregations to aggregations that involve exclusively *basic intentions* to aggregations that contain a *combination* of value domains and basic intentions. The incorporation of the explicit UoD in which the ‘intention’ is defined and can be identified by the names of the name class is essential for interpreting the meaning of the ‘basic intentions’ either as global or local concepts.

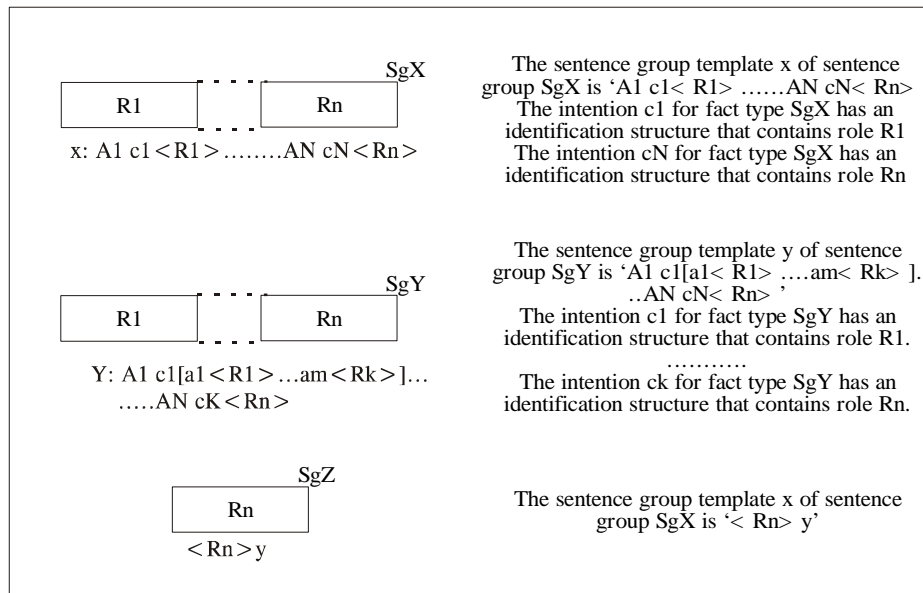


Fig. 5.8 Extended example legend for fact types.

In figure 5.8 we have given the extended legend for the (naming convention) fact types in which the referencing modes are verbalized. This legend contains the complete semantics of the graphical representations for the analyst and how these representations must be verbalized as declarative natural language sentences.

5.7 THE BASIC INFORMATION MODEL

In this paragraph we will give the definition of a basic information model.

Definition 5.7. A *Basic Information Model* (BIM) for a Universe of Discourse U is defined by

- a list of intentions and their definitions $\{(x_i, dx_i)\}$ applicable to the UoD U .
- a set of roles $R(U)$ in which each role is played by an intention from $\{x_i\}$ for which $CARD(EXT(x \in \{x_i\})) \geq 2$.
- a set of fact types $F(U)$ that consist of roles from $R(U)$ under the condition that the roles in $\{f | f \in F(U)\}$ are a partition of $R(U)$ ²⁷,
- a set of sentence group templates $S(f)$ for every fact type $f \in F(U)$ that contain(s) (alternate) descriptions of the fact type semantics and in which exactly one reference to each role contained in f is given.

An example of a basic information model of the university Enrollment example is given in figure 5.9.

$B = \{B_i | B_i \subset A\}$ is a partition if $B_i \not\subset \emptyset$ and $B_i \cap B_j = \emptyset$ and $\cup B_i = A$

| Concept | Definition |
|------------------|---|
| Student | a person that studies at Vandover University |
| Student ID | a name class |
| Major | a course program offered to students by Vandover or Ohoa university |
| Major name | a name class |
| Enrolled Student | a student who has been assigned a major |
| Course | an educational module |
| Course name | a name class |
| Credits | an award for the succesfull termination of a course |
| Natural number | a name class |

| | | | |
|--|-----|---|-----|
| Student ID | Ft1 | Major name | Ft2 |
| 1: < student ID> is a name from the student ID name class that can be used to identify a student among the union of students at Vandover University or Ohoa University | | 1: < student ID> is a name from the major name name class that can be used to identify a major among the union of majors at Vandover University | |

| | | | |
|--|------------|--------------|------|
| University code | Student ID | Chosen major | Ft10 |
| 1:Student [identified by the combination of <university code> and <student ID>] majors in major <chosen major> | | | |
| 2:Student [identified by the combination of <university code> and <student ID>] has chosen major <chosen major> | | | |

| | | | |
|--|------------|-----------|------|
| University code | Student ID | Last name | Ft11 |
| 1:Student [identified by the combination of <university code> and <student ID>] has <last name> | | | |

| | | | | |
|--|------------|-----------------|----------------|------|
| University code | Student ID | Credited Course | Course Credits | Ft12 |
| 1:Student [identified by the combination of <university code> and <student ID>] gained a number of credits <course credits> for <credited course> | | | | |

| | | | |
|---|------------|---------------|------|
| University code | Student ID | Total Credits | Ft13 |
| 1:Student [identified by the combination of <university code> and <student ID>] gained a total number of credits <total credits> in his/her freshman year | | | |

| | |
|---|------|
| Total Enrolled students | Ft14 |
| 1:There is currently a total number of enrolled students <total enrolled students> at the combined Ohoa and Vandover universities | |

| | | | |
|---|-----|---|-----|
| Course name | Ft3 | Natural number | Ft4 |
| 1: < course name> is a name from the course name name class that can be used to identify a course among the union of courses at Vandover University and Ohoa University | | 1: < Natural number> is a name from theNatural number name class that can be used to identify an amount of credits among the union of amount of credits | |

Fig. 5.9 Basic information model Ohoadover university enrolment.

In real life business settings we will always experience a specific *extension* of such a basic information model. We will now give the definition of the extension of a fact type and of the extension of a Basic Information Model for a given Universe of Discourse.

Definition 5.8. The *extension of a fact type* is a set of sentence instances for that fact type.

$$EXT(FT_k) = \{si_j/si_j \text{ is an instance of sentence group } SG_k\}$$

Definition 5.9. The *extension of a basic information model* is the union of the extensions of the fact types that are contained in that basic information model.

$$EXT(FT_k) \text{ is an extension of fact type } FT_k$$

$$EXT(BIM) = \cup_k EXT(FT_k)$$

We now have defined all the constructs necessary for analyzing natural language sentences. It was determined what parts were fixed and what parts were variable. For the variable parts or roles it was determined to what intention the variable instances refer. For each intention, a definition is contained in the basic information model of the UoD. Furthermore, the naming conventions for an intention in a role were determined. The application of these modeling constructs will lead to an abstraction of a UoD as a basic information model as for example, depicted in figure 5.9.

We will introduce in section 5.8, the modeling constructs that will enable us to express that some extensions of a basic information model are not allowed to exist. We will use part 2 of the university enrolment example 5.1 to illustrate that domain rules or business rules may exist in the UoD that do not allow some extensions of the basic information model to occur in real-life situations.

5.8 NLM MODELING CONSTRUCTS FOR THE ENCODING OF EXTENSIONAL CONSTRAINTS

If we look at part two of the University Enrollment example, we can conclude that not every extension of a BIM is an extension that is an allowed extension according to the business rules in the UoD. If we for example consider the extensions of fact types FT12 and FT13 then it becomes clear that the following example extension is not allowed according to the ‘business’ logic of the university enrolment although it is a possible extension of the BIM: *{student v 5677 gained 5 credits for the course accounting, student v 5677 gained a total number of 4 credits in his/her freshman year}*. In order to make a distinction into an extension of a basic information model regardless of the fact whether it is allowed to exist and an extension of a basic information model that is allowed to exist, according to the business rules in the application domain, we will introduce the concept of *population state*.

Definition 5.10. A *population state* is an extension of a basic information model that is allowed to exist according to the business rules in the application domain.

Let BR be a set of business rules
Let ps be a population state
 $ps \in \{EXT(BIM)_j \mid EXT(BIM)_j \text{ complies to the rules in } BR\}$

It becomes clear now that in addition to the modeling constructs that are needed for abstracting a UoD we need modeling constructs that enable us to specify additional limitations on the possible extensions of a basic information model. These modeling constructs we will call constraints. We will now give the definitions of the four constraint groups that we want to introduce in this thesis and the instances of these constraint groups that reflect (parts of) the business rules in our university enrolment example.

5.8.1 Definition of Population state constraints

This first group of constraints specify the limitations on the extensions of a basic information model that exist at any point in time.

Definition 5.11. A *population state constraint* p in a basic information model BIM is a proposition that limits the allowed extensions of the basic information model BIM to those extensions for which the proposition of p is true.

A population state constraint is a set valued function into the set of extensions of a basic information model of a universe of discourse.

$PC: \quad \{EXT_j(BIM)\} \text{ ----> } \{EXT_j(BIM)\}$

The type of constraints in this category are the *uniqueness constraints* and *set comparison constraints* that must apply in every possible state of the UoD (see section A.2 in appendix A). In the university enrolment example, we have derived four uniqueness constraints: $C1$, $C2$, $C3$ and $C4$ and nine set-comparison constraints: $C5$, $C6$, $C7$, $C8$, $C9$, $C10$, $C10$, $C11$, $C12$, $C13$. In figure 5.10 we have illustrated these constraints by adding them to the basic information model of our University Enrollment example from figure 5.9. See section A.2 in appendix A, for a formal definition and verbalization legends for this constraint type.

5.8.2 Definition of Population state transition constraints

The population state transition constraints specify the limitations on subsequent extensions of a basic information model.

Definition 5.12. A *population state transition constraint* q in a basic information model BIM is a proposition that limits the before-after extension combinations of the basic information model BIM to those combinations for which the proposition of q is true.

A population state transition constraint is a set valued function into the set of before-after extensions of a basic information model of a universe of discourse.

$$PTC: \quad \{ EXT_j(BIM) \} \times \{ EXT_j(BIM) \} \rightarrow \{ EXT_j(BIM) \} \times \{ EXT_j(BIM) \}$$

The transition constraints constrain the possible state sequences of the extension of the basic information model. Even if an extension of the BIM complies to the population state constraints, the allowed before/after combinations are further constrained by these state transition constraints. Constraint *C14* in figure 5.10 is an example of a state transition constraint that reflects some business rule from our university enrolment example. See also section A.3 in appendix A for a formal definition and a verbalization legend of this constraint type.

5.8.3 Definition of Derivation rule constraints

In addition to the population state- and population state transition constraints that limit the possible extensions of a basic information model in terms of for example uniqueness and set-comparison restrictions, a different group of constraints is needed that is able to specify limitations on the fact values of roles from the basic information model. We will call this type of constraint: *a derivation rule constraint*.

Definition 5.13. A *derivation rule (constraint)* is a function defined on instances of the ingredient fact types. The function range is a set of resulting fact instance(s) from the derived fact type.

Let FT_1 through FT_N be ingredient fact types for the derivation rule CP

Let FT_M be the resulting fact type for the derivation rule CP

$$CP: EXT(FT_1) \times \dots \times EXT(FT_N) \text{ -----} > \quad EXT(FT_M)$$

The derivation rule constraints, specify that instances of a given fact type can not be inserted or updated freely, but their value is restricted to the pre-conditions and derivation formula of a derivation rule constraint. In the university enrolment example, we have derived two derivation rule constraints: *C15* and *C16* (see figure 5.11). In section A.4 in appendix A a formal definition and a verbalization legend for this constraint type are given.

5.8.4 Definition of Event occurrence, Event, Event type and Impulse type constraints

In this section we will give a definition of the *event*, *event type* and *event occurrence* concepts and the group of constraints that constrain the behaviour within a UoD: the *impulse type constraints*.

In order to define the impulse type of constraints we need to define the concept of event occurrence first.

Definition 5.14. An *event occurrence* is a happening at a certain point in time in the application subject area that can lead to the execution of one or more derivation rules and/or the insertion or deletion of fact instances into/from the application's information base.

Let PH be the set of potential happenings
Let eo be an event occurrence
 $eo \in PH$

From definition 5.14 it follows that an event occurrence is a 'one-time' only thing. For example the event occurrence: *student 'V 2345' wants to enroll for major 'science' at 12:45:56 on 01/12/2004*. A different event occurrence is: *student 'V 2345' wants to enroll for major 'science' at 18:45:56 on 03/06/04*. We can group the former two event occurrences into the following event: *student 'V 2345' wants to enroll for major 'science'*.

Definition 5.15. An *event* is one or a number of potential happenings in the application subject area that can lead to the execution of one or more derivation rules and/or the insertion or deletion of fact instances into/from the application's information base.

Let e be an event
 $e \subset PH$

Definition 5.16. An *event type* is a class of events in the application subject area, each of these events can lead to the execution of one or more derivation rules (of the same type) and/or the insertion or deletion of fact instances (of the same fact types(s)) into/from the application's information base.

Let ET be an event type
Let $E = \{e_i\}$ be the set of events
 $ET \subset E$

Definition 5.17. An *impulse type (constraint)* is an ordered triplet that contains an event type, a condition type²⁸ under which the occurrence of an event of an event type can lead to the execution, of a specified derivation rule constraint or insert/delete operation.

Let IT be an impulse type
Let SET be the set of event types
Let SCT be the set of condition types
Let SDR be the union of the set of derivation rule constraints and the set of insert/delete operations
 $IT = (A, B, C) \mid A \in SET, B \in SCT, C \in SDR$

The impulse type constraints explicitly model the temporal relationships between ‘happenings’ or events in the UoD and information system events and enforces them upon the derivation rules and information base update operations. In figure 5.11 we have given the instances of the impulse type constraint *C17*, *C18*, *C19*, *C20*, *C21* and *C22* for our University Enrollment example part 2.

In appendix A we will give a detailed specification and illustration of the constraint types from this section that we consider relevant for a deeper understanding of the NLM requirements specification language. These constraint types cover the required static and dynamic constraints from chapter 2. In addition to the concepts of extension of the BIM, population we need to define a third state concept that reflects the extent in which the constraint types are enforced in the extension of the BIM.

Definition 5.18. An *information base state* is an extension of a basic information model that complies with those business rules in the application domain that can be encoded using the population state-, population state transition-, derivation rule- and impulse type constraints.

Let SEXT(BIM) be the set of extensions of a basic information model BIM
Let IBS be an information base state of a basic information model BIM
Let SPS be the set of population states for BIM
Let PS be the set of population state constraints defined on BIM
Let PST be the set of population state transition constraints defined on BIM
Let DR be the set of derivation rule constraints defined on BIM
Let IT be the set of impulse type constraints defined on BIM
Let BR be a set of business rules
Let ps be a population state ($ps \in PS$)
 $ps \in \{EXT(BIM)_j \mid EXT(BIM)_j \text{ complies to the rules in } BR\}$
 $IBS \subseteq (SEXT(BIM) = \{EXT(BIM)_j \mid EXT(BIM)_j \text{ complies to the constraints in } (PS \cup PST \cup DR \cup DT)\})$
 $SPS \subseteq IBS \subseteq SEXT(BIM)$

²⁸ Including the ‘empty’ condition type, which means that the occurrence of an event will unconditionally lead to the execution of a derivation rule and or/insert delete operation(s)

5.9 THE NLM REQUIREMENTS SPECIFICATION FOR A GIVEN UOD

We will now give a definition of a NLM requirements specification for a given UoD.

Definition 5.19. A NLM *requirements specification* referring to a UoD is a basic information model (BIM) for that UoD together with all population constraints for which a legend is defined, derivation rule constraints and impulse type constraints, that reflect the business rules in that UoD and which fulfill the following conditions:

- The intentions that are contained in a derivation rule argument should be contained in the Basic Information Model (BIM)
- A post-condition in a derivation rule constraint should be a proposition on the Basic Information Model, e.g. it should only reference role and fact types that are contained in the BIM.
- A pre-condition in a derivation rule constraint should be a proposition on the Basic Information Model, e.g. it should only reference roles and fact types that are contained in the BIM.
- A formula in a derivation rule constraint should be a function on the Basic Information Model, e.g. it should only reference role and fact types that are contained in the BIM and (possibly) values in the derivation rule argument.
- A condition in an impulse type constraint should be a proposition on the Basic Information Model, e.g. it should only reference roles and fact types that are contained in the BIM and/or values in the event argument.
- A derivation rule that is triggered in an impulse type constraint should be a derivation rule constraint for that UoD
- An insert or delete operation that is triggered in an impulse type constraint should only refer to a fact type in the BIM of that UoD.

Let RS be an requirements specification for the UoD U

Let BIM be a basic information model for the UoD U

Let {psc} be the set of population state constraints for the UoD U.

Let {ptc} be the set of population state transition constraints for the UoD U.

Let {drc} be the set of derivation rules constraints for the UoD U.

Let {itc} be the set of impulse type constraints for the UoD U.

$$\begin{aligned}
 RS = & \{ BIM \cup \{psc\} \cup \{ptc\} \cup \{drc\} \cup \{itc\} \mid \\
 & \forall r \in psc [r \in BIM]; \forall r \in ptc [r \in BIM]; \forall r \in drc [r \in BIM] \wedge \\
 & \forall prc, poc \in drc [prc \overset{29}{\bullet} BIM \wedge poc \bullet BIM] \wedge \forall ic \in itc [ic \bullet BIM] \\
 & \wedge \forall dr \in itc [dr \in \{drc\} \vee dr \bullet BIM] \}
 \end{aligned}$$

The consistency of the NLM modeling elements that are applied in the Basic Information Model and the model elements that are used in the different constraint types can be enforced by constraints in the NLM requirements specification of the

²⁹ The $x \bullet y$ operator means that x references an element in y.

NLM requirements specification (or the NLM meta model) in figure B.1 of appendix B.

5.9.1 The NLM Requirements specification for the University Enrolment UoD

We will give the NLM requirements specification for our integrated university Enrollment example in figures 5.10, 5.11 and 5.12. In figure 5.10 the *basic information model* for the University Enrollment example is shown plus the *population state*- and *population state transition* constraints. In figures 5.11 and 5.12, the *derivation rule*- and *impulse type* constraints for this UoD are given.

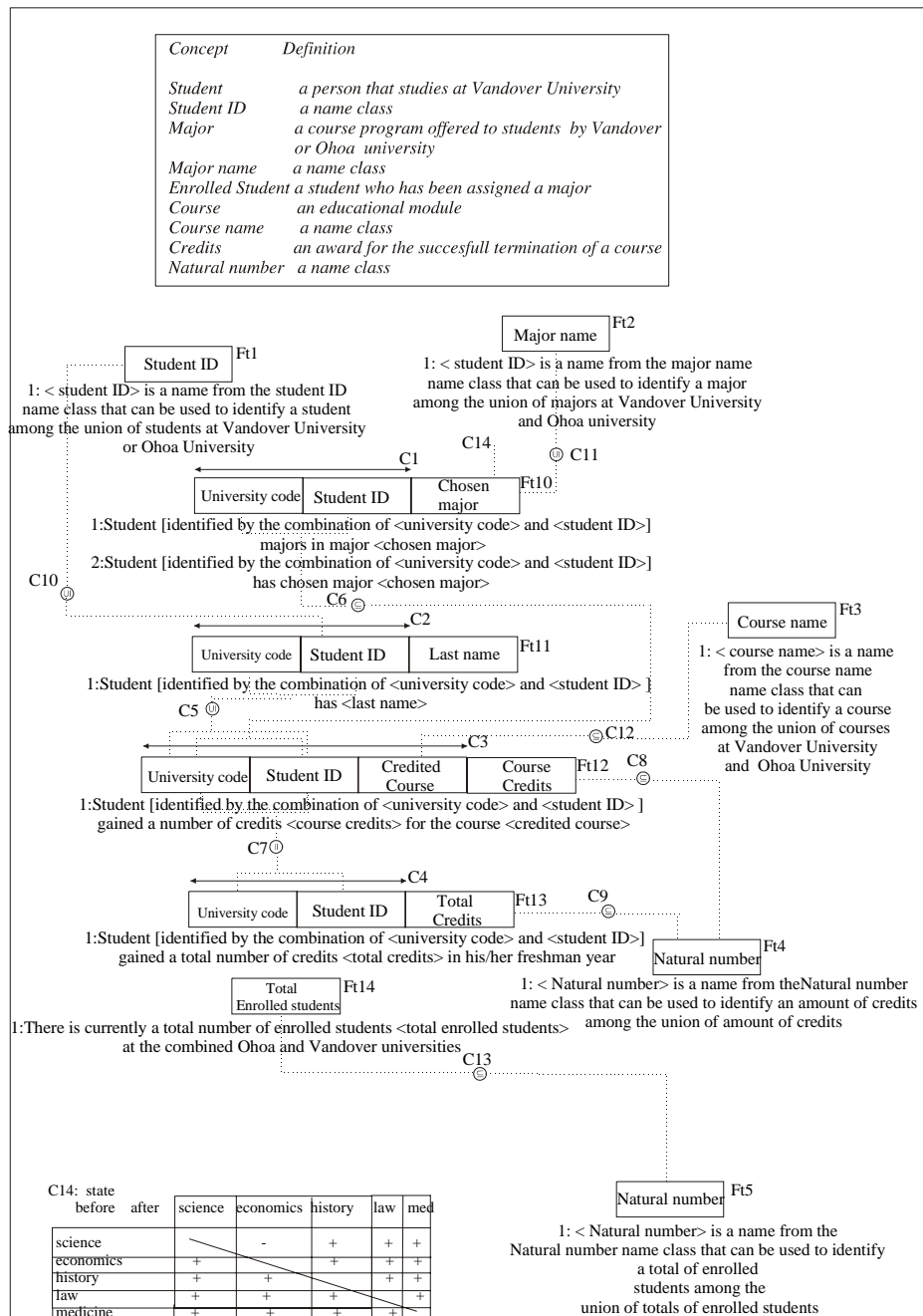


Fig. 5.10 NLM requirements specification (I):BIM and population constraints

| |
|--|
| <p>C15: Create total number of credits<{(arg₁,student)}> IF there exist an instance of FT12 SUCH THAT FT12.<university code>.<student ID>=arg1 THEN create an instance of fact type FT13 SUCH THAT FT13.<university code>.<student ID>:= arg1 FT13.<total credits>:=DF1 DF1:= FT12.<credits> [where FT12.<university code>.<Student ID>='arg1'] ENDIF</p> |
| <p>C16: Create total number of enrolled students IF there exist an instance of FT10 THEN create an instance of fact type FT14 SUCH THAT FT14.<total enrolled students>:=DF2 DF2:= COUNT(Ext(FT10)) ENDIF</p> |
| <p>C17 ON E T2: Insert(Student'x' wants to enroll in Major 'y') into application data base has succeeded (arg1:'x'; arg 2: 'y') DO Create total number of enrolled students</p> |
| <p>C18 ON ET3: Delete(Student'x' wants to enroll in Major 'y') from application data base has succeeded (arg1:'x'; arg 2: 'y') DO Create total number of enrolled students</p> |
| <p>C19 ON ET1: student requests enrollment in major(arg1: student, arg2:major) IF[FT13.<total credits> (Where FT13.<university code>.<Student.ID>='ET1.arg1')] > 24 AND [IF ET1.arg2='science' THEN(mathematics EXT (FT12.<credited course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND FT12.<course credits>[where FT12.<university code> . <Student.ID> = 'ET1.arg1' AND where FT12.<credited course > ='mathematics']>8) OR [IF ET1.arg2='history' THEN(language and culture EXT (FT12.<credited course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND FT12.<course credits>[where FT12.<university code> . <Student.ID> = 'ET1.arg1' AND where FT12.<credited course > ='language and culture']>5) OR [IF ET1.arg2='economics' THEN(macro econ. EXT (FT12.<credited course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND FT12.<course credits>[where FT12.<university code> . <Student.ID> = 'ET1.arg1' AND where FT12.<credited course > ='macro econ.']>8) OR [IF ET1.arg2='medicine' THEN(biology. EXT (FT12.<credited course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND FT12.<course credits>[where FT12.<university code> . <Student.ID> = 'ET1.arg1' AND where FT12.<credited course > ='biology']>5) OR [IF ET1.arg2='law' THEN(biology. EXT (FT12.<credited course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND FT12.<course credits>[where FT12.<university code> . <Student.ID> = 'ET1.arg1' AND where FT12.<credited course > ='finance']>5)] DO Insert (student'ET1.arg1' has chosen major 'ET1.arg2').</p> |

Fig. 5.11 NLM requirements specification (II): derivation rule- and impulse type constraints

| |
|--|
| C20 ON Et4: Insert(Student 'x' has gained the number of 'y' course credits for course 'z') into application data base has succeeded (arg1: 'x'; arg 2: 'y'; arg3: 'z') DO Create total number of credits(arg, := 'Et4.arg1') |
| C21 ON Et5: Credits granted to student(arg1: student; arg2: course; arg3: credits) IF ET5.arg1 EXT(FT11.<Universitycode>.<student ID>) DO Insert (Student 'Et5.arg1' has gained the number of 'Et5.arg2' credits for course 'Et5.arg3') |
| C22 ON Et6: Student graduates(arg1: studen) IF ET6.arg1 EXT(FT11.<Universitycode>.<student ID>) DODelete(Student'Et6.arg1' wants to enroll in Major 'y') |
| C23 ON ET1: student requests enrollment in major(arg1: student, arg2: major) IF[FT13.<total credits> (Where FT13.<university code>.<Student.ID>='ET1.arg1')] > 24 AND [IF ET1.arg2='science' THEN(mathematics EXT (FT12.<credited course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND FT12.<course credits>[where FT12.<university code> . <Student.ID> = 'ET1.arg1' AND where FT12.<credited course > ='mathematics']>8) OR [IF ET1.arg2='history' THEN(language and culture EXT (FT12.<credited course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND FT12.<course credits>[where FT12.<university code> . <Student.ID> = 'ET1.arg1' AND where FT12.<credited course > ='language and culture']>5) OR [IF ET1.arg2='economics' THEN(macro econ. EXT (FT12.<credited course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND FT12.<course credits>[where FT12.<university code> . <Student.ID> = 'ET1.arg1' AND where FT12.<credited course > ='macro econ.']>8) OR [IF ET1.arg2='medicine' THEN(biology. EXT (FT12.<credited course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND FT12.<course credits>[where FT12.<university code> . <Student.ID> = 'ET1.arg1' AND where FT12.<credited course > ='biology']>5) OR [IF ET1.arg2='law' THEN(biology. EXT (FT12.<credited course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND FT12.<course credits>[where FT12.<university code> . <Student.ID> = 'ET1.arg1' AND where FT12.<credited course > ='finance']>5) AND IF EXT (FT10.<chosen major>[where Ft10.<university cod>. <student ID>='ET1.arg1")]] DO Delete (student'Et1.arg1' has chosen major 'z'). Insert (student'Et1.arg1' has chosen major 'ET1.arg2'). |

Fig. 5.12 NLM requirements specification (III): impulse type constraints

5.9.2 The University Enrollment case study description revisited

In the following we have summarized part 2 of the University Enrollment example text from section 5.6.2 in which we have references to the relevant *fact types*, *event types* and *constraints* that encode a specific business rule from the text.

For our application it suffices to know that student can be identified by the combination of a University code and a locally unique student ID (*Ft1*, *c10*). Furthermore, we assume that the assignment of a student's university ID, which takes place outside our example UoD, will always lead to the recording of a student's last name (*Ft11*, *c5*). In this example we assume that the University system is closed in the sense that a major can only be selected by students that are currently doing a freshmen year at Ohoa and/or Vandover (*et1*). A student can apply for at most one major at a time (*c1*). The majors that are offered after the merger are a simple union of the existing majors at the 'old' universities: *science*, *history* and *economics* at Vandover University and *medicine* and *law* at Ohao University (*c14*).

In the integrated Ohoa and Vandover enrolment system it is decided to record all courses (*Ft12*) and the credits (*Ft13*) that have been obtained by them during their student's freshmen year at Ohoa or Vandover (*c7*). The definite enrolment in the major of their choice depends upon the number of credits that a student has earned in his/her freshman year (*Ft10*, *c6*). If the total number of (approved) credits for these freshmen courses is 24 or more and the specific required freshmen courses for the major of their choice are contained in their credited freshmen courses than a definite enrolment, for the student will always be recorded (*c19*).

A 'real-life' user example of the integrated Ohoadover enrolment system is given in figure 5.5. (*Fact types Ft10, Ft11, Ft12, Ft13, Ft14 and constraints c2, c3, c4 are abstracted respectively derived from this example*)

In the example of figure 5.5 we furthermore have Majors that can be identified by a major name among the union of majors at Vandover and Ohoa University (*Ft2*). We also have courses that can be identified by a course name among the union of courses at Vandover and Ohoa University (*Ft3*, *c12*).

A course credit (*Ft4*) or a total number of course credits (*Ft4*) for a student or the total number of enrolled students (*Ft5*) are expressed by a natural number (*c8*, *c9*, *c13*). At any point in time a student can have at most one total number of credits (*c4*). The total number of credits for a student is the arithmetic total of all individual credits for credited courses (*c15*). If a student does not have any credits assigned for at least one course the total number of course credits will not be shown (*c5*, *c7*).

There will be some kind of message coming from another part of the university system that acknowledges at a certain point in time that *student X has been credited Y credits for course Z*. (*c21*). As soon as such a message is received by the enrolment clerk the information is entered into the enrolment system (*et5*). After the new course credits have been entered into the enrolment system (*et4*) the total number of credits for that student will be recalculated (*c20*).

Furthermore, students are allowed to switch majors before graduation (*et7*). In that case the requirements regarding the content of their freshmen course and credits needed for this new major will be checked again (*c23*). In addition the management of Ohoadover has decided that not all enrolment switches are allowed. At this time the following restriction is applied: A student can not major in *Economics* after he has majored in *Science* (*c14*).

After a student has successfully finished his/her current major (this is decided outside the scope of our example UoD) (*et6*) he/she will be removed from the University Enrollment system (*c22*). Every time a student is enrolled (*et2*) or graduated (*et3*) the enrolment system will recalculate the total number enrolled students (*c17*,

c18). The total number of enrolled student at any point in time is calculated by inspecting which of the registered students have already enrolled for a major (*c16*).

5.10 CONCLUSIONS ON THE WAY OF MODELING IN THE NLM RDM

In this chapter of this thesis we have given the definitions of the modeling concepts for the requirements specification language of the NLM requirements determination method. In appendix B we have given the accompanying (meta) grammar for the application of these concepts in a requirements determination project. The way of modeling in NLM caters for the modeling of static as well as the dynamic rules of a business organization or an enterprise.

We have introduced the fact type as single information bearing construct. The fact type construct that we have defined in this chapter allows us to model any naming convention and semantic connection. The introduction of the sentence group template construct and the application concept repository allows us to capture the complete domain semantics of a UoD. We conclude that NLM fulfills *RMD 1* from chapter 4.

The introduction of naming convention fact types and compound referencing schemes share the same general structure and provide in combination with an accompanying sentence group template the context in which the naming convention is valid. The existence of this unified naming convention/referencing concept in NLM fulfills requirement *RMD 2* from chapter 4.

The definition of the role construct as a mandatory modeling construct in NLM and the mandatory naming convention for such a construct within the UoD of an analyst in the NLM specification language fulfills *RMD3* from chapter 4.

Furthermore we have given modeling provisions that allows us to define any type of static constraint that currently exists within the approaches that we have investigated: uniqueness and set-comparison constraints in chapter 3, this results in a compliance with *RMD 4*.

With respect to the definition of the modeling constructs in this chapter we have defined them in a way that the modeling constructs for the basic information model and the modeling constructs for constraining the possible extensions of the basic information model allow us to model ‘atomic’ chunks of business knowledge. This means that a NLM requirements specification complies with requirement *RMD 5* from chapter 4 because the basic information model can evolve with the changing UoD by adding or deleting ‘atomic’ fact types and population, derivation rule and impulse constraints.

The definition of an intention from the ACR (that has at least an extension of two elements) implies that such an intention plays at least one role in the basic information model. If instances of such an intention can only exist on their own, this must be explicitly modeled as a unary ‘existence postulating’ fact type (ter Hofstede et al., 1997:352). This means that the NLM modeling constructs comply with requirement *RMD 6* from chapter 4.

Business rules that can be phrased as propositions on the state of the application information base (static constraints) we have coined *population state*

constraints. We have given a legend and a definition of two groups of population state constraints: *uniqueness constraints* and *set-comparison constraints*. From their definition it follows that these constraint types do not intersect in terms of the implications for the allowed extensions of the intentional model, they exist independently and their definition is not dependent upon the arity of the underlying modeling construct from the basic information model, e.g. the definition of the *uniqueness*- and *set-comparison* constraints is fully scalable as a function of the arity (N). The definition of these constraint types caters for an explicit reference to the roles of the data model on which the constraint(s) is (are) defined. This means that we have complied with requirement *RMD 7* from chapter 4.

With respect to the dynamic constraints we have introduced language concepts for business rules that can be phrased as propositions on two subsequent states of the application information base. We have called these types of constraints: *population state transition constraints*. With respect to the transition constraints we remark that in our legend we have explicitly incorporated the relationship that the constraint has in terms of the values of the roles that are involved. The definition of a state transition constraint in NLM contains explicit references to *before*- and *after*- states of the application information base, therefore, NLM complies with *RMD 8* from chapter 4.

Business rules that can be phrased as transformations of *ingredient fact instances* into (an) derived fact instance(s) we have coined *derivation rules* that can reference some kind of passive constraint, e.g. a proposition on the information base that might be true or false at any point in time. Such a proposition we have called a *pre-condition*. Furthermore, the derivation rule constraints contain a reference to the roles from the Basic Information Model of the UoD and we have given a legend of how derivation rule constraints can be created in which the derivation logic can be encoded using some formalism that is known to the analyst. This means that NLM complies with *RDM 9*

An impulse models those dynamic business rules in which the occurrence of an event can lead to the execution of a derivation rule or the insertion and/or deletion of fact instances into/from the application's data base. We have made a distinction into internal and external events in NLM. This leads to the compliance to *RMD 10*. In the impulse, an information base condition can be contained. Such an information base condition (IBC) is evaluated at some point in time. If the application information base at that point in time in combination with the information base condition yields the value true then the derivation rule and/or insert/delete operation will be executed. If it yields the value false nothing will happen. This means that requirement *RMD 11* from chapter 4 has been fulfilled by the NLM way of modeling.

5.10.1 The added value of the NLM requirements specification language

In this chapter we have introduced the modeling constructs in Natural Language Modeling (sections 5.1 through 5.9). The added value in terms of understanding the requirements determination process of NLM, in addition to fulfilling the operationalized design criteria that were mentioned in the former section, is in the consistent application of the modeling primitives onto the UoD of the requirements analyst itself. We also have provided a notation legend that enables us to use any graphical convention for denoting the language concepts in the NLM requirements specification language.

We conclude by stating that the way of modeling in NLM is *complete* because it contains all the necessary modelling constructs for the conceptual specification of requirements in the static and dynamic perspectives of an enterprise. Furthermore, the way of modeling in NLM is *efficient*, because the average number of modeling constructs that serve the same purpose is very small if compared to the UML, (E)ER and ORM requirements specification languages. This means that a prospective NLM analyst can learn and apply these modeling constructs with less effort than learning for example UML. In appendix B this is illustrated by a significant part of the NLM meta model. The *consistency* of the modeling constructs for the data model and the constraints defined on its extension in NLM is guaranteed, because the definition of these modeling constructs was based upon the same set of primitives.

5.11 REFERENCES

- Abrial, J. (1974): Data Semantics. In: Klimbie, J. , Koffeman, K. (eds.): Data Base Management, North Holland, Amsterdam: 1-59
- Ambrosio, A., Metais, E., Meurier, J-N. (1997): The linguistic level: Contribution for conceptual design, view integration, reuse and documentation. Data & Knowledge Engineering **21**: 111-129
- Broekstra, J., Klein, M., Decker, S., Fensel, D., van Harmelen, F., Horrocks, I. (2002): Enabling knowledge representation on the web by extending RDF schema. Computer Networks **39**: 609-634
- Bubenko, A., Wangler, B. (1992): Research Direction in Conceptual Specification Development. In: Conceptual Modeling, Databases, and Case (Loucopoulos, P., Zicari, R. (eds.)). Wiley. 389-412
- Chakravarthy, S., Mishra, D. (1994): Snoop: An expressive event specification language for active databases. Data & Knowledge Engineering **14**: 1-26
- Choobineh, J., Venkatraman, S. (1992): A methodology and tool for derivation of functional dependencies from business forms. Information Systems **17**(30): 269-282
- Conlon, S., Conlon, J., James, T. (2004): The economics of natural language interfaces : natural language processing technology as a scarce resource. Decision Support Systems **38**: 141-159
- Davies,J., Duke, A., Stonkus, A. (2002) : Ontoshare: Using ontologies for Knowledge Sharing. In Proceedings of the WWW2002 Semantic Web workshop, 11th International WWW Conference WWW2002, Hawaii, USA

- Davies, J., Weeks, R., Krohn, U. (2004): QuizRDF: Search Technology for the Semantic Web. Proceedings of the 37th annual Hawaii international conference on system sciences (HICSS'04)
- Dayal, U., Hsu, M., Ladin, R. (1990): Organizing Long-Running Activities with Triggers and Transactions. ACM-SIGMOD international conference on management of data 1990: p. 204-214.
- De, P., Sen, A. (1984): A new methodology for database requirements analysis. MIS quarterly, september: 179-193.
- Falkenberg, E. (1976a): Concepts for Modelling Information. In: Nijssen, G. (ed.), Modelling in Database Management Systems, North-Holland, Amsterdam : 95-109
- Falkenberg, E. (1976b): Significations: the key to unify data base management. Information Systems **2**:19-28
- Grönbaek, K., Sloth, L., Bouvin, N.O. (2000): Open hypermedia as user controlled meta data for the web. Computer Networks **33**: 553-566.
- Halpin, T. (1995): Conceptual schema and relational database design: a fact based approach. 2nd ed. Prentice-Hall, Englewood Cliffs
- Halpin, T., Orlowska, M.(1992): Fact-oriented Modelling for Data Analysis. Journal of Information Systems **2**: 97-118
- Henderson-Sellers, B., Edwards, J. (1990): The object-oriented systems life-cycle. Communications of the ACM **33**(9): 143-159.
- Ter Hofstede, A., Proper, H., van der Weide, T. (1997): Exploiting fact verbalisation in conceptual modelling. Information Systems **22**(6/7): 349-385.
- van der Lek, H., Bakema, G., Zwart, J. (1992): De unificatie van objecttypen en feittypen. Informatie **34**(5): 279-295 (in dutch)
- Leung, C. , Nijssen, G. (1988): Relational Database design using the NIAM conceptual schema. Information Systems **13** :219-227.
- McFadden, F., Hoffer, J. (1994): Modern database management, 4th edition, Benjamin/Cummings.
- Nijssen, G. (1989): An Axiom and Architecture for Information Systems. In: Falkenberg, E., Lindgreen,P. (eds.): Information System Concepts : An In-depth analysis, Elsevier Science publishers North-Holland, Amsterdam: 157-175
- Nijssen, G., Bollen, P. (1995): Universal Learning: A science and methodology for education and training, in: W,Gijselaers, D. Tempelaar, P. Keizer, J. Blommaert, E.

Bernard & H. Kasper (eds.), Educational innovation in economics and business administration: the case of problem-based learning, pp. 428- 435.

Prabhakaran, N., Falkenberg, E. (1988): Representation of Dynamic Features in a Conceptual Schema. Australian Computer Journal **20** (3): 98-104

Resource Description Framework (RDF), model and syntax specification, W3C recommendation, 10 february 2004, <http://www.w3.org/TR/REC-rdf-syntax/>

Rolland , C. (1983): Database dynamics. Data base, spring 1983: 32-43.

Senko, M. (1976): DIAM as a detailed example of the ANSI/SPARC architecture. In: Nijssen, G. (ed.): Modelling in Database Management Systems, North-Holland, Amsterdam.

Smith, J., Smith, D. (1977): Database abstractions: Aggregation. Communication of the ACM **20**(6): 405-413

Sowa, J. (1984): Conceptual structures: information processing in mind and machines. Addison-Wesley, Reading, Ma.

Verheijen, G., van Bekkum J. (1982): NIAM: An Information Analysis Method. In: Verrijn-Stuart,A., Olle T., Sol H., (eds.): Proceedings of IFIP TC-8 Conference on Comparative Review of Information Systems Methodologie (CRIS-1), North-Holland Amsterdam 537-590

CHAPTER 6³⁰

THE WAY OF WORKING AND THE WAY OF CONTROLLING IN NATURAL LANGUAGE MODELING

6.1 INTRODUCTION

A requirements determination method is a combination of a set of modeling constructs (in general referred to as the requirements specification language) and an accompanying (set of) procedure(s) that ‘prescribe’ how this language must be applied in practice. In this chapter a *way of working* and a *way of controlling* for *natural language modeling* (NLM) will be provided. The modeling constructs in the NLM requirements specification language are based upon the axiom that all verbalizable information (reports, web-pages, note-books, traffic signs and so forth) can be translated into declarative natural language sentences (natural language axiom) (see chapter 5). Taking this axiom as a starting point in requirements modeling will constrain the feasible requirements specification constructs to those constructs that enable analysts to model *declarative natural language sentences*. It also means that it is *not* a real or abstract world that is subject to modeling, but that it is the *communication about* such a real or abstract world (Hoppenbrouwers et al., 1997:79; van der Lek et al., 1992:279). In chapter 5 we have introduced the NLM requirements specification language or *the way of modeling*. In the first part of this chapter we will focus on the NLM modeling procedures or *way of working*. In the last part of this chapter we will discuss the *way of controlling* when NLM is used as a requirements determination method.

Earlier work on natural language descriptions in the context of requirements determination include Wrycza (1990), Choobineh and Venkatraman (1992) and Silva and Carlson (1995). Although these approaches all recognize the importance of natural language specifications, they lack the explicit incorporation of domain knowledge to be provided by the domain user into the way of working of a requirements determination method. Hoppenbrouwers et al. (1997) acknowledge the importance of natural language in the communication between user and analyst. Capuchino et al. (2000) give a linguistics based conceptual modeling approach in which they need the output of a requirements elicitation process as a starting point for the derivation of conceptual OO-specifications. Tseng and Mannino (1989:53-54) give a survey of earlier work on the field of user view modeling based upon forms or examples.

³⁰ An earlier version of this chapter has been published as ‘The Natural Language Modeling Procedure’, P.Bollen, in: A. Halevy and A. Gal (eds.), proceedings Fifth Workshop on Next Generation Information Technologies and Systems (NGITS’2002). Lecture Notes in Computer Science **2382**. Springer-Verlag Berlin, Heidelberg (2002) 123-146.

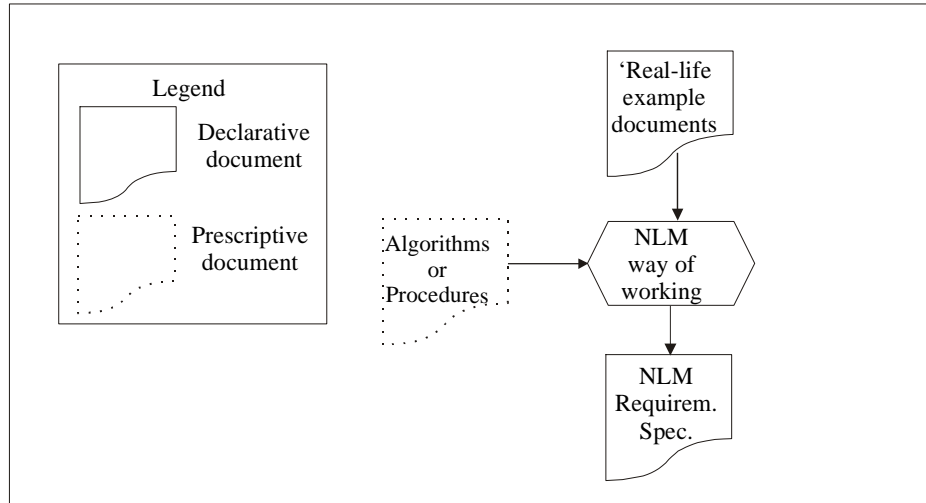


Fig. 6.1 The way of working in the NLM requirements determination method

The way of working in NLM is laid down in a *prescriptive* document (see figure 6.1). Such a prescriptive document not only tells the NLM analyst *what* to do but more importantly it must contain precise guidelines on *how* to do it and moreover it should explicitly state *when* what type of user input is needed in order to arrive at a complete, precise and consistent requirements specification for a given application subject area. We will define this completeness, preciseness and consistency within the context of an agreed upon set of explicit informational documents (or formal organization) of the application UoD that is the subject of analysis. The extent in which an informal organization exists and the extent in which the existence of such an informal organization is acknowledged by the management and the requirements determination analysts severely effects the quality of a requirements specification (Land, 1980; Oonincx, 1982:73-76).

In this chapter of this thesis we will exactly specify *when* a *specific type* of user input is required. These user inputs can be considered *semantic bridges* between the ‘real-world’ that is subject to modeling and the ‘model-world’ that consists of projections of such a real-world in terms of the requirements specification language in the methodology that is used.

6.1.1 Organization of chapter 6

In section 6.2 through 6.8 we will specify the way of working in Natural Language Modeling. In section 6.2 we will specify how a NLM analyst can create a basic information model for a UoD. The specification of the way of working will be done by introducing three algorithms that capture the domain semantics from the examples in the UoD and result in a set of naming conventions and a set of sentence groups. In section 6.3 an algorithm is provided that enables an analyst to derive an atomized basic information model for an individual analyst/user interaction for a given (set of) example(s). In section 6.4, we will introduce the way of working for those cases in

which a requirements determination project is divided into sub-projects in which different domain users interact with different analysts most of the time over different ‘real-life’ examples. The integration algorithm specifies how the basic information models from the sub-projects can be integrated in a way that preserves the overall domain semantics. In sections 6.5 through 6.8, respectively we will provide the modeling procedures for the derivation of population state-, population state transition-, derivation rule- and impulse constraints. In section 6.9 we will discuss the way of controlling in NLM.

6.2 THE DESIGN PROCEDURE FOR A SIMPLE BASIC INFORMATION MODEL

In the remainder of this chapter we will define the procedures that specify how a NLM requirements specification for a given Universe of Discourse (UoD) can be created. The starting point for every requirements determination process will be a (set of) *real-life user example(s)* that represent(s) a specific ‘external’ user view on the subject area. We note that the application of ‘real-life’ user examples is not restricted to ‘as-is’ situations but also applies to projected ‘real-life’ examples of a ‘to-be’ or ‘reengineered’ application domain. The only requirement is that a user example must contain verbalizable information. Wu et al. (2002) give an overview of previous research in which ‘forms’ are used for information system design³¹.

The real-life user examples, however, can only be used in the requirements determination process when the people that use these examples are involved in this requirements determination process. The assumption that we will use in this chapter of this thesis is that users will be available that have the discretion within the organization to ‘verbalize’ these user documents in a knowledgeable way and that will be able to accept and or reject (combinations) of real-life example documents further on in the requirements determination process, based upon the knowledge of the application UoD that they possess. Finding these domain users in some UoD’s is not a trivial task, in some organizational forms, e.g. professional bureaucracies, political issues might have a big impact on the availability and selection of the users that may/can participate in the requirements determination process because it is not always evident from the ‘formal’ organizational structure who has the discretion to act on behalf of ‘the organization’ in this organizational type (Mintzberg, 1991: 207-211).

Legacy systems may contain hidden logic which cannot be verbalized by present or past users. In such cases we assume that reverse engineering leads to artificial-life user examples.

³¹ Sometimes the concept of ‘form’ is used to refer to any structured document (e.g. See Choobineh and Venkatraman (1992:p.270).

Definition 6.1. A user example is an informational document. An informational document can have several manifestations. It can be paper-based, it can be a web-page, a computer screen, a note-book, and it can even be a formatted conversation. An informational document should contain *verbalizable* information.

An example of a document that is *not* an informational document and therefore not verbalizable is for example an ‘art’ painting by Rembrandt or van Gogh or a satellite picture of a city (you can see things but no single individual can name all of the things he/she sees on the picture).

6.2.1 The verbalization transformation

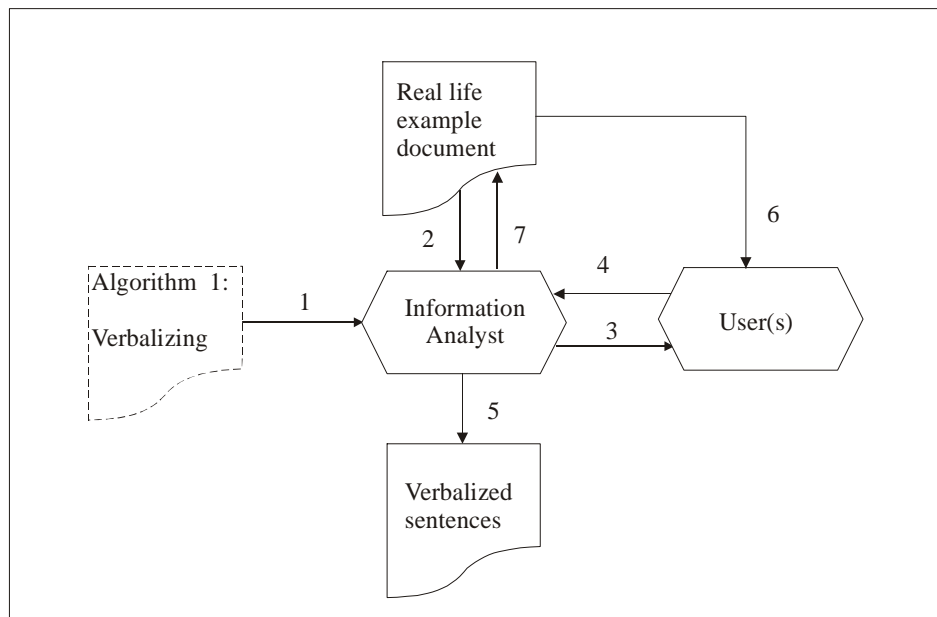


Fig. 6.2 The verbalization transformation.

In NLM the requirements specification language constructs are applied in the analysis of the structure of declarative natural language sentences (as laid down in the natural language axiom). This means that we must translate ‘real-life’ examples that are not in the format of ‘declarative natural language sentences’ into *declarative natural language sentences*. The first step in this modeling procedure, therefore is the ‘visual-to-auditory’ transformation (Nijssen, 1986) that will ‘standardize’ each *verbalizable* example into declarative natural language sentences.

During this transformation the user is asked to verbalize the content of a verbalizable 'real-life' user document that **not** necessarily should contain written natural language sentences. The result of this verbalization is a set of (verbalized) sentence instances. In figure 6.2 the *information flow diagram* is given for the *verbalization* transformation. When the *verbalization* transformation is executed the following sequence of actions should take place. Firstly, a knowledgeable user is asked to read aloud the content of a *representative* part of a document, sentence by sentence, as if he/she were talking to a colleague (information flow 3 in figure 6.2) via the telephone (the 'Aunt Annie in Reykjavik way of verbalizing', see Nijssen and Halpin (1989:3); Nijssen and Twine (1989:971)). While the knowledgeable user is reading (information flow 6 in figure 6.2) the analyst is listening to the user (information flow 4 in figure 6.2), subsequently the analyst will shade the verbalized parts in the original example (information flows 2 and 7 in figure 6.2) and add *every* new sentence onto the output document (information flow 5 in figure 6.2).

After all the *representative* information is read aloud, it is checked with the user that each sentence on the output document can be traced back to a shaded pattern on the input document by reconstructing (the representative part of) the original example. If it is not possible to reconstruct the original example then these steps have to be performed again until the reconstruction of the original example(s) is possible.

Initial check on naming conventions. In the *verbalization* transformation, the user is asked to verbalize the content of a *real-life example*. Whilst verbalizing, the user will reference the relevant *concepts* and *entities* in his/her UoD by using names. In the *NLM* methodology the concept of *naming convention* is crucial for bridging the gap between the *business domain* (or *knowledge domain*) and the *formal structure* with respect to the requirements specification. In the *NLM* requirements determination method we will exactly determine what name classes are of interest. We will enforce the domain users to select those 'candidate' name class(es) for the naming convention for a specific intention within the application subject area that can be considered a reference type.

Definition 6.2. A naming convention is called a *reference type* for a selected portion of the real world if every element in the selected portion of the real world can be referenced by exactly one name from the name class used in the naming convention and that one name from such a name class references at most one element in that selected portion of the real world.

Let IS be an naming convention

T: elements or concepts in a selected portion of the real world

N: All names from a given name class

$$IS = \{(t, a) \mid t \in T \wedge a \in N\}$$

$$\left(\forall_{(c,t),(d,v) \in IS} [c = d \Leftrightarrow t = v] \right) \Leftrightarrow (IS \text{ is a reference type})$$

The analyst must check whether **all** names that are uttered by the user for referencing elements in that selected portion of the real world belong to a name class that is of the reference type. If this is **not** the case then a different name class must be selected. Finding an appropriate naming convention for a given intention in

cooperation with a user can be considered *a semantic bridge* that is embodied in the choice of the proper name class by the user for a specific *intention*.

Example 6.1:

Consider the ‘real-life’ example in figure 6.3.

| ABC COMPANY INVOICE | | | | |
|---------------------|-------------|----------|----------|-----------------|
| | | | 345 | Client: 123145 |
| Item | Description | price | quantity | subtotal |
| Ab102 | Hose | \$ 12,-- | 2 | \$ 24,-- |
| Cd879 | Pipe | \$210,50 | 1 | <u>\$210,50</u> |
| Invoice total: | | | | \$ 234,50 |

Fig. 6.3 ‘real-life’ ABC invoice document (example 6.1).

The ABC company creates an invoice for each client order. A *client* is identified by a *client code* among the union of clients of the ABC company. An *item* is identified by an *item code* among the union of items that are contained in the assortment of the ABC company. A *description* and a *unit price* exist for each item. An *amount (of money)* is identified by a *decimal number*.

Furthermore, it is explicitly recorded how many units of a specific item are ordered by a *client*. Such a product *quantity* is identified by a *natural number*. A subtotal is derived for each item in the invoice by multiplying the (unit) price for that item by the quantity that is ordered. A specific *invoice* is identified by a combination of a *client* and a *rank number*.

Finally, the total invoice amount is given on the example. The analyst has to check that every verbalized sentence is self-contained. For example: The two sentences: *There exists an invoice 345 for client 123145* and *on that invoice the total amount is \$ 234,50* must be replaced by the following sentence: *The invoice 345 for client 123145 totals \$ 234,50*. Furthermore, if the user has verbalized the following sentence: *The item hose has a unit price of \$ 12,--*, the analyst should ask the user following questions: *Does the name "Hose" refer to exactly one item among the union of items in ABC's assortment ?* and *Does the name \$12,--refer to exactly one price among the union of prices ?* It should be noted that in many cases the initial verbalization is very *compact*. In these cases a *formal check* on the naming convention will take place during the *classification and qualification* transformation.

Definition 6.3. *Verbalization* is the process of transforming *user examples* into *natural language sentences*.

We will now give the algorithm for the transformation *verbalization*.

Algorithm 1. Verbalization {UoD_i is the universe of discourse that contains 1 or more 'real-life' user examples. G is the group of users of the 'real life' examples in UoD_i}

```

BEGIN VERBALIZATION (UoDi, G)
WHILE still significant parts of user examples are not shaded
DO   let knowledgeable user (g∈G) verbalize the next unshaded part from the significant32 part of the real-life example in the UoD33.
      The analyst will shade this part on the real-life example and he/she will add the verbalized sentences on the document verbalized sentences.
ENDWHILE
Replace dependent sentences by self-contained sentences.
{Reconstruction check}
Let the analyst recreate the original example documents by translating the verbalized sentences document onto the corresponding parts on the original document.
IF the recreated document is identical with the shaded part of the verbalized document
THEN {no information loss has occurred}
ELSE {information loss has occurred,
      VERBALIZATION(UoDi, G)
      {Have the user verbalize the example again, thereby using a different naming convention and/or verbalizations that refer to bigger parts on the example document}
ENDIF
END

```

Algorithm 1, basically specifies how (a) 'real-life' example document(s) can be transformed into a verbalization and how the quality check for this transformation is implemented (the reconstruction check).

The document *verbalized sentences* exclusively contains natural language sentences. (A part of) the result of the verbalization transformation applied on the invoice example in figure 6.3 is given in figure 6.4.

³² A significant part of a Universe of Discourse in this stage of the way of working of the NLM requirements determination method should be considered a set of example 'instances' that contain all possible variation in the sentence groups.

³³ The **bold fonded** parts of this and the other algorithms in this chapter denote that this information must be supplied by the user and therefore constitute (parts of a) semantic bridge(s).

The item Ab102 has the description hose and a unit price of \$ 12,--
 The subtotal for item Ab102 on the invoice 345 by client 123145 is \$ 24,--
 The item Cd879 has the description pipe and a unit price of \$ 210,50
 The subtotal for item Cd879 on the invoice 345 by client 123145 is \$ 210,50
 The item Ab102 is ordered in the quantity of 2 on the invoice 345 by client 123145
 The item Cd879 is ordered in the quantity of 1 on the invoice 345 by client 123145
 The invoice 345 of client 123145 has a total of \$ 234,50

Fig. 6.4 Result of verbalization transformation of example 6.1.

The *verbalization* transformation is essential for the implementation of the modeling foundation of *NLM* (the natural language axiom in section 5.3). The *verbalization* transformation is the *unification* transformation that uses *all* appearances of *declarative verbalizable* information as an input and will converge it into declarative natural language sentences as an output.

6.2.2 The grouping transformation

The transformation *grouping* is the process of sorting sentences from the document *verbalized sentences* according to what they have in common into a number of (sentence) groups. The document verbalized sentences should contain a *significant* set of sentences. If a resulting sentence group only has *one* sentence instance on the significant part of the example document the user should verbalize a second sentence of the same type (eventually from a different ‘positive’³⁴ example for the same UoD) until all possible variability is reflected within the sentence groups.

Definition 6.4. *Grouping* is the process that divides *verbalized sentences* into groups of the same type.

The result of grouping is a partition of a set of sentences.

Let B be a partition of A

$B = \{B_i | B_i \subset A\}$ is a partition if $B_i \not\subset \emptyset$

and $B_i \cap B_j = \emptyset$ and $\cup B_i = A$

³⁴ A positive example is an example accepted by the user, otherwise it is negative (Tseng and Mannino, 1989:55)

Grouping is the mapping of N sentences into M ($\leq N$) groups of sentences by some *similarity criterion*. The grouping transformation will lead to a partition of the sentence groups (out of the 2^N that are theoretically possible). We note that in principle a set of N sentences can be grouped in different ways. The resulting information models for these groupings are (semantically) equivalent, in the sense of *entity type - fact type* conversions (Nijssen and Halpin, 1989: 222-223). In general the algorithms that are used in modeling processes have an impact on the outcome of the modeling process (Kaufman and Rousseuw, 1990:37). The *grouping* algorithm can lead to different (semantic equivalent) outcomes when applied in cooperation with two different users that have the same knowledge of the same UoD. However, in order, to be able to create ‘semantic equivalent’ models, the existence of some population state constraints must be enforced. To cater for this semantic equivalence transformation that is induced by the grouping algorithm, the *lexical* constraint type must be defined (Nijssen and Halpin, 1989:159).

Example 6.3:

Verbalized sentences:

The ABC company had a turnover for item ab 102 of \$ 345 in week 34 of year 1997.
The ABC company had a profit for item ab 102 of \$ 45 in week 34 of year 1997.
The ABC company had a turnover for item ab 103 of \$ 745 in week 35 of year 1996.
The ABC company had a profit for the item ab 103 of \$ 75 in week 35 of year 1996.

Result of transformation grouping by user 1:

Group 1:

The ABC company had a turnover for item ab 102 of \$ 345 in week 34 of year 1997.
The ABC company had a turnover for item ab 103 of \$ 745 in week 35 of year 1996.

Group 2:

The ABC company had a profit for the item ab 102 of \$ 45 in week 34 of year 1997.
The ABC company had a profit for the item ab 103 of \$ 75 in week 35 of year 1996.

Result of transformation grouping by user 2:

Group 1:

The ABC company had a turnover for item ab 102 of \$ 345 in week 34 of year 1997.
The ABC company had a profit for the item ab 102 of \$ 45 in week 34 of year 1997.
The ABC company had a turnover for item ab 103 of \$ 745 in week 35 of year 1996.
The ABC company had a profit for the item ab 103 of \$ 75 in week 35 of year 1996.

We will now give the *grouping* transformation algorithm for a given UoD and user group G .

Algorithm 2. Grouping {VS is the document verbalized sentences which contains the results of the verbalization transformation applied on UoD_i }

```

BEGIN GROUPING( $UoD_i, G, VS$ )
Divide the sentences into groups of the same type (SG)
WHILE still groups in SG
DO    take next sentence group { $gr \in SG$ }
      IF    the number of Sentences in  $gr = 1$  { $s_1$ }
      THEN  let knowledgeable user ( $g \in G$ ) verbalize a second piece of information belonging to that sentence group from a second example document instance of the same type. This second piece of information together with the first sentence from this sentence group should contain all variability within the sentence group. The analyst will shade this part on the real-life example and he/she will add the verbalized sentences on the document verbalized sentences.
      IF such a second piece of information can not be found
      THEN  Add the verbalized sentence  $s_1$  to the ACR.
            Remove this sentence group from the document VS.
      ENDIF
      ELSE  assign a unique code to the sentence group35
      ENDIF
ENDWHILE
{completeness check} Each sentence in the verbalized sentences document should belong to a sentence group in the grouped sentences document or must be contained in the ACR.
END

```

In figure 6.5 the result of the grouping transformation for example 6.1 is given.

³⁵ Preferably for each individual analyst/user interaction, a set of sentences group codes must be reserved. Those codes must not be overlapping among the different analyst/user interactions that are carried out within one requirements determination project

Sentence group Sg1

The item Ab102 has the description hose and a unit price of \$ 12,--
The item Cd879 has the description pipe and a unit price of \$ 210,50

Sentence group Sg5

The subtotal for item Ab102 on the invoice 345 by client 123145 is \$ 24,--
The subtotal for item Cd879 on the invoice 345 by client 123145 is \$ 210,50

Sentence group Sg6

The item Ab102 is ordered in the quantity of 2 on the invoice 345 by client 123145
The item Cd879 is ordered in the quantity of 1 on the invoice 345 by client 123145

Sentence group Sg7

The invoice 345 of client 123145 has a total of \$ 234,50
The invoice 346 of client 123567 has a total of \$ 4,20

Fig. 6.5 Result of grouping transformation of example 6.1.

6.2.3 The classification and qualification transformation

Colleagues know the background of their co-workers and the language they use refers to a *shared world*. An example of such an implicit communication between colleagues is: *35467 is due for 23-97* (example 6.2). Suppose another sentence is communicated between two colleagues in this subject area: *35469 is due for 25-97*. We can now group these sentences as follows:

Group 1: *35467 is due for 23-97*
 35469 is due for 25-97

For the colleagues in the *logistics and supply* department of a company these sentences have a definite meaning but it can be very hard for an *outside* person (who works for the same company) to find out what the communication is about. Because a requirements specification should reflect the ‘organizational’ semantics of the *complete* application subject area we have to ask the user that has verbalized the sentences to inject ‘additional semantics’ in such a way that the resulting sentences can be understood by a colleague from another department within the same company. The result of this ‘semantic injection’ would yield sentences like:

*The supply order with order number 35467 is due for week 23 of the year 1997 and
The supply order with order number 35469 is due for week 25 of the year 1997.*

This is the reason that we need a transformation that transforms of the sentence groups on the document *grouped sentences* into a *semantic rich format* that specifies exactly what the communication is about (e.g. what concepts are involved and how these concepts are defined) and what naming conventions are used to identify instances of these concepts. We will call this transformation the *classification and qualification* transformation. Firstly the variable and fixed parts for each sentence group will be determined (the *classification* sub transformation). Secondly the intention of the concepts that play the roles will be determined and a naming convention will be established, e.g. it will be made explicit to what *name class* the individual names, which reference those concepts, belong (the *qualification* sub transformation).

The classification sub-transformation. In the classification transformation we will investigate *each* sentence group at a time. We will depict those parts of a sentence group from example 6.1 that are fixed and the parts that are variable :

Example 6.1(ctd.): Sentence Group1=: { The item Ab102 has a unit price of \$ 12,--,
The item Cd879 has a unit price of \$ 210,50}
Variable parts sentence Group1=: { The item Ab102 has a unit price
of \$ 12,--, The item Cd879 has a
unit price of \$ 210,50}
Fixed parts sentence Group1=: { The item Ab102 has a unit price of
\$ 12,--, The item Cd879 has a
unit price of \$ 210,50}

We will call the names in the sentences that are variable within a significant set of sentences of the sentence group: *individual names*. The remaining positions in the sentence groups contain text parts that are fixed for every sentence (instance) of the sentence group. In the remainder of this paper we will call the *variable* parts in a sentence group *roles* and we will call the *fixed* parts in a sentence group *verb parts* in the accompanying fact type template. We will now be able to specify a sentence group by replacing the sentence positions for the roles by a *role name*. This leads to the following sentence group template for sentence group 1: *The item <r1>has the description <R2>and a unit price of <R2b>*. An equivalent graphical notation for a sentence group for this example is shown in figure 6.6 (see also chapter 5).

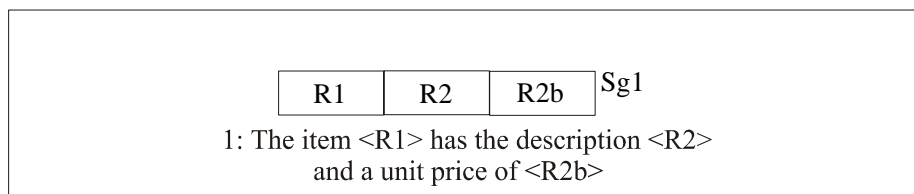


Fig. 6.6 Graphical notation sentence group for (a part of) example 6.1.

In the graphical notation we will denote each role by a box that contains the role name. The sentence group template(s) will be placed under the ‘role-boxes’ and each template will contain at least one reference to *each* ‘role box’ (see chapter 5). We must remark that the focus in this thesis is not on the notational convention (chapter 1), we will use the box notation to facilitate the expression of a number of constraint types that will be later on in this chapter, however, any suitable notational legend could be used for this purpose.

The qualification sub-transformation. Now we have classified the sentence group elements into *variable* and *fixed* parts we can start deriving the additional semantics for the Universe of Discourse by establishing additional semantic bridges with the user. Firstly, the *type* of concept or thing (defined as *intention*) and its *definition* to which the *individual names* in a role of the sentence group refer, will be recorded. For every *role* in a sentence group we will determine its intention from a sample extension of that concept by posing the *what* question

*To what **type of thing or concept** refer the individual names in this role ?*

For each *intention* that is distinguished by the user in the answers to the *what* question, a definition should be recorded in the *application concept repository (ACR)*. Every definition of an intention should be expressed in terms of general known concepts and/or intentions that are already defined in the *application concept repository* (see section 5.5). Dependent upon the way in which the initial user verbalization has taken place, a specific intention might *not* be contained in the sentence group (e.g. the example in the grouping transformation in which an ‘inter-colleague’ level of verbalization exists).

In those cases we will add the intention to the sentence group by putting the intention in front of the role names in the sentence group template.

Example 6.2(ctd.):

Sentence group: *The item ab102 has a unit price of the amount \$ 12,--
The item cd879 has a unit price of the amount \$ 210,50*

Sentence group template: *The item <R1> has a unit price of the amount <R2>*

*Intention(ab102, cd879):= Item;
Intention (\$12,--, \$ 210,50):= Amount*

Asking the user to define the intention in terms of other ‘known’ intentions leads to the *application concept repository (ACR)* in figure 6.6. We remark that the ACR should be based upon the ontology of the integrated application subject area, and it is therefore, defined on a business level. Furthermore, the *specialization* and *generalization* relationships that exist between intentions within a UoD must be incorporated into the ACR. In very large requirements determination projects we recommend to add the name of the department (or organizational unit) and the names of the knowledgeable users that have defined the concepts to the ACR, in case of definition and interpretation conflicts that might in the course of the requirements determination project.

| Intention | Synonym | definition |
|------------------|----------------|---|
| Item | | <i>a product that is contained in ABC's assortment</i> |
| Client | Customer | <i>a person that has ordered or that is about to order an item at ABC</i> |
| Invoice | | <i>a document that specifies the payments for an order</i> |
| Amount | | <i>a specific quantity of money in dollars</i> |
| Product quantity | | <i>a specific quantity of items</i> |

Fig. 6.7 Initial application concept repository for (a part of) example 6.1.

Secondly, we will *formally* establish the naming convention for the intentions that have been defined in the UoD in this sub-transformation. We will ask the user the *how* question one time for *every* intention that has been distinguished. In some situations this question serves as a quality check on the initial naming convention that has been performed during the *verbalization* transformation in which the user initially has verbalized the ‘real-life’ example.

How ? (or by what names) are instances of a given intention in the application repository within this UoD identified ?

This question determines if a *name class* can be specified that configures a *reference type naming convention* for the selected portion of the real world that consists of the *union* of instances of the intention.

In the example of *The item <R1> has a unit price of the amount <R2>* sentence group the following two *how* questions can be posed:

Question 1: *How ? (or by what names) are instances of an item in the ABC invoice UoD identified ?*

Question 2: *How ? (or by what names) are instances of an amount in the ABC invoice UoD identified ?*

The answer to question 1 is that an *item* is identified by a name from the *item code* name class and the answer to question 2 is that an *amount* is identified by a name from the *decimal number* name class. The answer to these ‘how’ questions is another ‘semantic injection’ to the existing sentence groups. We will model these additional semantics as a sentence group that is ‘connected’ to the appropriate intention. In this example the naming convention sentence group for the intention *Item* is :*<R3> is a name from the **item code** name class that can be used to identify an item within the union of items in ABC's assortment* and for the intention *Amount* it is: *<R7> is a name from the **decimal number** name class that can be used to identify a specific amount of money in dollars within the union of money amounts*. It should be noted that the selected portion of the real world in which the names from the name class can be considered to be of the reference type should be explicitly mentioned in the naming convention fact type template. In figure 6.8 we have illustrated how all extensions of the role *R1* that are played by the intention *item* at any time must be a subset of the instances of the name class *item code* (subset constraint C1) and how all extensions of role *R2* played by the intention *amount* at any time must be a subset of the instances of

the name class *decimal number* (subset constraint C6) (see section 5.8 for the definition of a subset constraint).

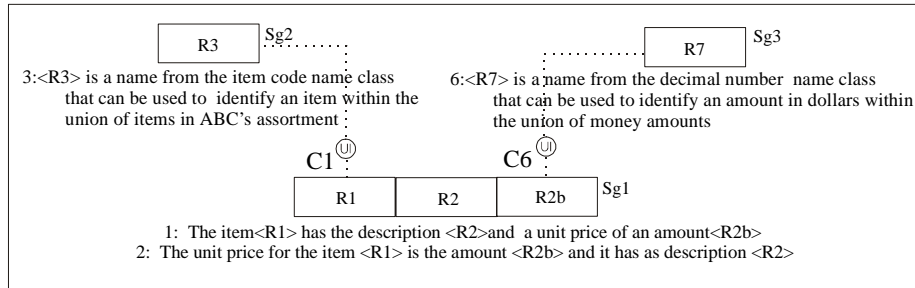


Fig. 6.8 Application of naming convention fact types in NLM.

We have now shown that in principle we can pose *two* questions for *every* role in a sentence group. The answer to the *what* question will lead to the identification of a specific application *intention* for that *role*. The answer to the *how* question leads to the detection of a specific *name class* and is encoded as *a naming convention* sentence group plus the appropriate referencing constraint. Until now we have assumed that the name of the name class that can be used to identify instances of the intention is different than the name of the intention. However, it is possible that an intention of an individual name coincides with the name class. Consider the following verbalized sentences from example 6.1 (that constitute one sentence group):

The invoice 345 for client 123145 totals \$ 234,50
The invoice 345 for client 578995 totals \$ 125,00
The invoice 348 for client 578995 totals \$ 25,75

The latter sentence group has three variable parts (denoted by an underscore):

The invoice 345 for client 123145 totals \$ 234.50
The invoice 345 for client 578995 totals \$ 125.00
The invoice 348 for client 578995 totals \$ 25.75

The intentions for the respective role extensions are the following:

Intention(345,348):= *number*
 Intention(123145, 578995):= *client*
 Intention (\$ 234.50, \$125.00, \$ 25.75):= *amount in dollars*

If we would now ask the *how* question for every intention that was discovered, we yield:

Name class (*number*):=*number*
 Name class (*client*):=*person name*
 Name class (*amount in dollars*):=*number*

In this example we see that for the ‘first’ role in the sentence group the name class that can be used to identify instances of the intention is *identical* to the intention itself. This means that there does **not** exist an intention for this role other than the name class itself. In this case we do not define a naming convention fact type. Instead we will record the name class in the position of the intention name in the sentence group. We can now conclude that for each *role* in a sentence group we must record the *intention* of the individual names that play that role in the sentence group and the *naming convention sentence group* for that intention **or** we must record the *name class* to which the individual names that play that role in the sentence group belong.

Compound referencing schemes. In some subject areas users have introduced naming conventions that are *complex*, e.g. that consist of names that have an *internal* structure. The qualification of every role in such a sentence group, therefore, will **not** necessarily lead to the detection of all *intentions* that have extensions that consists of value combination of two or more roles in a sentence group. Thus, in some sentence groups the *intentions* are **not** linked to exactly *one* role. We will illustrate this once again in example 6.1.

Example 6.1(ctd.):

Sentence group: *The invoice 345 for client 123145 totals \$ 234,50*
 The invoice 345 for client 578995 totals \$ 125,00
 The invoice 348 for client 578995 totals \$ 25,75

Sentence group template: *The invoice <R4> for client <R5> totals <R6>*

Qualified Sentence group template: *The invoice having rank number <R4> for client <R5> totals the amount in dollars <R6>*

We have discovered a fourth intention or name class in this qualified sentence group: *invoice*. If we once again ask the question : *by what naming convention is an instance of an invoice depicted ?*, it will turn out that this is a complex identification structure, consisting of combinations of individual names in roles <R4> and <R5>.

The invoice [identified by rank number <R4> for client <R5>] totals the amount in dollars of <R6>

This means that in those sentence groups that contain at least one intention with a compound referencing scheme we will have to incorporate the naming convention sentence group, in the communicated sentence group itself. In case such a ‘compound’ intention (or aggregation (Smith and Smith, 1977)) is not contained in the initial verbalization by the user we will have to trace the existence of such a ‘compound’ intention by systematically confronting the user with all possible role combinations (of a sentence group) and ask the user whether such a role combination can be considered as a referencing scheme for a potential ‘compound’ intention. The classification and qualification transformation should enforce the analyst and the user(s) to specify all intentions that have a compound referencing scheme. Consequently, the naming convention template for these ‘compound’ intentions will be incorporated into the ‘flat’ sentence group in which they appear. We will now give the algorithm for the classification and qualification transformation for a specific sentence group.

Algorithm 3. Classification and Qualification {SG is the set sentence groups on the document grouped sentences that is the result of the transformation grouping}

```

BEGIN CLASSIFICATION and QUALIFICATION(UoDj, G, SG)
WHILE Still sentence groups
DO   Take next sentence group {sg∈SG}
     List all sentences in the sentence group sg.
     Mark the common parts throughout all the sentence
     instances in the group sg.
     Insert a role code36 for
     every variable part.
     WHILE still roles in the sentence group sg
         Take next role in sg {rg}
     DO Determine the intention that plays the role rg
        in the sentence group sg by posing the what
        question (answer: Ix).
        Pose the How question for Ix: (answer Nx)
        IF Nx=Ix THEN Ix is a name class.
            Let the user record the name
            class definition in the ACR
        ELSE Ix is an intention.
            Let the user define the intention
            and record this definition in the
            ACR. Determine the naming
            convention fact type37 that
            connects name class Nx to
            intention Ix. Add a subset
            constraint between the role and
            the role of the naming convention
            fact type

        ENDIF
    ENDWHILE
    IF arity of sg >=2
    THEN Check on compound intentions with the users38

```

³⁶ Such a role code must be a *reference type* naming convention within a requirements determination project or a group of projects.

³⁷ In the template of the naming convention fact type the 'selected portion of the real world' in which the naming convention is of the *reference type* should be included.

³⁸ Consider a fact type consisting of roles R_1, \dots, R_N . For each combination of j ($2 \leq j \leq N$) roles try to determine an intention, that has a **compound referencing scheme**. As soon as all roles are contained in a (compound) referencing scheme, we can stop this algorithm. We will now replace this compound referencing scheme by creating one role for each intention having such a complex identification structure. The resulting sentence group will be called the **compound** sentence group as opposed to the **flat** sentence group. For example, the flat sentence group Sg5 contains 4 roles, the compound sentence group Sg5 will contain 3 roles (in this cases role R16 and R17 are joined into one compound role). See for an earlier discussion on this issue Biller (1979:280-282).

6.3 THE ATOMIZATION PROCEDURE

In NLM we use ‘real-life’ examples of communication or ‘knowledgeable user’ views as a starting point for the requirements determination sub process that involves the requirements elicitation on an individual user (or eventually user group) level. In the *ANSI/SPARC three-schema architecture* (Tsichritzis and Klug, 1978) different of these *external user schemata* on the same *conceptual schema* can be defined. A conceptual schema according to the *ANSI/SPARC three-schema architecture* should enable all workers in the enterprise to access all *corporate* facts. The definitions in a requirements specification, therefore, should not favour one external schema over another. Furthermore, a requirements specification must contain all domain semantics that are needed to eventually implement an automated information system to support the business management and operations. For example, a relational data base implementation, requires that the tables in the logical database design are in 5th Normal Form (5NF). If the application requirements specification does not explicitly contain the semantics with respect to functional dependencies, this would imply that during the database normalization process, a database designer once again has to consult the users in the application domain to help him/her in determining the functional dependencies. For these reasons, the sentence groups in the conceptual schema or requirements specification in this architecture need to be *atomic elements* of which the compounds in the external schemata are created and that contain all required semantics for the implementers.

Definition 6.5. An *elementary* or *atomic sentence group* is a sentence group of which the sentence instances can not be split up into two or more sentences without losing information and can not be contained in another atomic sentence group referring to the same Universe of Discourse. An elementary sentence group is also called semantic irreducible (Falkenberg, 1976).

Let $\{FFT_{ij}\}$ be the set of sentence group templates defined on UoD_k and a user group G . Where FFT_{ij} refers to sentence group template i for sentence group j . Let the sentence α be an instance of a sentence group template FFT_{im} for sentence group $FT_M (\subset \{FFT_{ij}\})$ referring to the universe of discourse UoD_k . The sentence α is atomic \Leftrightarrow

$$\begin{aligned} & \quad \quad \quad m(G)^{39} \\ (\neg \exists \beta_1, \dots, \beta_N \blacklozenge^{40} \{FFT_{ij}\} \setminus FFT_{im} \ [\alpha \Rightarrow \beta_1, \dots, \beta_N]) & \quad \vee \\ & \quad \quad \quad m(G) \quad \quad \quad m(G) \\ (\exists \beta_1, \dots, \beta_N \blacklozenge \{FFT_{ij}\} \setminus FFT_m \ [\alpha \Rightarrow \beta_1, \dots, \beta_N \wedge \beta_1, \dots, \beta_N \neg \Rightarrow \alpha]) & \end{aligned}$$

³⁹

$m(G)$

where $\alpha \Rightarrow \beta_1, \dots, \beta_N$ is defined as : The existence of sentences β_1, \dots, β_N is implied by sentence α according to user group G in the given *universe of discourse*. See also Biller (1979).

⁴⁰ Where $[a \blacklozenge A]$ is defined as: a is an instance of A .

We will now give the algorithm for the *atomization* transformation for a given UoD and a user group G.

Algorithm 4. Atomization (Sg) {Sg is the set of sentence groups that results from the application of algorithm 3: classification and qualification}

BEGIN ATOMIZATION({SG_i},G)

Consider exclusively the compound sentence group formats as defined in algorithm 3 for every sentence group

WHILE not last sentence group from SG AND
arity(sg) > 2

DO Take a sentence group template (sgt) from sentence group sg (\in SG)

Take following sentence as a first sentence instance of this sentence group template {Let the arity of sgt= N): a₁ b₁ .. N₁

{ Comment: we define the set of different role combinations consisting of j roles within the sentence group template sgt as follows:

$RCOMBSI(j) := \{c_{j1}, \dots, c_{jk}, \dots, c_{j\binom{N}{j}}\}$

j:=1

WHILE j \neq N-2

DO k:=1

WHILE k $\neq \binom{N}{j}$

DO Create a new sentence instance of the sentence group template sgt: si2
si2 should have different names in all roles except the roles in c_{jk}.

Check with the user in user group G whether a combination of sentences si1 and si2 is allowed

IF such a combination is not allowed
THEN the sentence group sg of sentence group template sgt is not atomic. **Ask the user to split the sentence group sg into 2 or more fact types sg₁, sg₂ sg_p**⁴¹ such that the

⁴¹ We assume that the extensions of the name classes that will be used for referring to atomic fact types will be disjunct in the different user/analyst interactions. A naming convention for the fact types would be the following: For each sentence group that is atomic keep the same name. If

```

        shading of instances of these new
        sentence groups on the original
        examples used in the verbalization are
        equal to the shaded part for the
        corresponding instances of the non-
        atomic sentence group  $sg_i$ 
         $\{SG\} := \{SG\}/sg \cup \{sg_1, \dots, sg_p\}$ 
    ELSE  $k := k + 1$ 
         $j := j + 1$ 
    ENDIF
     $J := N - 2$ 
ENDWHILE
ENDWHILE
take next sentence group from SG
ENDWHILE
{reference check}
Check that each sentence group on the (input) document
'classified and qualified sentence group' refers to at
least one fact type on the output document 'atomized
sentence groups'.
END

```

We remark that even in the case in which there exists *exactly one* external view on the application information base we still need to atomize the sentence groups that are contained in the document *sentence group templates* simply because an *elementary* fact should be *stored at most one* time in the *application information base* in order to avoid update anomalies. The atomization process results in a conceptual schema or requirements specification for which algorithms can be defined that group these atomic fact types into relation types in an optimal normal form (ONF) (Leung and Nijssen, 1987). The mapping in (Nijssen and Halpin, 1989:254-260) typically generates tables in fifth normal form (5NF). For an elaboration on atomization we refer to Kent (1978) and Nijssen and Halpin (1989). We note that we will use the term *fact type* to refer to those sentence groups that are *atomic* within the UoD. In figure 6.10 we have given the result of the application of this transformation for example 6.1.

a sentence group has to be split up use the original name plus one or more new names from the designated name class to denote these atomized sentence groups or fact types.

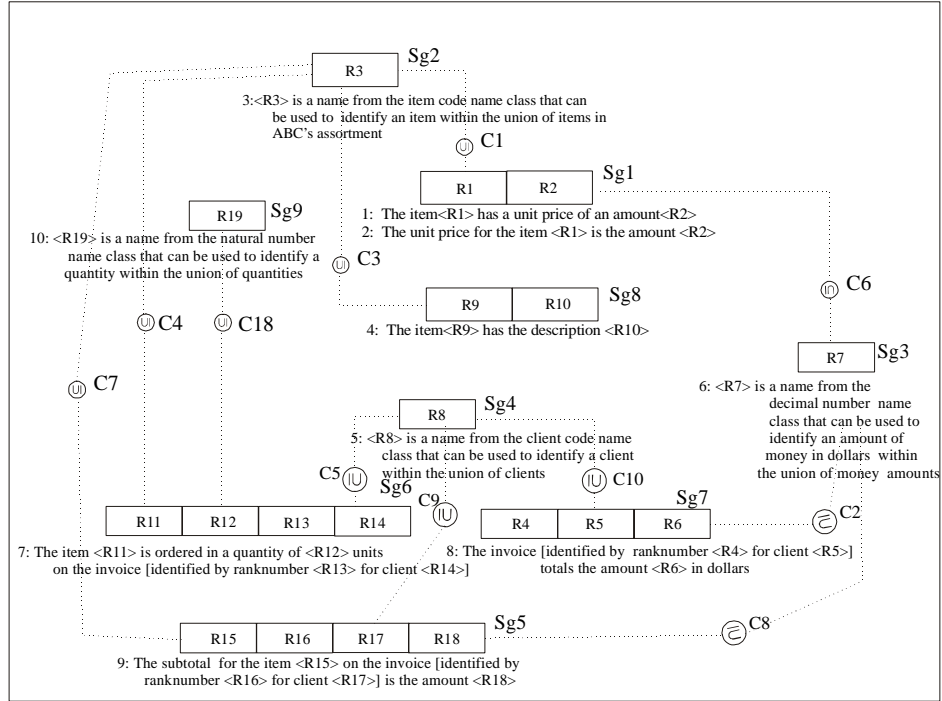


Fig. 6.10 Result of atomization transformation of example 6.1.

We note from the example in figure 6.10 that if compared with the results of the classification and qualification transformation in figure 6.9 only sentence group *Sg1* has been split up into the new and atomized fact types *Sg1* and *Sg8*.

6.4 THE PROCEDURE FOR INTEGRATING BIMs IN NLM

We will now give a number of equivalence transformations that are required in order to be able to integrate information models that are created in a number of different analyst/user interactions in different requirements determination sub processes. This is generally known as *schema integration* (Beynon-Davis et al., 1997). Many schema integration approaches are based on the (E)E-R approach for conceptual modeling (Beynon-Davis et al., 1997; Johannesson, 1997; Kwan and Fong, 1999; Lim and Chiang, 2000; McBrien and Poulosvassilis, 1998; Navathe et al., 1984). An approach that is based upon a (binary) fact oriented approach can be found in Shoval and Zohn (1991). In Johannesson (1997) the schema integration process is divided into *schema comparison*, *schema conforming* and *schema merging*. In Lim and Chiang (2000) schema-level conflicts are classified into (*relationship*) *naming*, *structural*, *identifier*, *cardinality* and *domain* conflicts. Kwan and Fong (1999) in addition, propose to

resolve *conflicts on synonyms and homonyms*, merge entities by generalization and merge entities by subtype relationships.

6.4.1 Conflicts on naming, synonyms and homonyms

In NLM schemas we can encounter naming conflicts on two levels: names for *intentions* and names for the *verbs* in the fact types. To resolve the first conflict we need *primary naming convention postulation*. In principle it is possible that more than one reference type naming convention is used for a specific intention in the different sub-schemas. In an integrated UoD however it is essential to have *exactly one* primary naming convention for each intention.

Definition 6.6. Primary naming convention postulation is the process of selecting one naming convention for every intention for (a) given integrated UoD (s).

After this negotiation process (Meersman, 1994; Shoval and Zohn, 1991:237), the primary naming conventions for the intentions in the application subject⁴² area must be known together with their synonyms that can be incorporated into the conceptual information model, by means of *synonym fact types*, e.g. *The employee with employee ID <r1> has social security number <r2>*. Furthermore, for all intentions that are contained in the first UoD **and** the second UoD and that have identical naming convention fact types, it should be checked whether these naming conventions still can be considered having a *reference type naming convention* in the integrated UoD.

Example 6.3:

UoD 1: *affiliate A: Employee with employee id⁴³ <R1> lives on Address <R2>*

UoD 2: *affiliate B: Employee with employee id <R1> lives on Address <R2>*.

In example 6.3 the conceptualization into an enterprise wide requirements specification forces users in the integrated UoD to determine a *new* naming convention for *employees*. In such a case it is likely to create a compound naming convention in which the extension of the first role consists of names from the name class *employee ID* and the extension of the second role of the names from the name class *affiliate code* (e.g. A or B). An alternative naming convention that is of the reference type for the integrated UoD in example 6.3 is an *employee ID* that identifies employees within the integrated application subject area. The advantage of selecting the former naming convention is that the company can capitalize on the existing naming conventions.

⁴² We note that synonyms for intentions or concepts (Mirbel, 1997:184) are directly incorporated into the Application Concept Repository (ACR) (Shoval and Zohn, 1991:227).

⁴³ We assume that the instances of the name class *employee ID* are (potentially) overlapping within the union of UoD1 and UoD2: The name class *employee ID* is of the reference type within UoD1 and within UoD2, but it is **not** of the reference type within the integrated UoD.

Concerning the possibility of homonyms (Mirbel, 1997:184) for application intentions, provisions have been made in the application concept repository in which the same definition for different intentions are prohibited, but if they do occur, investigations can take place by involving the relevant user groups. With regard to the names in the verbs we remark that when we integrate two fact types from two sub-UoD's having different fact type templates in which the same intentions are involved (in the same roles) we need to determine whether they belong to the same group of fact type templates (for a given fact type) or whether they can be considered to represent two *different* fact types. In order to facilitate such a comparison we suggest that the main concepts that are contained in the verb-parts of the fact type templates should be defined and added to the Application Concept Repository (ACR). In such a way a common business ontology is preserved.

6.4.2 Specialization and generalization relationships

The second integration transformation is the *generalization/specialization* transformation. The reason for applying this transformation is that it can not be expected that all users have complete domain knowledge on the integrated enterprise subject area. For some user groups the intentions *person* and *student* can be considered synonyms because the extensions of both intentions will always contain the same instances. Whenever the subject area is extended (which is the case when UoD's are integrated) there will be a possibility that the extensions of some intentions in the integrated model are overlapping or are contained in one another. For example the following intentions *person*, *student*, *traveller*, *customer*, *dutch citizen* can refer to overlapping or inclusive classes of "physical persons" depending upon the scope of analysis (Kung, 1990). In case the extension of an intention is an inclusive class of the extension of another intention we will call the former a *subtype* of the latter. Furthermore, it is possible that the extensions of two intentions that always exclude each other or that partially overlap can be generalized into an 'overlapping' intention in an integrated UoD whose extension is defined as the *union* of the extensions of the intentions in the different UoD's. We will call such an intention in the integrated UoD a *super type*.

Definition 6.7. Generalization/specialization is the process of determining super types and subtypes for the intentions in the integrated basic information model⁴⁴.

⁴⁴ We thereby assume that specialization/generalizations that exist in a single user/analyst UoD will be modeled explicitly in the BIM by defining the appropriate subset constraints between the fact types in which the super/subclass intentions participate or between the appropriate 'existing-postulating' fact types (ter Hofstede et al., 1997:352). Furthermore, specializations and their definitions and/or generalizations need to be explicitly stored in the application concept repository (ACR).

Example 6.5:

UoD 1: *Tennis player <R1> lives on address <R2>*
UoD 2: *Employee <R3> lives on address <R4>*
Integrated UoD: *Person <R5> lives on address <R6>*
Tennis player <R7> is a person.
Employee <R8> is a person

*This transformation is called: **generalization***

In example 6.5 we have defined the intention *Person* as a super type of the intentions *Tennis player* and *Employee*. It should be noted that the generalization transformation can only be applied when there exists a naming convention for the derived *super type* that is of the *reference type* (Bollen, 2002) in the UoD. See also Navathe et al. (1984:84 case 1)

Example 6.6:

UoD 1: *Person <R1> lives on address <R2>*
UoD 2: *Student <R3> lives on address <R4>*
Integrated UoD: *Person <R5> lives on address <R6>*
Person <R7> is a student

*This transformation is called: **specialization***

We will define the intention *Student* as a subtype of the intention *Person* in the integrated UoD of example 6.6.

6.4.3 Identifier, cardinality and domain conflict

The integration of sub-schemas in NLM is defined on a basic information model level. This means that identifier (or key) conflicts and cardinality conflicts (Lim and Chiang, 2000:158) will not occur. It is assumed that in a ‘first-pass’ requirements analysis the population constraints will be derived directly in the integrated UoD⁴⁵. Furthermore, the attribute domain conflicts are not relevant in NLM because the only fact encoding construct is the fact type. Algorithm 5 specifies the integration algorithm for 2 basic information models. For the general case in which N basic information models need to be integrated we will apply the algorithm on the two first BIMs and subsequently re-apply algorithm 5 on the (cumulative) integrated model and the next BIM and so forth (see for the ‘integration-plan’-tree: Shoval and Zohn (1991:242-243))

⁴⁵ In section 6.9 where we will discuss the way of controlling in NLM we will elaborate on the precedence requirements and the specific users (or user groups) that need to be involved in those stages of the requirements determination process.

Algorithm 5. Integration of basic information models {BIM₁ and BIM₂ are two basic information models that refer to sub-UoD's of UoD, G₁ and G₂ are the respective user groups from these sub-UoD's}

```
BEGIN      Basic      information      model      integration
(BIM1,BIM2,G1,G2.)
```

```
Determine the set of intentions that are contained in
BIM1 and BIM2 {OVERLAPINT}
```

```
WHILE still intentions in OVERLAPINT
```

```
DO take next intention {int}
check the naming convention fact types
for int in BIM1 and BIM2.
```

```
IF naming conventions for int is not equal in BIM1
and BIM2
```

```
THEN IF the name class for intention int within
the sub-models BIM1 and BIM2 is not of the
reference type in the integrated UoD
```

```
THEN create a new naming convention for that
intention for the integrated UoD that is a
reference type in the integrated UoD.
```

```
ENDIF
```

```
ENDIF
```

```
ENDWHILE
```

```
WHILE still intentions left in the (integrated)ACR
```

```
DO take next intention {int}
IF int is a generalization of two or more
Different intentions in the ACR
```

```
THEN Create a generalization hierarchy by adding
subset constraints between the specialized
intentions in the ACR and the generalized
intention
```

```
ENDIF
```

```
ENDWHILE
```

```
WHILE still fact types in the integrated BIM in
which the same set of intentions play a role
{SIFT}
```

```
DO take next group of these fact types {sgi,
sgj,...∈SIFT}
```

```
WHILE still pairs [{sgk,sgl}∈SIFT |k>1]
```

```
DO Take next pair {sgk,sgl}
IF the semantics of a pair wise comparison
sgk, sgl in SIFT reveals no semantic
difference
```

```
THEN replace all but one fact type and
add the sentence group template of the
other to the sentence group templates
```

```

                                of this fact type
                                ENDIF
                                ENDWHILE
                                ENDWHILE
                                {ontological equivalence check}
                                Every fact type that can be expressed in each sub-model
                                should be expressed in the integrated model. Every fact
                                type that can be expressed in the integrated model
                                should be expressed in at least one sub model.
                                END

```

6.5 THE POPULATION STATE CONSTRAINT MODELING PROCEDURES

After a basic information model is created for a specific UoD, the analyst can elicit additional business rules from the domain user(s) by systematically confronting him/her (them) with new (combinations of) ‘real-life’ examples from the domain. The domain user(s) only needs to confirm or reject the possibility that such a (combination of) examples can exist. In this section we will give an illustration of such an algorithm for the *uniqueness constraints* and *set comparison constraints*

6.5.1 The derivation of uniqueness constraints

Uniqueness constraints will constrain the occurrence of two or more fact instances in which a subset of the roles have *identical* value combinations.

Lemma 6.1. For each elementary (or atomic) fact type f with arity N assuming a ‘one to one’ naming convention⁴⁶, one of the following rules apply⁴⁷:

- 1) No uniqueness constraint exists.
- 2) There is at least *one* uniqueness constraint defined on exactly $N-1$ roles of fact type f .

⁴⁶ We will use the **compound** fact types for the derivation of uniqueness constraints according to the definition given in algorithm 3.

⁴⁷ According to the definition in Tehrani and Nijssen (1985), every elementary fact type has at least one uniqueness constraint that involves *at least* $N-1$ roles. In NLM we do not consider a uniqueness constraint that spans N roles to be a ‘real’ uniqueness constraint, because an extension of a fact type is a ‘set’ of ordered instance value combinations and therefore such a constraint would always be implied.

It is assumed that the basic information model consists of *atomic* fact types. This assumption underlies algorithm 6. It is however, possible to, create an algorithm in which the possibility of uniqueness constraints that cover less than N-1 roles is considered. Such an extended algorithm can serve as a quality check on the outcome of (the atomization) algorithm 4.

Algorithm 6. Uniqueness constraint derivation

```

BEGIN  UNIQUENESS((I)BIM ,UoD ,G {(I)BIM is basic
information model that refers to an (integrated UoD)})
WHILE not last fact type of arity >1
DO48 take a random sentence instance from a complex fact
type template for this fact type from the example
UoD: (a1,..., aN): ft∈ (I)BIM
Take the first role from this fact type (m:=1)
WHILE not last role in fact type
DO Create an example sentence where the instance
of role m is altered. Determine whether the
combination of this sentence with the first
sentence is allowed
IF the existence of such a sentence is allowed
together with (a1,..., aN)
THEN add this sentence to the uniqueness
significant population
ELSE define a uniqueness constraint UC49 on
roles {1,...,N}\m of fact type ft
ENDIF
Go to the next role in fact type (m:=m+1)
ENDWHILE
Take next fact type
ENDWHILE
{N-1 law check}.Apply the N-1 law in Lemma 6.1 on each
fact type
END

```

After the uniqueness constraint derivation procedure has been applied on the BIM of our example the analyst can add the uniqueness constraints C11, C12, C13, C16, C17 to the application's basic information model (see figure 6.11).

⁴⁸ When the fact type is contained in more than one BIM then this confrontation with the real-life examples must be performed with every example document/user group combination that has verbalized sentence instances for this fact type (see also section 6.9 on the way of controlling). If this process leads to multiple uniqueness constraint configurations on the focal fact type then the least constraining set of uniqueness constraints will be chosen.

⁴⁹ We need to assign a unique name to every instantiated constraint in the requirements specification to distinguish this constraint among the union of constraints in the integrated requirements specification for the project.

The value combinations in the sets of extensions of each combination are equal:
equality

The value combinations in the sets of extensions of each combination are exclusive:
exclusion

The value combinations in the sets of the extensions of one combination are always contained in the set of extensions of the other: **subset**

We will now provide the algorithm that can be used for the instantiation of set comparison constraints for a given *basic information model* and *universe of discourse*. The outcome of the algorithm will always lead to one of the following outcomes in terms of the existence of a proposed set-comparison constraint: such a constraint does *not* exist, such a constraint *exists* as a *subset* constraints, such a constraint *exists* as an *equality* constraint or such a constraint *exists* as an *exclusion* constraint.

Algorithm 7. set comparison constraint derivation.

```
BEGIN SETCOMPARISON(IM ,UoD , G1 ..Gk )
Let ROLCOMB be the set of all possible role Combi- nations51
that refer to the same set of intentions in IM.
WHILE still role combinations left
DO take next role combination ∈ ROLCOMB
  Let (R1, ....RN) and (RN+1, ....R2n)52 be the
  role(s)combination on which the set comparison should
  be performed.
  Let (a1,... aM) be a sentence instances of the fact
  type (FT1) that contains roles (R1, ....RN) (M≥N)
  Let bN+1,... b2N+L and gN+1,... g2N+L be sentence
  instances of the fact type (FT2) that contains roles
  (RN+1, ....R2n) (L≥0).
Let IM:={FT1,FT2}. Create three user examples that re-
flect the following extensions of fact type FT1 and FT2:
EXT1(IM): { (a1,... aM) }
EXT2(IM): { (a1,... aM), ( bN+1,... b2N+L) | a1=bN+1 .. aN=b2N }
EXT3(IM): { (a1,... aM), ( bN+1,... b2N+L), (gN+1,... g2N+L)
              | a1=bN+1..... aN= b2N }
```

⁵¹ The roles of the naming convention fact types are excluded from this analysis

⁵² {R₁,R_N} ≠ {R_{N+1},R_{2n}}

Let the user⁵³ determine which of these extensions refer to an allowed population state for the universe of discourse by showing (sets of) real-life examples that match these three extensions one at a time.

```

IF (∃ Popstate1(UoD) [Popstate1(UoD)= EXT1(IM)] ∧
    ∃ Popstate2(UoD) [Popstate2(UoD)= EXT2(IM)] ∧
    ¬∃ Popstate3(UoD) [Popstate3(UoD)= EXT3(IM)] )
THEN
  (There is a subset constraint54 defined from role
  combination (RN+1, .R2n) to role combination (R1, .R..N))
ELSE IF (¬∃ Popstate1(UoD) [Popstate1(UoD)= EXT1(IM)] ∧
    ∃ Popstate2(UoD) [Popstate2(UoD)= EXT2(IM)] ∧
    ∃ Popstate3(UoD) [Popstate3(UoD)= EXT3(IM)] )
THEN
  (There is a subset constraint defined from role
  combination (R1, ....RN) to role combination
  (RN+1, ....R2n) )
ELSE IF
  (¬∃ Popstate1(UoD) [Popstate1(UoD)= EXT1(IM)] ∧
  ∃ Popstate2(UoD) [Popstate2(UoD)= EXT2(IM)] ∧
  ¬∃ Popstate3(UoD) [Popstate3(UoD)= EXT3(IM)] )
THEN
  (There is an equality constraint defined from
  role combination (R1, ....RN) to role
  combination (RN+1, ....R2n) )
ELSE
  IF (∃ Popstate1(UoD) [Popstate1(UoD)= EXT1(IM)] ∧
  ¬∃ Popstate2(UoD) [Popstate2(UoD)= EXT2(IM)] ∧
  ¬∃ Popstate3(UoD) [Popstate3(UoD)= EXT3(IM)] )

```

⁵³ If the role combinations are contained in more than one BIM then this confrontation with the real-life examples must be performed with every example document/user group combination that has verbalized sentence instances for this fact type. If this process leads to multiple set comparison constraint configurations on the focal role combinations then the following decision table must be used to determine the constraint configuration for the integrated requirements specification: no-no → no, excl-subset → no, excl-eq → no, no-excl → no, eq-subset → subset, no-subset → no, no-eq → no.

⁵⁴ We need to assign a unique name to every instantiated constraint in the requirements specification to distinguish this constraint among the union of constraints in the integrated requirements specification for the project.

```

THEN (There is an exclusion constraint defined
      from role combination ( $R_1, \dots, R_N$ ) to
      role combination( $R_{N+1}, \dots, R_{2n}$ ) )

ENDIF
ENDIF
ENDIF
ENDWHILE
END

```

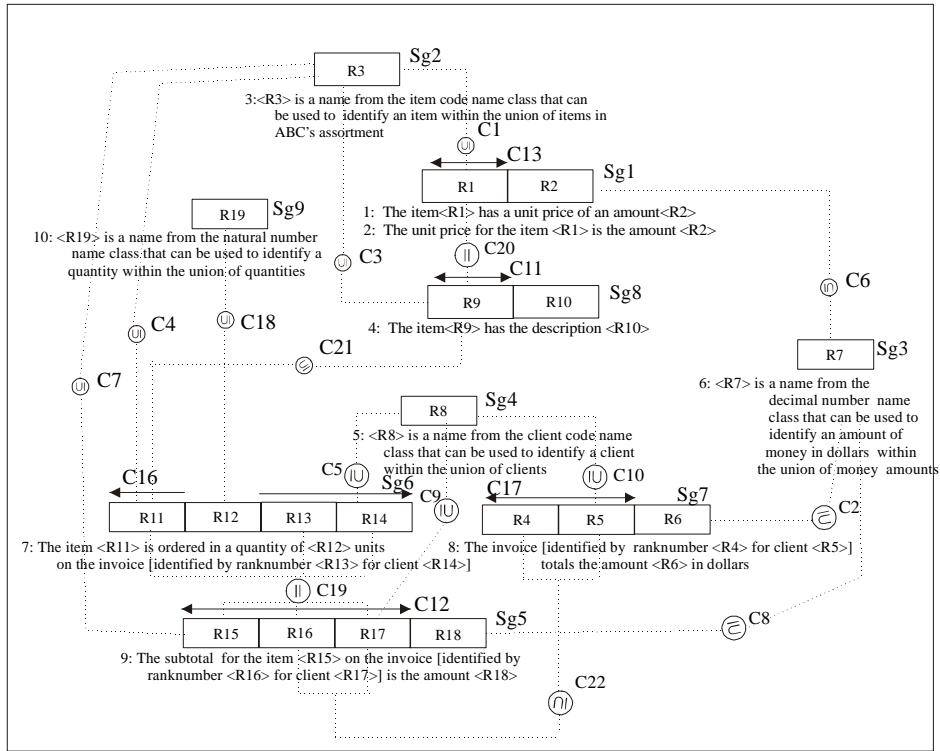


Fig. 6.12 Basic information model for example 6.1⁵⁵ with uniqueness and set-comparison constraint(s).

We note that population constraints c19, c20, c21, c22 are set comparison constraints that are derived in interaction with user by applying algorithm 7.

⁵⁵ Excluding the application concept repository (ACR).

6.6 THE POPULATION STATE TRANSITION CONSTRAINT MODELING PROCEDURE

The formal definition of a state transition constraint (see definition 5.11) includes all population state constraints. In this section, however, we will limit ourselves to those population constraints that can *not* be expressed as population state constraints. Furthermore, we will, consider the state transitions as pairs of population states. This means that we restrict the applicability of the transition constraints to an application information base that has a 'history' of 1 state. If we consider the invoicing example from this chapter, we can see that there exists a "business rule" stating that an item can not have a description other than its initial description.:

An item can not have a description other than its initial description.

In Twisk (1994), Twisk and van Montfoort (1994) and Spijkers (1994), the state transition constraints are derived in a user-analyst interaction wherein pairs of real-life examples are presented (representing a before and after state respectively).

The listing of all allowed (or the complement: the non-allowed) before and after combinations can be considered as a state transition constraint defined as a set of allowed before/after extensions. However, all possible future fact instances must be available in order to create those before/after collections. This is only possible in those cases where there exist *enumerable* value elements (Twisk, 1994) for the names in those roles.

Algorithm 8: Derivation of transition constraints

```
BEGIN TRANSITION ((I)BIM, UoDj, Gj)
  WHILE still fact types in (I)BIM
  DO take next fact type {ft}
    determine state equivalence classes for the
    functional role56 of ft
    WHILE not last state equivalence class
      before/after combination
    DO take next combination of state equivalence
      Classes.
      Let the user determine if this before/after
      combination is allowed in the UoD.
      IF combination not allowed
      THEN IF no state transition constraint is
        defined on the fact type yet
      THEN create a state transition
        constraint57 defined on the values
```

⁵⁶ The functional role of a fact type on which exactly one uniqueness constraint is defined is the role that is *not* covered by a uniqueness constraint.

⁵⁷ We need to assign a unique name to every instantiated constraint in the requirements specification to distinguish this constraint among the union of constraints in the integrated requirements specification for the project.


```

        for the functional role in the fact
        type.
    ENDIF
    add this combination to the non-
    allowed transition value
    combinations of the transition
    constraint
ELSE
    add this combination to the
    allowed transition value
    combinations of the transition
    constraint
ENDIF
ENDWHILE
ENDWHILE
END

```

We will try in most practical applications to give a state transition constraint by means of (decision) tables, formulas which serve as a legend for the analyst (see chapter 5 and appendix A) and that can subsequently be verbalized into declarative natural language sentences in which equivalent value-elements are grouped into *state equivalence classes*, for example:

The constraint 'c14' implies that there can not exist a [before extension of the basic information model] in combination with an [after extension of the basic information model] in which the extension for the role <R10> for a given item is different.

We see from this verbalization that we have introduced facilities in the constraint legend for denoting the type of extension (after or before) and the constraint(s) defined on one or more role(s) in either the after and/or before state into the analyst legend for the state transition constraints.

In figure 6.13 the resulting information model for the order invoicing application area is shown. We have added the business rule that states that the initial item description that is assigned to an item in ABC's assortment can never be changed. The legend for the interpretation of the population state transition constraint symbol (for constraint C14) that is used in figure 6.13 is given in figure A.2.

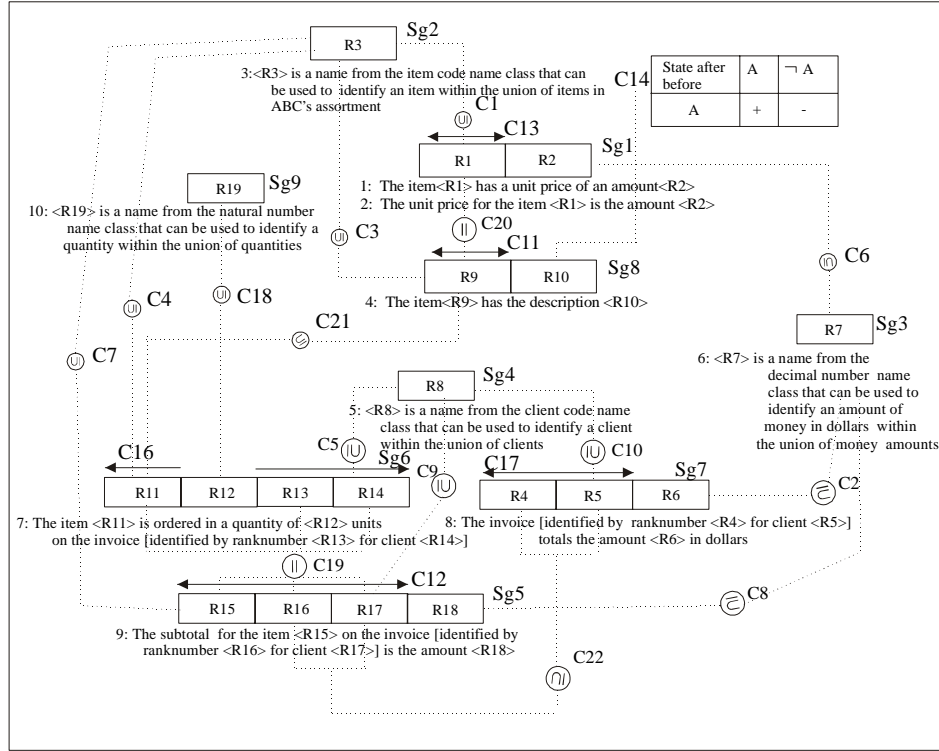


Fig. 6.13 Basic information model⁵⁸ for example 6.1 with uniqueness-, set comp and transition constraint(s).

6.7 THE DERIVATION RULE CONSTRAINT PROCEDURE

In this section we will give the way of working for the detection of *derivation rules* in an application subject area. We deliberately use the term application subject area, because the term Universe of Discourse until now has been used to demarcate the examples that we consider to be relevant for the process of requirements determination. The 'knowledgeable' users from the user groups in our (sub) UoD'(s) need to be able to verbalize these examples and be able to accept or reject example combinations in order to detect population state- and population state transition constraints. However, the responsibility for creating, inserting, deleting or deriving facts does not necessarily coincide with the responsibilities of the users from these user groups at all times. In order to make a distinction between users that have discretion with respect to the latter

⁵⁸ Excluding the application concept repository (ACR).

operations and users who do not have this discretion within the application subject area, we will introduce the concept of Sphere of Influence (SoI).

Definition 6.8. The *Sphere of Influence* (SoI) is a set of users that are considered relevant for the application subject area and that have responsibility for creating, inserting, deleting or deriving facts.

Let U be the set of users in the application subject area.
Let SoI be the sphere of influence
 $SoI \subseteq U$

Algorithm 9: derivation rule creation

```
BEGIN derivation rule creation(IM, UoD, SoI)
  WHILE still user groups left in UoD
    DO take the next user group from SoI {g}
      WHILE still fact types left in UoD to be considered
        for user (group) g
          DO take next fact type {ft ∈ IM}
            IF FT is derived under the responsibility of a
              User (group) g
            THEN
              IF (a pre-condition can be phrased in terms
                of the extension of IM
              AND a derivation formula can be specified in
                terms of the extension of IM)
              THEN create a derivation rule
                constraint59.
                Define a derivation rule argument
            ENDIF
          ENDIF
        ENDWHILE
      ENDWHILE
    ENDWHILE
  END
```

Although it is possible to specify derivation rules whenever a basic information model (BIM) of the UoD is known, we recommend performing this procedure after the population constraints have been derived. Performing the procedures in this order will enable us to capitalize on the knowledge that is contained in the population constraints when the derivation rules will be specified. We could extend algorithm 9 with quality control checks, e.g. the pre- and post-conditions and the possible outcomes of the derivation formula must be compatible with the population constraints.

⁵⁹ We need to assign a unique name to every instantiated constraint in the requirements specification to distinguish this constraint among the union of constraints in the integrated requirements specification for the project.

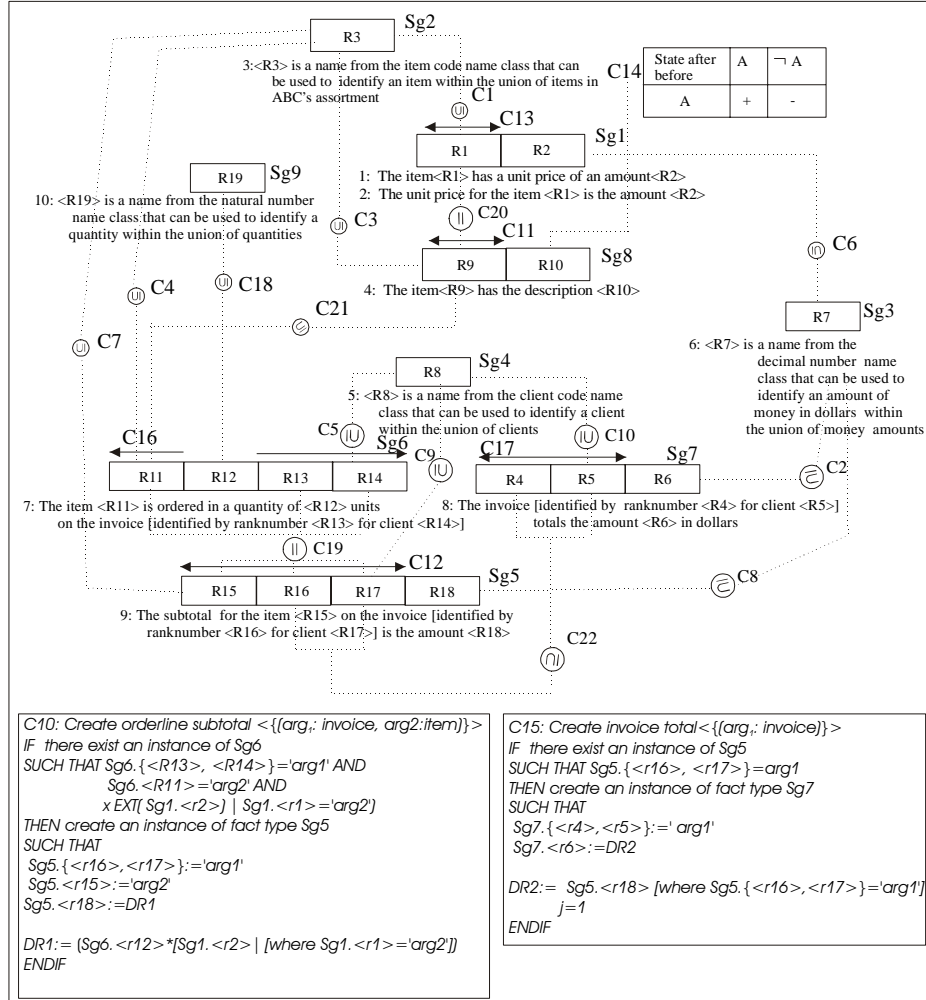


Fig. 6.14 Basic information model⁶⁰ for example 6.1 with uniqueness, set comparison, transition constraint and derivation rule constraints

6.8 THE IMPULSE CONSTRAINT PROCEDURE

The last type of constraints that must be instantiated in the NLM requirements determination way of working are the impulses in the application subject area in which the relevant events are derived together with the event arguments and where it is

⁶⁰ Excluding the application concept repository (ACR).

specified under what condition(s), the event occurrences of such an event will lead to the instantiation of (a) derivation rule(s) or the insertion and/or deletion of one or more fact instances to/from the application information base.

Algorithm 10: Impulse constraint derivation((I)BIM, SoI)

```

BEGIN
  WHILE still user groups left in SoI
    DO take next user from user group in SoI {g}
      WHILE still derivation rules left
        DO Take the next derivation rule constraint
          {drce ∈ (I)BIM}. Ask the users in {SoI}
            what event type(s) invoke such a
            derivation Rule61
          IF event type not listed
            THEN name the event and determine
              the event type argument
          ENDIF
          Determine the condition on the
            information base and event
            argument(s) under which the
            derivation rule is instantiated
          IF the condition is different from
            an existing condition on the
            same event type and derivation
            rule
            THEN Make a new (combined)
              Condition which contains the
              old and new condition type
            ELSE the impulse is already
              defined.
            ENDIF
          Create and name62 an impulse and
            determine the impulse mapper63 and
            name the impulse
        ENDWHILE
      WHILE still fact types left in (I)BIM64

```

⁶¹ It should be investigated whether a user defined event type coincides with an information base or internal event (see Prabhakaran and Falkenberg, 1988), e.g. fact instance of fact type FT2 inserted (arg1:date).

⁶² We need to assign a unique name to every instantiated constraint in the requirements specification to distinguish this constraint among the union of constraints in the integrated requirements specification for the project.

⁶³ The impulse mapper is a specification of how the values of the event argument determine the instantiation values for the arguments of the derivation rule(s) that is (are) instantiated in an impulse.

⁶⁴ Although impulse type constraints are constraints in the event perspective, we must also specify the unconstrained behaviour or the 'discretion' that users within the SoI have to add

```

DO Take next fact type {f}
IF fact instance of this fact type can be
inserted into the application base on
the discretion of user g without
invoking a derivation rule and without
any specific information base condition
other than the business logic that is
enforced by the population constraints
THEN add an impulse type constraint with
name insert {f} having an empty impulse
condition and invoking the operation
insert {f}
ELSE IF fact instance of this fact type
can be deleted from the
application base on the
discretion of user g without
invoking a derivation rule and
without any specific information
base condition other than the
business logic that is enforced
by the population constraints
THEN add an impulse type
constraint with name
delete{f} having an empty
impulse condition and
invoking the operation
delete{f}
ENDIF
ENDIF
ENDWHILE
ENDWHILE
END

```

and/or remove fact instances into/from the information base, as an impulse type to show that these insertion/deletion operations are allowed on their own. See for example impulse constraints c25 and c26 that tell us that it is allowed that the users in this UoD add and remove orderlines onto/from an invoice document (see figure 6.15).

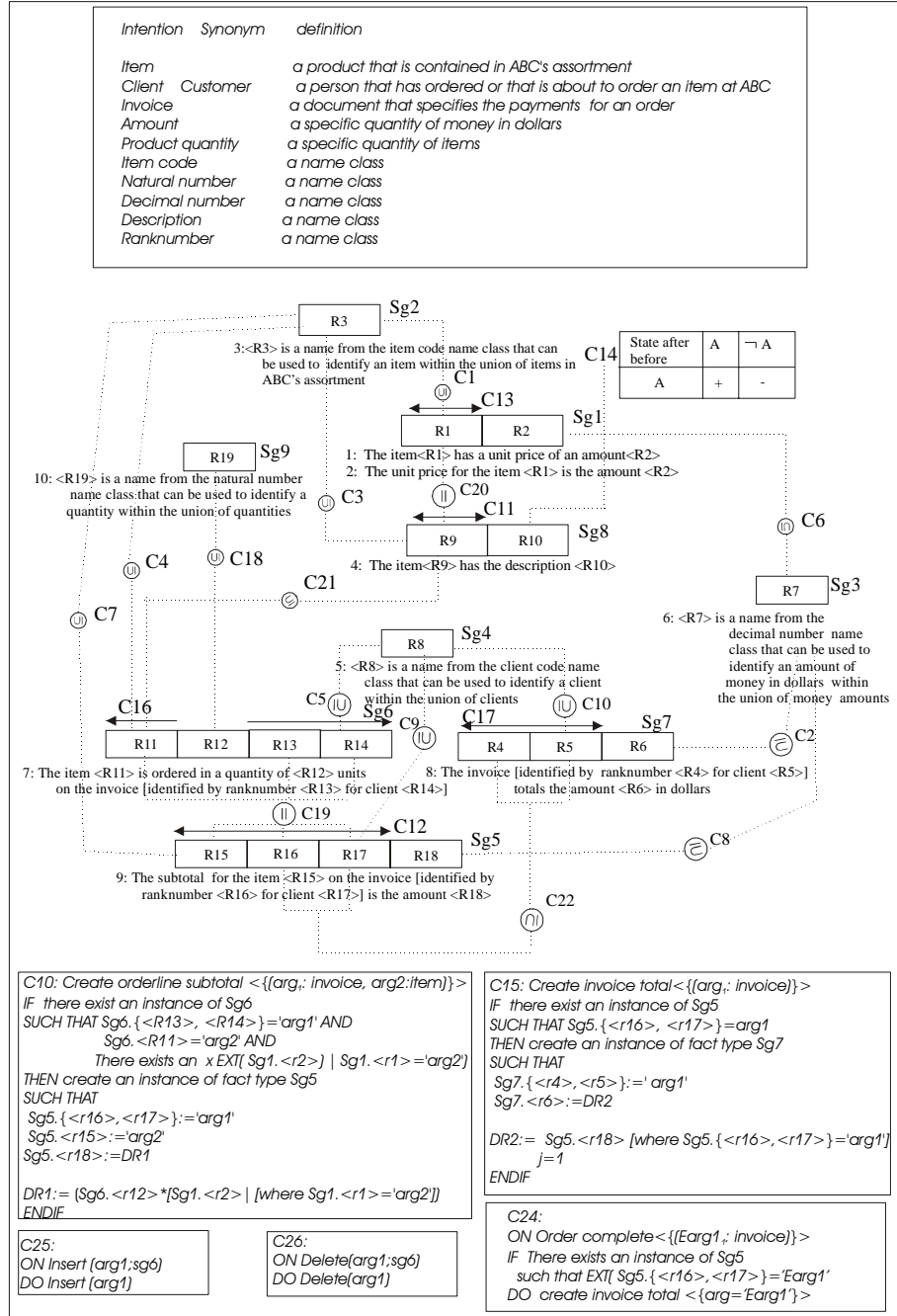


Fig. 6.15 Complete NLM requirements specification for example 6.1 that contains a basic information model with uniqueness, set comparison, transition constraint, derivation rule constraints and impulse constraints

Example 6.1 (ctd)

We assume that in the process of order intake for the ABC company at any time the invoice total only appears on the order clerk's computer screen when the clerk pushes a button named "order complete".

In figure 6.15 the complete requirements specification for the invoice example 6.1 is given. This requirements specification contains an atomic basic information model together with all instantiated constraints of all constraint types that we have defined in NLM.

6.9 THE WAY OF CONTROLLING IN NLM

In chapter 5 we have given the *way of modeling* in NLM and in sections 6.1 through 6.8 we have given an accompanying *way of working* for NLM. In this section we will give the way of controlling in NLM. The way of controlling deals with issues regarding the organization and (project) management of the requirements determination process. The NLM requirements determination method uses 'real-life' examples of business communication as a starting point for the requirements determination process. The first question that must be addressed in the way of controlling is how to arrive at these relevant examples in the requirements determination process. Secondly, we will discuss the project management restrictions in terms of precedence relationships that exist regarding the applications of the different stages that were defined in the way of working. Although the focus of this dissertation is on the conceptual analysis of the requirements we will touch upon a number issues that are relevant for requirements management at large, for example, scoping and requirements prioritizing .

6.9.1 The demarcation of the requirements determination project

In most information systems development projects, some type of information strategy planning has been taken place, that normally results in a lists of IS development projects that contain the priorities and in many times a business case in financial terms as to what the expected pay-offs of each IS implementation will be for the host organization (Jessup and Valacich, 1999: chapter 13). The demarcation of an IS project in such a business case is mostly laid down in a rather vague description, for example, a billing system, a HRM system and so forth. These information planning methods in general result at best in a list of business processes (activities) and business entities (or data classes). An example of such a planning method is Business Systems Planning (Sebus, 1981). These information planning methods result in 'models' on such a level of abstraction that it is not possible to specify the precise data model and the constraints defined on the data model. The first activity that must be carried out when an information systems development project must be carried out then is to establish clear borders regarding what functionality will be contained in the final information system and what functionality will be left out. In NLM this distinction between what must be incorporated into the system and what not is clearly demarcated in the types of user

examples that will be considered in the requirements determination process and the ones that will be left out ('scoping'). Also the issue of requirements prioritization can be dealt with in the stage of 'example' selection.

A second pass of prioritization can be implemented between the stages 'verbalization' and 'grouping', in which sentences that do not have priority for the application that has to be developed can be left-out. We strongly suggest that the information analyst will make explicit notes that state that these facts are contained on the example but for the current development 'time-box' (Jalote et al., 2004) are considered non-relevant. In a later development stage or 'time-box', such a requirement might, however, be a candidate for further analysis.

We note that the phenomenon of 'informal' organization must be dealt with during this stage in the requirements determination process, because it means that decisions have to be made regarding the incorporation of the 'informal' view(s) on the UoD with already selected 'formal' real-life informational documents (see also Oonincx, 1982:74).

For a newly designed information system or an information system that must operate in a reengineered environment, the availability of existing 'real-life' examples is not always guaranteed. In these situations, the requirements determination process must be preceded by a *reengineering* and/or *prototyping stage* in which the (to be) involved user(s) of the application domain will have to create 'mock-ups' or 'prototypes' of the examples that they are going to use in their future (reengineered) activities (algorithm 0).

It must be emphasized that the NLM requirements determination method assumes a strict distinction into domain knowledge that is possessed by the users in the application domain and the knowledge of the requirements determination process that is possessed by the NLM analyst. In NLM there is a clear 'separation-of-concerns' between the responsibility for the content of the requirements specification and the responsibility for the way in which this desired specification is created in a requirements determination process.

6.9.2 The required precedence of the requirements determination process in terms of the way of working

We will now derive the precedence relationships that we must respect when performing a requirements determination project using NLM. We assume that the 'real-life' types of examples that demarcate the functionality of the project have been selected. In principle the first stage in the requirements determination process can begin: create a basic information model. This stage means that for every example we will perform the basic information modeling procedure (algorithms 1 through 4) in combination with a domain user. The basic information model can then be added to the existing basic information models using the model integration procedure (algorithm 5). We note that the phenomenon of 'informal' organization must be dealt with again, during this stage in the requirements determination process, because decisions have to be made regarding the integration of the 'informal' view on the UoD with the 'formal' view. In many cases this will lead to the 'relaxation' of the constraints from the 'formal' view to be able to accommodate the requirements from the 'informal' view.

We remark that the application concept repository (ACR) ideally must be a shared document in which all concepts definitions of all sub-projects within the requirements determination process at large are defined piecemeal. Subsequently we will perform the uniqueness constraint, set-comparison and transition constraint derivation (algorithms 6, 7 and 8) that can be performed with those user(s) that use the individual or pairs of examples. After the integrated basic information model is finished it is already possible to determine the derivation rules (algorithm 9) and consecutively the impulses (algorithm 10).

Alternatively, it is possible to perform most activities (algorithms 1 through 10, except for algorithm 5), consecutively with the individual users. However, this comes with an expense, namely the additional derivation of set-comparison constraints in which multiple user (groups) are involved and the addition of derivation rules and impulse types that were initially left out because of demarcation issues. In this situation, additional impulses also need to be identified that cross the spheres of influences of individual users.

We can conclude that a NLM requirements determination process that is performed in an organizational context contains a large number of variables that can be set according to operational constraints, for example, analyst capacity, required delivery dates, desired completeness and preciseness regarding uniqueness and set-comparison constraints (and eventually other state constraint types) and domain user availability. A further degree of flexibility can be achieved when an organization for example, only needs a requirements specification that contains a basic information model, uniqueness constraints and derivation rules because the target implementation software does not support other constraint types. From a software development point of view NLM can be applied under different software development models: *waterfall development*, *iterative* and *iterative time boxing* (Jalote et al., 2004).

The semantic bridges from the natural language modeling methodology in the information perspective can be summarized as follows:

- semantic bridge 1): Capturing the general domain knowledge (or sentence groups) (algorithm 1: verbalization and algorithm 2: grouping).
- semantic bridge 2): Capturing the intention of the individual names (algorithm 3: classification and qualification)
- semantic bridge 3): Capturing the naming conventions (algorithm 1: verbalization and algorithm 3: classification and qualification)
- semantic bridge 4): Capturing the right level of atomization (algorithm 1: verbalization and algorithm 4: atomization)
- semantic bridge 5): Arbitrating on the primary naming conventions for the integrated UoD (algorithm 5: integration of (sub)-models)
- semantic bridge 6): Capturing domain generalizations and specializations for the integrated UoD (algorithm 5: integration of (sub)-models).
- semantic bridge 7): Capturing additional business rules that can be encoded as population constraints, derivation rule constraints and impulse constraints (algorithm 6: uniqueness constraint derivation, algorithm 7: set comparison constraint derivation, algorithm 8: transition constraint derivation, algorithm 9: derivation rule constraint creation and algorithm 10: impulse constraint derivation).

These semantic bridges constitute the foundation for a semantically correct requirements specification and therefore the approach for the way of working in the to-be designed requirements determination method fills a niche in the MIS research field that we discussed in chapter 1.

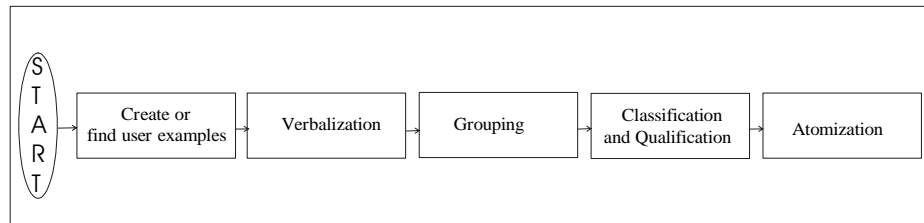


Fig. 6.16 AON network for activities in a NLM requirements determination project (I)

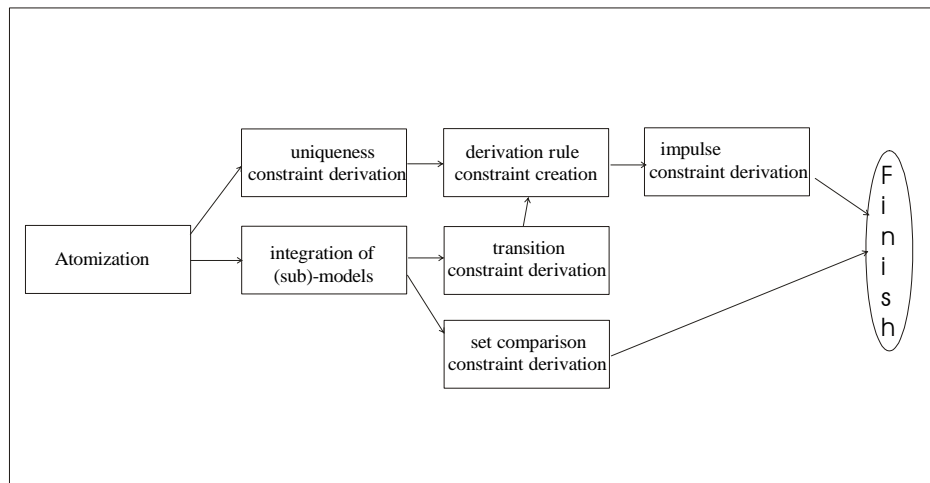


Fig. 6.17 AON network for activities in a NLM requirements determination project (II)

In figures 6.16 and 6.17 we have given a project plan for a requirements determination process in which all the minimal set of precedence relationships are extended with additional precedences that should facilitate the definition of derivation rules. In figures 6.16 and 6.17 an Activity-on-Node (AON) network (Mantel et al., 2001:113) for this way of controlling is given.

6.9.3 Resource planning for a requirements determination project in NLM's way of working

The application of NLM will allow us to establish metrics for the requirements determination process. It is easy to estimate the number of 'analysis steps' that are needed for the derivation of uniqueness and set comparison constraints when a basic information model is available. The number of analysis steps for determining the population, derivation and impulse constraints is in principle deterministic. The

resource planning in terms of ‘analysis steps’, therefore can be made relatively easy when a basic information model is available. It is also possible to base a capacity planning upon ‘experience’ data regarding the number of communication examples and the average number of fact types that can be traced to an example. With respect to the management of human resources, metrics can be determined for the level of experience of analysts.

6.10 CONCLUSIONS ON THE WAY OF WORKING AND THE WAY OF CONTROLLING IN NLM

In order to arrive at the basic information model (BIM) of an application UoD, in which the modeling constructs from the data model can be instantiated for the application UoD, we have specified algorithms 1 through 4. A significant extension to the ‘state-of-the-art’ in requirements modeling is the definition of the application concept repository (ACR) in which generalizations and specializations can be incorporated and in which the intentions, their semantic definitions and their name classes are incorporated. In order to derive all instances of the static constraint types for which we have supplied a constraint legend, we have specified state constraint derivation algorithms 6 and 7. NLM has the provisions to accommodate other constraint types whenever a notational legend is provided and an accompanying instantiation algorithm can be given. In order to derive all instances of the transition constraints we have specified transition constraint derivation algorithm 8 as sub-procedure in NLM’s way of working . In order to derive all instances of the derivation rule constraints we have specified the derivation rule constraint algorithm 9 as sub-procedure in NLM’s way of working in which the precise specification (or derivation formula) can be established. In algorithm 10 we have incorporated the question in which an internal event can lead to the execution of a derivation rule or another information base event. Furthermore, the algorithm systematically confronts the users in the SoI with derivation rules and tries to elicit the potential ‘external’ events that might invoke such a derivation rule . We can conclude that the way of working in NLM fully complies to *RDM 12* from chapter 4.

Furthermore, the explicit incorporation of an integration algorithm (algorithm 5) into the way of working is fully in line with requirement *RMD 13* (view integration sub-procedure).

The application of the natural language axiom in an organizational setting in which domain users are held responsible for the ‘knowledge content’ and in which users are confronted with combination of real-life examples can be considered a transformation from implicit tacit knowledge held by these users into explicit knowledge in the requirements specification. This fulfills requirement *RMD 14*.

The application of the natural language axiom in NLM also allows us to apply NLM in many organizational settings, ranging from abstract to tangible UoD’s and from natural language descriptions to other descriptions that can only be understood by users. This leads to compliance to *RMD 15*.

The sub-division of the modeling procedures in NLM’s way of working into formal algorithms has been done in such a way that the amount of modeling steps that

have to be performed by (an) analyst(s) is minimized and therefore NLM is in line with *RDM 16* from chapter 4.

The *reconstruction* check in algorithm 1, the *completeness* check in algorithm 2, the *consistency* check in algorithm 3, the *reference* check in algorithm 4, the *ontological equivalence* check in algorithm 5, and the *N-1 law* check in algorithm 6 are explicit quality-assuring verification sub-procedures that are built-into NLM's way of working. Every algorithm within NLM's way of working contains (bold-fonded) parts that confronts users with his/her assumptions. We can conclude that this precise specification of the NLM modeling procedure in a number of algorithms fulfills requirement *RMD 17* as given in chapter 4.

The elements that constitute the *way of controlling* in chapter 1 are all covered in the NLM requirements determination method. NLM enables the management of analyst resources, to plan and control their efforts using project management techniques, e.g. precedence analysis and capacity requirements and quality management. NLM is fully scalable in terms of project size and complexity. It's natural language features and the mandatory referencing schemes that are imposed upon the way of working of the analyst, will enable the analysts to record and maintain all project information and analysis results (*RMD 18*). The existence of a mandatory naming conventions for the elements (projects, user examples, sentence groups, fact types, constraints) in the subsequent results in the requirements determination process in NLM fulfills the traceability requirement (*RMD 19*).

6.10.1 The added value of the NLM requirements determination method

We can conclude that the NLM requirements determination procedure explicitly shows the separation of concerns between the analyst and the user in the process of requirements modeling by providing the *semantic bridges* for this analyst-user dialogue. In addition to the creation of a NLM requirements specification that is an allowed extension of the NLM meta model from appendix B that is a syntactically correct specification (see chapter 1), we need guidance on what specific extension of this meta model reflects the domain semantics in a *precise* and a *consistent* way. We have shown that such a semantic correct specification will be achieved when the algorithms that we have introduced in this chapter will be applied in a requirements determination project in which the sequence of their application is performed under the precedence requirements that were given in figures 6.16 and 6.17.

Although a number of procedures in the NLM requirements determination method at first sight have a 'trivial' appearance, the consistent application of the procedures in the 'way-of-working' in this chapter in practice has proven to improve the 'quality' levels of the resulting requirements specification, because even the experienced analyst can always 'fall back' on the procedure in those situations in which the application subject area becomes too complex. The biggest advantage, however, is that inexperienced analysts will be able to create requirements specifications that have the same quality level as the specifications that are created by experienced analysts. In a project in which the NLM requirements determination method is applied for the creation of a requirements specification, the division into sub-projects and the order in which these sub-projects are executed does not have an impact on the final specification. The NLM requirements determination method, therefore is fully scalable in terms of the complexity of the subject area, analyst capacity and user availability.

Another advantage of the application of the NLM requirements determination method is in the full accountability of the modeling results in which the user inputs and the analyst modeling transformations are precisely defined.

An interesting practical application of NLM for integrating business processes in the implementation process of a SAP R/3 module at multi-divisional pharmaceutical company, was carry out by Natasja Enter as her graduation project in international business studies (Enter, 1999)⁶⁵. In this project NLM was used to reduce the problems around the complexity of the system by defining the core concepts and their relationships within the integrated system in a precise way.

An application of (an earlier version of) NLM on a static business knowledge domain, e.g. general ledger accounting was performed by Marjan Wolthuis (Wolthuis, 1997), which the business process redesign of the accounting function was illustrated. In this master's thesis, the basic information model for the accounting application domain together with uniqueness, set comparison and lexical constraints were derived. Subsequently, the derivation rule constraints in the accounting application domain were derived. She concluded in this thesis that a number of 'manual' verification steps in this domain can be considered redundant in the IT-era.

6.11 REFERENCES

Beynon-Davies, P., Bonde, L., McPhee, D., Jones, C. (1997): A Collaborative Schema Integration System. *The journal of collaborative computing* **6**: 1-18

Biller, H. (1979): On the notion of irreducible relations. In: Bracchi, G., Nijssen, G. (eds.): *Data base architecture*, North-Holland, Amsterdam : 277-295

Bollen, P. (2002): Using the OO paradigm for conceptual modeling: the need for a methodology', In: M. Hunter and K. Dhanda (Eds.) *proceedings ISoneworld 2002*, Las Vegas, U.S.A.

Capuchino, A., Juristo, N., van de Riet, R. (2000): Formal justification in object-oriented modelling: A linguistic approach. *Data & Knowledge Engineering* **33**: 25-47

Choobineh, J., Venkatraman, S. (1992): A methodology and tool for derivation of functional dependencies from business forms. *Information Systems* **17**(30): 269-282

Enter, N. (1999). The semantics of the CIC SAP R/3 core. Final thesis. International business studies. University of Maastricht.

Falkenberg, E. (1976): Significations: the key to unify data base management. *Information Systems* **2**: 19-28

⁶⁵ For confidentiality reasons CIC is not the real name of this company.

- Ter Hofstede, A., Proper, H., van der Weide, T. (1997): Exploiting fact verbalisation in conceptual modelling. *Information Systems* **22**(6/7): 349-385.
- Hoppenbrouwers, J., van der Vos, B., Hoppenbrouwers, S. (1997): NL structures and conceptual modeling: Grammalizing for KISS. *Data & Knowledge Engineering* **23**: 79-92.
- Jalote, P., Palit, A., Kurien, P., Peethamber, V. (2004) : Timeboxing : a process model for iterative software development. *The journal of systems and software* **70**: 117-127
- Jessup, L., Valacich, J. (1999): *Information systems foundations. Que education and training.*
- Johannesson, P. (1997): Supporting schema integration by linguistic instruments. *Data & Knowledge Engineering* **21**: 165-182.
- Kaufman, L., Rousseeuw, P. (1990): *Finding groups in Data: an introduction to cluster analysis.* Wiley & Sons.
- Kent, W. (1978): *Data and reality*, North-Holland, Amsterdam
- Kung, C. (1990): Object subclass hierarchy in SQL: a simple approach. *Communications of the ACM* **33** (7): 117-125.
- Kwan, I., Fong, J. (1999): Schema integration methodology and its verification by use of information capacity. *Information Systems* **24** :355-376
- Land, F. (1980): A critical view of some recent assertions about MIS and DBMS. *Information & Management* **3** : 129-131.
- van der Lek, H., Bakema, G., Zwart, J. (1992): De unificatie van objecttypen en feittypen. *Informatie* **34** (5): 279-295 (in dutch)
- Leung, C. , Nijssen, G. (1987): From a NIAM Conceptual Schema onto the Optimal SQL Relational Database Schema. *Australian Computer Journal* **19**(2):69-75.
- Leung, C. , Nijssen, G. (1988): Relational Database Database design using the NIAM conceptual schema. *Information Systems* **13**:219-227.
- Lim, E., Chiang, R. (2000): The integration of relationship instances from heterogeneous databases. *Decision Support Systems* **29**: 153-167
- Mantel, S., Meredith, J., Shafer, S., Sutton, M. (2001): *Project management in practice.* Wiley and Sons
- McBrien, P., Poulouvasilis, A. (1998): A formalisation of semantic schema integration. *Information Systems* **23** (5): 307-334

Meersman, R. (1994): Some methodology and representation problems for the semantics of prosaic application domains. In Z. Reis and M. Zemanskosal (eds.): Methodologies for intelligent systems. 39-45.

Mirbel, I. (1997): Semantic integration of conceptual schemas. Data & Knowledge Engineering **21**: 183-195

Mintzberg, H. (1991): Organisatie structuren. Prentice-Hall/ Academic Service (in dutch)

Navathe, S., Sashidar, T., Elmasri, R. (1984): Relationship merging in schema integration. Proceedings of the 10th international conference on very large data bases 78-90

Nijssen, G. (1986): On experience with Large-Scale Teaching and Use of fact-based Conceptual Schema's in Industry and University. In R. Meersman and T.B. Steel Jr. (eds.): Proceedings of IFIP conference on Data Semantics (DS-1), Elsevier North-Holland.189-204.

Nijssen, G., Halpin, T. (1989): Conceptual schema and relational database design: A fact based approach, Prentice-Hall, Englewood Cliffs.

Nijssen, G., Twine, S. (1989): De rol van formele methoden bij het effectief ontwerpen van een relationele database. Informatie **31**(12): 966-976 (in dutch)

Oonincx, J. (1982): Waarom falen informatiesystemen nog steeds ? Samsom. Alphen aan den Rijn. (in dutch)

Prabhakaran, N., Falkenberg, E. (1988): Representation of Dynamic Features in a Conceptual Schema. Australian Computer Journal **20** (3): 98-104

Sebus, G. (1981): BSP: 'Business Systems Planning'. Informatie **23**: 142-152 (in dutch)

Silva, M., Carlson, C.R. (1995): MOODD, a method for object-oriented database design. Data & Knowledge Engineering **17**: 159-181

Shoval, P., Zohn, S. (1991): Binary-relationship integration methodology. Data & Knowledge Engineering **6** : 225-250

Smith, J., Smith, D. (1977): Database abstractions: Aggregation. Communications of the ACM **20**(6): 405-413

Spijkers, P. (1994): A manager's experience with NIAM-ISDM in a large scale critical project, in G.M. Nijssen and J.Sharp (eds.), Proceedings second NIAM- ISDM working conference. pp.A1-A27.

- Tehrani, S., Nijssen, G. (1985): UCL: A User-friendly Conceptual Language. Australian Computer Journal **17**(4): 174-178
- Tseng, V., Mannino, M. (1989): A Method for Database Requirements Collection. Journal of Management Information Systems **6**(2): 51-75
- Tsichritzis, D. , Klug, A. (1978): The ANSI/X3/SPARC DBMS framework. Information Systems **3**: 173-191.
- Twisk, F.(1994): Effektieve InformatieSysteemOntwikkeling met NIAM-ISDM, , Kluwer Bedrijfswetenschappen, Deventer (in dutch).
- Twisk, F. and van Montfoort, R. (1994): UI en NIAM-ISDM: een concreet alternatief (1). Informatie **36** (7/8): 438-446 (in dutch).
- Wolthuis, M. (1997): Afstudeerscriptie: Boekhouden op basis van Universele Informatiekunde. University of Limburg. (in dutch)
- Wrycza, S. (1990): The ISAC-driven transition between requirements analysis and ER conceptual modeling. Information Systems **15**(6): 603-614
- Wu, J-H., Doong, H-S., Lee, C-C., Hsia, T-S, Liang, T-P. (2002): A methodology for designing form-based decision support systems. Decision Support Systems **32** : 1-23

CHAPTER 7

CONCLUSIONS, GENERAL DISCUSSION AND RECOMMENDATIONS

7.1 INTRODUCTION

In this final chapter of this thesis we will evaluate our main research question that we have stated in chapter 1:

“Does there exist a requirements determination method that is applicable in a wide range of business organizations and that can be used for specifying the complete domain requirements for a given business application subject area in an efficient, and formal way ?”

In chapter 1 we illustrated the relevance of the field of requirements determination. We have concluded that the theory development on the field of our research topic: *requirements determination*, has taken place in a number of fields of study. Among those fields of study are: *management information systems, information systems development methodologies, speech-act theory, ontology and conceptual modeling*. Furthermore, we concluded that the theory development on the field of requirements determination has been plentiful but a sound methodology for the specification of application requirements is missing. In chapter 1 we have also pointed at a MIS research niche that we want to exploit. This research niche is concerned with the *semantic verification* of (initial) requirements (Dullea et al., 2003). This has lead us to formulate the design object of the research in this study as: the development of a method for requirements determination that has modeling provisions that guide an analyst in eliciting the initial requirements from domain users and that contains a semantic verification or validation procedure that guarantees user validation of the requirements.

We have sketched a research approach that is suitable for our research purposes: the ‘design-research’ approach. In this approach we have applied the ‘design-research’-cycle (Van Engelen and Van der Zwaan, 1994) in which we first have to establish a design objective and subsequently a design specification in order to be able to evaluate existing designs for a requirements determination method.

The design specification for a requirements determination method was given in chapter 2 in which we have synthesized from the literature, four (groups of) criteria that a requirements determination method must comply to: *domain richness, completeness, efficiency and formality*.

In chapter 3 we have covered the first group of ‘alternative’ designs, namely an evaluation of the existing alternative designs for requirements determination methods by surveying the existing literature on requirements determination, e.g. DFD’s ISAC, EER, UML, ARIS and ORM. In the research methodology literature this is called the ‘evaluation problem’. In chapter 3 we have concluded that no single

approach fulfills the design specification that was given in chapter 2. In case no single existing design can be found that conforms to our design specification we need to develop an alternative design that must fulfill the requirements that we have postulated in the final research question (the development problem).

In chapter 4 of this thesis we subsequently have derived a detailed design specification that resulted in 19 requirements method demands (RMDs) for a 'to-be' designed RDM.

In chapter 5 we have documented the way of modeling for a proposed requirements determination method: Natural Language Modeling (NLM) and in chapter 6 we have documented the way of working and the way of controlling for NLM. Chapters 5 and 6, therefore, constitute the results of the generation of an alternative design as is given in the 'generation of alternative designs' stage from the 'design-research'-cycle (Van Engelen and Van der Zwaan, 1994).

We concluded in sections 5.10 and 6.10 that the NLM requirements specification language, its set of accompanying modeling algorithms and its project management precedence logic jointly fulfill all 18 design requirements that were derived in chapter 4 for a to be designed RDM. Hence, we can conclude that Natural Language Modeling (NLM) is a requirements determination method (*RDM*) that complies to the design objective in this thesis. Natural Language Modeling, therefore, provides the answer to our main research question from chapter 1.

7.1.1 Organization of chapter 7

This chapter is organized as follows. In section 7.2 we will summarize the research findings for the research sub-questions that were derived in chapter 1. Furthermore, we will, show in section 7.2 how the answers to the sub-questions of our research lead to the answer to our definite research question. In section 7.3 we will defend our research methodology from a retrospective point of view. In section 7.4 we will give recommendations for future research areas in the field of Management Information Systems. In section 7.5 we will give recommendations for practitioners in the MIS field. Finally, in section 7.6 we will reflect upon the research questions for this thesis, our research approach, the research outcome and the research process.

7.2 RESEARCH FINDINGS

In chapter 1 we have introduced the subject of this study: requirements determination. We have also sketched the application of computerized information systems in the past 50 years and we have shown that in recent history the emphasis within the application of information systems in businesses has been on ERP systems. We concluded that a complete and consistent requirements specification is still needed as a starting point for the customisation and implementation of an ERP system. Existing approaches for requirements determination often use a natural language statement of the initial requirements as a starting point for the creation of a requirements specification (Goldin and Berry, 1997). However, none of the tools that were mentioned in this study give much help on how such an (initial) language statement can be obtained.

The application of the ‘design-research’-cycle to carry out the research that will enable us to achieve the (preliminary) goal of our research will lead to a number of research sub-questions.

Research sub question 1 :

What are according to the existing requirements determination literature, the quality criteria for a requirements determination method that can be used for eliciting, verifying and specifying the complete domain requirements for a given business application subject area in a wide range of business organizations in an efficient and formal way ?

Research findings for sub question 1 :

We have synthesized (four groups of) criteria for a requirements determination method: *domain richness, completeness, efficiency and formality*.

Domain richness criterion

A literature review of the requirements determination literature has lead to four dimensions for the domain richness criterion.

The first dimension that characterizes a domain is what we have labelled the dimension *perception*. The actual ‘value’ on this dimension for any given domain can range from “uniform for all users” (*similar* perception) to “different for all users” (every user has a *different* perception of a underlying reality) (Galliers and Swan, 2000).

The second dimension that characterizes a domain is labelled the dimension *turbulence*. This dimension actually represents the extent (or frequency) in which the rules, information and procedures in an application domain are subject to change (Land, 1998).

The third domain dimension that we have derived from our literature study in chapter 2 is the dimension *tacitness*. The tacitness can range from a fully ‘tacit’ application domain in which no single knowledge creating process is explicit to a fully ‘explicit’ UoD in which every knowledge or information generating process can be made explicit.

The fourth dimension for domain richness, is the dimension *anchoring* (Bubenko and Wangler, 1992; Flynn and Warhurst, 1994), ranging from a ‘tangible’ starting point for the requirements determination process to an ‘abstract’ starting point.

Completeness criterion

The completeness criterion for a requirements determination method has been operationalized along two dimensions that define *what* must be incorporated in a requirements specification for application domains. The first dimension is the perspective dimension: the *data-oriented* perspective, the *process-oriented* perspective and the *behaviour-oriented* perspective. The conceptual data-oriented perspective should concentrate on the business data and must capture the domain concepts, the definition and the naming conventions for those domain concepts, the semantic

relationships between the domain concepts and other ‘static’ and ‘structural’ knowledge in the enterprise. The process-oriented perspective should be able to capture the business activity and user perceivable tasks and describe the ‘cross-reference’ on how the ‘elements’ in the static structure are created, or what procedures exist for the creation of instances of semantic relationships. Finally, the behaviour-oriented perspective describes how ‘events’ can be cross-referenced to ‘elements’ in the process- and data-oriented perspectives. This means that any requirements specification should potentially consist of models that covers these three (conceptual) perspectives. The second dimension is concerned with the question what elements must be contained in every perspective (see table 7.1)

Table 7.1 Types of rules within perspectives for completeness criterion

| | state | state | action |
|---------------------------|--------------|--------------------|---------------------|
| Data-oriented | Data model | Static constraints | Dynamic constraints |
| Process-oriented | | Static derivation | |
| Behaviour-oriented | | | Dynamic rules |

Efficiency criterion

Another criterion that we can use for evaluating requirements determination methods is concerned with the amount of resources that are needed to create a requirements specification when such a requirements determination method is applied in a given application UoD. This criterion is generally known as *efficiency*. The operationalization of this criterion for the purpose of evaluating requirements determination methods has taken place for the *way of modeling* and the *way of working* as well as the *way of controlling*.

With respect to the *way of modeling* the number of equivalent modeling constructs in the specification language determines the value on this criterion,

With respect to the *way of working* of a requirements determination method we can say that the availability of a (set of) procedure(s) that guides an analyst in the requirements determination project will determine the efficiency of the way of modeling of requirements determination method.

With respect to the *way of controlling* we can define efficiency on two areas. Firstly, the area of quality management. In this philosophy, quality deficiencies must be prevented by having a number of ‘quality-checking’ sub-procedures. Secondly, the way of controlling is concerned with the project management of the requirements determination project. The efficiency regarding these project management issues must be measured in terms of three project targets: *performance*, *cost* and *time* (Mantel et al., 2001)

Formality criterion

The relevant formality dimension to which a requirements specification must comply are the following: *consistency* and *preciseness*. This means that the modeling constructs that are used for creating requirements specifications in the different perspectives must be formally defined, in order to prove their consistency. Secondly,

the way of working, must be formal: a formal modeling procedure(s) must exist that precisely specifies how the consistent modeling constructs that were defined in the way of modeling, must be instantiated in a requirements determination project in order to obtain semantic correctness in complicated application subject areas.

With respect to the way of controlling we must be able to formalize the planning of activities that have to be carried out in a requirements determination project, for example in a precedence diagram and we must be able to give provisions that enable traceability.

Research sub question 2 :

Why do the existing requirements determination approaches from the literature not comply with the quality criteria for assessing requirements determination methods ?

Research findings for sub question 2 :

Research findings for the domain richness criterion

The application of a requirements determination method must lead to a requirements specification that reflects the (possibly) different perceptions of an underlying reality by different user groups. It is possible to reflect these different perceptions by using the EER, UML and ORM approaches, whenever they are embedded in a procedure that enables an analyst to integrate the different views from different user groups on the 'underlying reality' by integrating the sub-schemas of these users into a final 'overall' requirements specification in which the different perceptions are made explicit. The EER, UML and ORM approaches that we have discussed in chapter 3 do not give provisions for this.

The 'turbulence' dimension characterizes the extent in which an application domain is subject to changes in the business data and business rules. We concluded that the EER and UML approaches are most prone to remodeling because of the multitude of information bearing constructs. ORM has a problem with a multitude of naming conventions which might lead to unstable models.

With respect to the 'tacitness' dimension, the EER, UML and ORM approaches basically have the assumption that users will be able to express their initial requirements in natural language. This restricts the applicability of these approaches to those domains that exclusively contain explicit knowledge

With respect to the 'anchoring' dimension, the requirements determination processes in which we use EER and UML models for our specification language are in principle not limited to any specific range on the anchoring scale. ORM is anchored in *familiar examples* and it requires the domain expert to come up with these real examples and therefore is applicable for those domains that are on the 'tangible' side of the anchoring scale. The initial language in ORM is the language of verbalizable *familiar examples* and it requires the domain expert to verbalize these examples in (a subset of) natural language. The initial language in EER and UML is not specified but it can be anything because no procedure is given how to get from an initial requirements statement to the EER diagram or UML model(s).

Research findings for the completeness criterion

With respect to the encoding capabilities of a given approach for the data model, the static constraints, the dynamic constraints, static derivation rules and dynamic rules, the main conclusion is that no single approach is able to comply fully with the *completeness* criterion. There exists a large difference between the families of approaches and even between members within a given family in terms of the extent in which the application domain semantics can be expressed in the *data model*, and as *static* or *dynamic constraints*, *static derivation (rules)* and *dynamic rules*. Furthermore, the existing approaches, generally, lack a formalized way of working that will assure completeness, in the sense that all existing relationships and constraints in an application subject area, will be ‘detected’ by the analyst in the requirements determination project. This means that there still is an opportunity to improve the requirements determination approaches we have surveyed in chapter 3 in terms of completeness.

Research findings for the efficiency criterion

With respect to the *efficiency* criterion we must remark that in EER and UML in a number of cases remodelling is necessary because of the application of the attribute modeling construct in the initial requirements specification. The main finding of this literature survey is that 2 out of these 3 approaches use more than 2 information bearing constructs which can lead to instable requirements specifications. Furthermore, the non-existence of a precise modeling procedure in all approaches might lead to unnecessary rework in the requirements determination process, because verification is not enforced. Furthermore, with respect to the way of controlling, the existing approaches do not cover the project management and quality assurance steps.

Research findings for the formality criterion

With respect to the *consistency* dimension we can conclude that in many (E)ER approaches and in the UML it is not possible to use a single definition for minimum cardinalities or multiplicities across all types of semantic relationships. In UML it is not clear how the modeling concepts that are used in the 9 different diagram types are related on the level of an application requirements specification. In ORM an inconsistency is found with respect to the treatment of derived fact types, sometimes they will be contained in the application information grammar sometimes they will not.

With respect to the *preciseness* dimension we remark that the optionality of some modelling constructs in all three approaches that we’ve studied might lead to imprecise requirements specifications. With respect to the questioning of assumptions we can conclude that in EER, UML and ORM no procedure exists that allows an analyst to question the assumptions on which the utterance of the domain semantics is based. The position of these approaches is basically that the domain requirements that are uttered by the user are encoded in the model 1-on-1. ORM claims to perform checks on sample populations, however, it does not give guidelines on how to formally perform these checks in a dialogue with the responsible domain user.

Table 7.2 Requirements method demands for the way of modeling

| RMD | Requirements method demands for the way of modeling |
|------------|--|
| 1 | A to-be designed RDM must contain 1 information bearing modeling construct. This construct must be able to express the complete, precise and consistent communication semantics of any N-ary relationship |
| 2 | The modeling construct(s) for naming conventions must allow for one domain-based naming convention and must be able to capture the semantics regarding the context in which the naming convention is valid. |
| 3 | The to be designed requirements method must contain a role construct and an explicit naming convention for roles |
| 4 | The static constraint types in the to-be designed requirements method must at least contain those types that enable us to encode those business rules that can be encoded by relationship cardinalities in EER and UML |
| 5 | A requirements specification that is the result of the application of the to-be designed requirements determination method must be able to adapt to an evolving application logic without unnecessary remodeling |
| 6 | The definition of an application object or entity in the to be designed requirements method must not imply that it can exist on its own by default |
| 7 | The definition of the static constraint types in the to-be designed requirements method must be the same for all arities of the semantic relationships in the data model and must contain an explicit reference to the elements in the data model. |
| 8 | The definition of the dynamic constraint types in the to-be designed requirements method must enable us to explicitly refer to the (actual and projected states of the) application's data base |
| 9 | The definition of static derivation (rules) in the to-be designed requirements method must contain an explicit reference to the elements in the data model that serve as an input for the static derivation (rule) and it must contain a precise specification on how these elements lead to the result of the static derivation (rule) |
| 10 | An internal event in the to-be designed requirements method must be defined as the insertion or deletion of a specific piece of domain knowledge into or from the application's data base An external event in the to-be designed requirements method must be defined as something that happens in the application domain and that can lead to the insertion or deletion of a specific piece of domain knowledge into or from the application's data base or the execution of a static derivation rule (eventually) under some condition on the content of the application's data base |
| 11 | A condition in the to-be designed requirements method must be defined as a proposition on the application's information base that must yield the value true or false when evaluated at any point in time |

In chapter 3 of this thesis it was concluded that for the three requirements determination approaches that were studied in detail in this chapter (E)ER, UML and ORM no single approach fulfills all the quality criteria for a RDM that were derived in chapter 2.

Research sub question 3 :

What are the necessary elements for the way of modeling, the way of working and the way of controlling for a requirements determination method so that this method complies with the quality criteria that we have given for the design specification ?

Research findings for sub question 3 :

The diagnosis of these modeling deficiencies in the state-of-the-art in requirements determination has lead to the formulation of 18 requirement method demands (RMD's) for the specification of a to-be designed requirements method in chapter 4. We have divided the 19 RMD's into RMD's for the way of modeling (table 7.2), RMD's for the way of working (table 7.3) and RMD's for the way of controlling (table 7.4).

Table 7.3 Requirements method demands for the way of working

| RMD | Requirements method demands for the way of working |
|------------|---|
| 12 | The definition of the modeling constructs for the data model in the to-be designed requirements method must be accompanied by some kind of guidance on how all instances of these modelling constructs can be found in an application subject area. The definition of the constraint types in the to-be designed requirements method must be accompanied by some kind of guidance on how such instances of a constraint type can be found in an application subject area. |
| 13 | A view integration sub-procedure must be defined in the to-be designed RDM in which it is specified how an analyst must carry out the integration of views on the application domain by user (groups) that have a different perception on the 'underlying' reality |
| 14 | The to-be designed requirements determination method must provide facilities for transforming implicit tacit knowledge into explicit knowledge |
| 15 | The to be designed requirements determination method must accommodate every possible starting point in the requirements determination process ranging from abstract to tangible; ranging from natural language description to documents that can only be understood by domain users. |
| 16 | Formal modeling procedure(s) must be defined in the to-be designed requirements method in which it is precisely specified how an analyst must carry out a modeling step in the most efficient way. |

Table 7.4 Requirements method demands for the way of controlling

| RMD | Requirements method demands for the way of controlling |
|------------|---|
| 17 | The way of working in the to-be designed RDM must have explicit formal quality assuring sub-procedures for the activities in the work breakdown structure and formal checks that enables an analyst to validate the information that is supplied by the user and that confronts a domain user with his/her assumptions and enables a user to validate the information that is supplied to the analyst |
| 18 | The way of working in the to-be designed requirements determination method must have a work breakdown structure that allows to formally plan the activities in a requirements determination project. |
| 19 | The way of modeling and the way of working in the to-be designed RDM must have provisions that enable traceability. |

We now go back to our main research question

Main research question :

Does there exist a requirements determination method that is applicable in a wide range of business organizations and that can be used for specifying the complete domain requirements for a given business application subject area in an efficient and formal way ?

Research findings for main research question :

In chapter 5 NLM's way of modeling was defined. The modeling constructs for the *specification* of an application requirement in a basic information model and the accompanying constraints and their naming conventions were given. Furthermore, their applicability and generalizability in business UoD's was illustrated. In the first part of chapter 6, the elements for the way of working in NLM were defined, consisting of procedures or algorithms that specify how an analyst must carry out the requirements *elicitation* process in a dialogue with a knowledgeable domain user. Every procedure or algorithm contains built-in quality preserving and verification step(s) that verifies the recorded requirements segment (generally) in a dialogue with the domain user. In the second part of chapter 6 the elements in the way of controlling for NLM, were given in which the (project) management of the requirements determination process using the NLM method was illustrated.

In chapters 5 and 6 we have defined a requirements determination method that contains the necessary elements as they were laid down in 19 RMD's from chapter 4 and which therefore gives an answer to our main research question from chapter 1. The Natural Language Modeling (NLM) requirements determination method turns out to fulfill all necessary requirements for a to-be requirements determination method as was defined in chapter 1. In figure 7.1 we have shown how the research (sub)-questions are related.

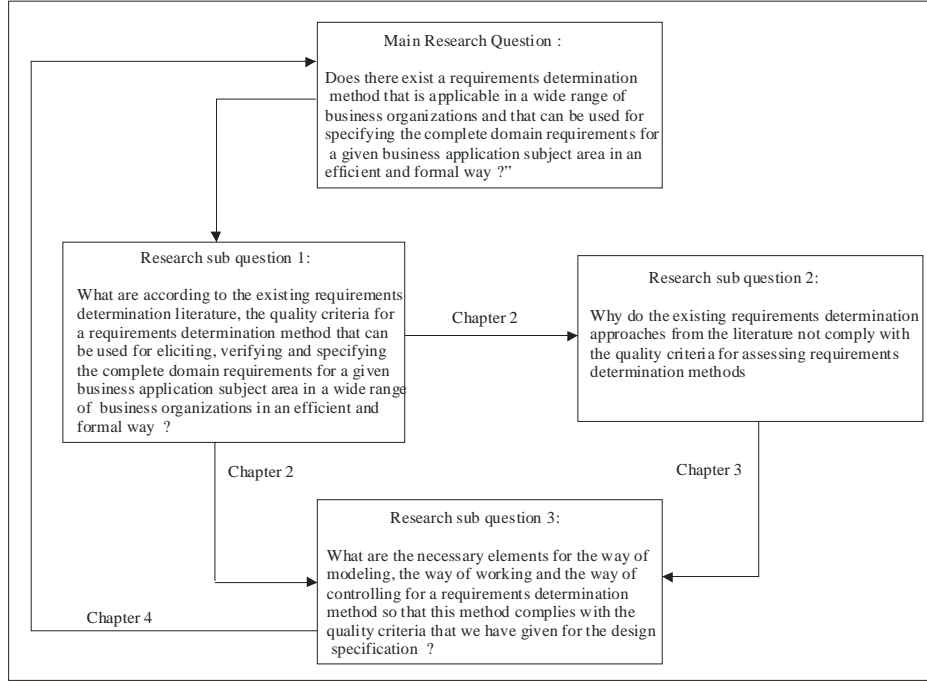


Fig. 7.1 Relationship between research (sub) questions)

The main findings regarding the extent in which NLM satisfies the 19 demands that we have derived are summarized here.

Findings for the way of modeling

The NLM requirements specification language contains only one information bearing construct: the fact type and it allows us to model any naming convention and semantic connection. The introduction of the sentence group template construct and the application concept repository allows us to capture the complete domain semantics of the UoD and therefore fulfills requirement *RMD 1*. The introduction of naming convention fact types and compound referencing schemes in combination with an accompanying sentence group template that enables us to capture the context in which the naming convention is valid fulfills requirement *RMD 2*. The definition and consistent application of the role construct and the mandatory naming convention from such a construct within the UoD of an analyst in the NLM specification language fulfills requirement *RMD 3*. Furthermore we have given modeling provisions that allows us to define any type of static constraint that currently exists within the EER and UML (compliance to *RMD 4*). It was also shown that NLM leads to requirements specifications that can easily evolve with changing application requirements (*RMD 5*). If instances of an intention can only exist on their own, this can be modeled as a unary fact type. This means that the NLM modeling constructs comply with requirement *RMD 6*. The definition of the uniqueness- and set-comparison constraints is fully

scalable as a function of the arity (N). This means that we have complied with requirement *RMD 7*. With respect to the transition constraints we remark that in our legend we have explicitly incorporated the relationship that the constraint has in terms of the values of the roles that are involved and it contains explicit references to before- and after- states of the application information base and therefore, NLM complies to *RMD 8*. The derivation rule constraints contain a reference to the roles from the Basic information Model of the UoD. This means that the derivation rule constraint that we have defined in the NLM's way of modeling complies with *RMD 9*. We have made a distinction into internal and external events in NLM. This leads to the compliance to *RMD 10*. In the impulse, an information base condition can be contained. Such an information base condition (IBC) is evaluated at some point in time. If the application information base at that point in time in combination with the information base condition yields the value true than the derivation rule and/or insert/delete operation will be executed. If it yields the value false nothing will happen. This means that requirement *RMD 11* has been fulfilled.

Findings for the way of working

The application of algorithms 1 through 4 will lead to the detection of all semantic relationships and naming conventions in the application subject area. We have specified static constraint derivation algorithms 6 and 7 to detect all uniqueness and set comparison constraints. In order to derive all instances of the dynamics constraints we have specified transition constraint derivation algorithm 8 as sub-procedure in NLM's way of working. In order to derive all instances of the derivation rule constraints we have specified the derivation rule constraint algorithm 9 in which the precise specification (or derivation formula) can be established. In algorithm 10 we have incorporated the question in which an internal event can lead to the execution of a derivation rule or another information base event. Furthermore, the algorithm systematically confronts the users in the SoI with derivation rules and tries to elicit the potential 'external' events that might invoke such a derivation rule. We can conclude that *RMD 12* has been fulfilled. In algorithm 5 a view integration algorithm has been defined. This fulfills *RMD 13*. The application of the natural language axiom in an organizational setting in which domain users are enabled to make implicit knowledge, explicit fulfills requirements *RMD 14*. The application of the natural language axiom in NLM also allows us to apply NLM in many organizational settings, ranging from abstract to tangible UoD's and from natural language descriptions to other descriptions that can only be understood by users. This leads to compliance to *RMD 15*. The subdivision of the modeling procedures in NLM's way of working into a number of formal algorithms has been done in such a way that the amount of analysis steps that have to be performed by (an) analyst(s) is minimized and therefore NLM fulfills *RMD 16*. The precise specification of the NLM modeling procedure in a number of algorithms with built-in formal quality assurance checks fulfills requirements *RMD 17*.

Findings for the way of controlling

The way of working in NLM has a work breakdown structure that consists of 10 activities or transformations that are laid down as formal algorithms and therefore can be formally planned as activities in a requirements determination project according to

RMD 18. Furthermore, NLM contains provisions that enable traceability in the requirements determination processes, by forcing an analyst to use naming conventions for the concept that he/she uses in the process of requirements determination and therefore NLM fulfills requirement *RMD 19*.

7.3 RESEARCH METHODOLOGY

In sections 5.10 and 6.10 we have already concluded that the NLM requirements determination method complied with all 19 requirements method demands RMD's that were specified in chapter 4. In this concluding chapter of this thesis we will reflect on the final stage from the 'design-research'-cycle: selection of the desired design from the set of alternative designs.

We can now conclude that NLM is a 'satisficing' solution to our main problem statement from chapter 1, since it 'satisfices' all 19 demands for a requirements determination method that were derived, based upon the literature research on the state-of-the-art in requirements determination methods. The research goal that was phrased in chapter 1: "to develop a method for requirements determination for which the way of modeling allows the analyst to capture all business entities and all business rules. This to-be developed RDM should have a way of working that contains modeling provisions that guide an analyst in eliciting the initial requirements from domain users. Finally, this method's way of controlling must contain quality preserving procedures that guarantees that a requirements specification that is the result of the application of this method have been validated by the user(s)" ..., therefore, is achieved, by developing an alternative design that complies to the requirements that were derived in chapter 4.

7.4 FUTURE MIS RESEARCH PROPOSALS

We will conclude this chapter with a number of topics for future research.

We have documented the NLM requirements determination method in chapters 5 and 6 of this thesis. The NLM requirements determination method clearly, provides a number of advantages over the 'state-of-the-art' in requirements determination methods. One of the most distinguishing features of NLM compared to for example, (E)ER, ORM or UML is the way in which the application of modeling constructs, for example, constraint types, is made explicit in a 'constraint'-legend and an accompanying 'instantiation algorithm'. In the appendix A to this thesis, we have provided the readers with some example constraint legends and in chapter 6 we have shown the accompanying instantiation algorithms for these constraint types. An agenda for future research, is to define more (in the sense of 'orthogonal' to the existing constraint types in this thesis) constraint types, that prove to be significant for business application subject areas, but above all, to develop accompanying instantiation

algorithms that can be used in an analyst-domain user dialogue, and that are based upon the acceptance and/or rejection of ‘real-life’ examples by knowledgeable domain users.

7.5 RECOMMENDATIONS FOR PRACTITIONERS IN THE MIS FIELD

We can conclude that the NLM requirements determination procedure explicitly shows the separation of concerns between the analyst and the user in the process of requirements modeling by providing the *semantic bridges* for this analyst-user dialogue. In addition to the creation of a NLM requirements specification that is an allowed extension of the NLM meta model we need guidance on what specific extension of this meta model reflects the domain semantics in a *precise* and a *complete* way. We have shown that such a semantic correct specification will be achieved when the algorithms that we have introduced in this thesis will be applied in a requirements determination project in which the sequence of their application is performed under the precedence requirements that were given in chapter 6. Furthermore, this will result in the most *efficient* way of working. Although a number of procedures in the NLM requirements determination method at first sight have a ‘trivial’ appearance, the consistent application of the procedures in the ‘way-of-working’ in this thesis in practice has proven to improve the ‘quality’ levels of the resulting information models, because even the experienced analyst can always ‘fall back’ on the procedure in those situations in which the application subject area becomes too complex. Another advantage is that inexperienced analysts will be able to create requirements specifications that have the same quality level as the specifications that are created by experienced analysts. In a project in which the NLM requirements determination method is applied for the creation of a requirements specification, the division into sub-projects and the order in which these sub-projects are executed does not have an impact on the final specification.

As we pointed out earlier, the objective of this thesis research was to develop a requirements modeling language and a modeling procedure, rather than to specify an ‘optimal’ notation legend for such a language. Practitioners, however, need to be able to communicate, with domain users, management and peers in many cases using a pre-defined diagramming technique or notational legend. The conclusions from this research in terms of giving a preference to a modeling language that has *one* information bearing construct, that has uniform modeling facility for naming conventions and that has a number of *orthogonal constraint types*. The definition of these constraint types in combination with an instantiation procedure has a built-in guarantee that these constraint instances can always be derived in any application UoD, whenever the appropriate instantiation procedure is applied in combination with a knowledgeable domain user. The practitioner should evaluate the requirements specification/determination approach that he/she is currently using. After this evaluation the practitioners can decide to limit or redefine the modeling constructs that they want to keep and define new modeling constructs if one more necessary constructs are missing. In a second stage a ‘notational’ legend must be (re)defined that preferably

is 'backwards compatible' with the old way of modeling and the old way of working. In the third stage of the evaluation of the current approach, practitioners must decide on what types of constraints are relevant for the specific type(s) of application domain(s) in which the requirements determination method is going to be applied. We emphasize that for these constraint types, accompanying procedures must be specified on such a level of concreteness that an analyst can apply these procedures in a dialogue with a knowledgeable user.

7.5.1 Application of NLM in practice

The NLM requirements determination that we have documented in chapters 5 and 6 has been applied by master students in MIS a number of times in large and small enterprises, see for example Bogget (1994) and Enter (1999). Other students have applied this approach on object-oriented models (Clayes, 1996) and on the accounting knowledge domain (Wolthuis, 1997).

7.6 CONCLUDING REMARKS

In this thesis we have studied the field of requirements determination for enterprise information systems. We discovered that in the ERP era that characterizes the information systems in many (large) enterprises at the beginning of the 21st century, the issue of requirements determination is still relevant. We also discovered that the 'state-of-the-art' of this field still shows a number of omissions in the definition of requirements specifications modeling constructs and methodology. We have chosen to specify 'the requirements' or demands for a requirements method itself using four (groups of) criteria. These criteria were operationalized by studying the three most dominant requirements determination approaches that exist today. These criteria were subsequently translated into 19 specific demands (RMD's) for a to be designed requirements determination method. In the second part of this thesis (chapters 5 and 6) we have introduced the Natural Language Modeling (NLM) approach for requirements determination. It turns out that NLM satisfies all 19 requirements and therefore can be considered an appropriate design alternative, as is specified in the design research cycle (Van Engelen and Van der Zwaan, 1994). Hence the choice of Natural Language Modeling as answer for our research objective is justified.

7.7 REFERENCES

- Bogget, M. (1994): Implementation of a Management Reporting System & Preparation of MRP model at PTZ Nelahiozeves Unilever. Final Thesis Business Economics. University of Maastricht
- Bubenko, J., Wangler, B. (1992): Research Directions in conceptual specification developments. In: Conceptual Modeling, Databases, and Case (Loucopoulos, P., Zicari, R. (eds.)). Wiley. 389-412
- Clayes, C. (1996): Final thesis. International business studies. University of Maastricht.
- Dullea, J., Song, I-Y., Lamprou, I. (2003): An analysis of structural validity in entity-relationship modeling. *Data & Knowledge Engineering* **47**: 167-205
- Enter, N. (1999): The semantics of the CIC SAP R/3 core. Final thesis. International business studies. University of Maastricht.
- Flynn, D., Warhurst, R. (1994): An empirical study of the validation process within requirements determination. *Information Systems Journal* **4**: 185-212
- Galliers, R., Swan, J. (2000): There's more to information systems development than structured approaches: information requirements analysis as a socially mediated process. *Requirements Engineering* **5**(2): 74-82
- Goldin, L., Berry, D. (1997): Abstfinder, a prototype natural language Text Abstraction Finder for Use in Requirements Elicitation. *Automated Software Engineering* **4**: 375-412
- Land, F. (1998): A Contingency Based Approach to Requirements Elicitation and Systems Development. *Journal of Systems and Software* **40**: 3-6
- Mantel, S., Meredith, J., Shafer, S., Sutton, M. (2001): Project management in practice. Wiley and Sons
- Van Engelen, J., Van der Zwaan, A. (1994): Bedrijfskundige methodologie 2: een technisch-methodologische context. *Bedrijfskunde* **66** (2): 85-94 (in Dutch)
- Wolthuis, M. (1997): Afstudeerscriptie: Boekhouden op basis van Universele Informatiekunde. University of Limburg. (in dutch)

APPENDIX A:

THE SPECIFICATION OF THE CONSTRAINT TYPES IN THE NLM REQUIREMENTS SPECIFICATION LANGUAGE

A.1 INTRODUCTION

In this appendix we will zoom in on the modeling constructs that will enable us to express that some extensions of a basic information model are not allowed to exist. We will use part 2 of the university enrollment example in chapter 5 to illustrate the different constraint types.

A.2 POPULATION STATE CONSTRAINTS

We can consider a population state as a further reduction of the extensions in the set of possible extensions of a basic information model. After the restriction of the names to the name classes, that can be used to identify a specific thing, entity or concept in the application UoD, we will further restrict the extensions that are allowed to exist by incorporating specific domain knowledge or those domain rules (or business rules) that can be expressed as propositions on the basic information model and that must be true in every population state. We will call such a proposition a *population state constraint*.

Definition A.1 (=5.11). A *population state constraint* p in a basic information model BIM is a proposition that limits the allowed extensions of the basic information model BIM to those extensions for which the proposition of p is true.

A population state constraint is a set valued function into the set of extensions of a basic information model of a universe of discourse.

$$PC: \quad \{EXT_j(BIM)\} \text{ ----> } \{EXT_j(BIM)\}$$

Example 5.1: University Enrollment part 1 (ctd.)

BIM={FT1, FT2, STUDENTNAME, STUDENTMAJOR}

Business rule: a student must be enrolled in at most one major at a time.

Domain extensions

Range extensions

{student V1234 majors in major science}; {student V1234 majors in major science}
 { student V1234 majors in major science,
 student V1234 majors in major economics}

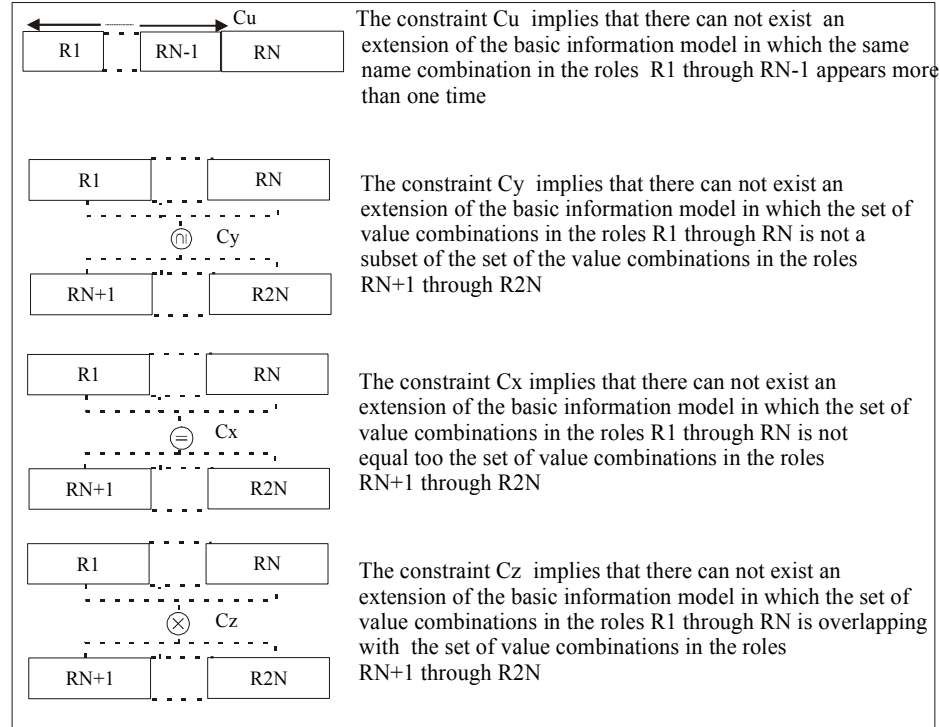


Fig. A.1 Example legend for uniqueness-, exclusion-, subset- and equality-⁶⁶ population state constraints.

A.3 POPULATION STATE TRANSITION CONSTRAINTS

The business rule: *a student can be enrolled in at most one major at a time*, can be expressed as the following constraint instance from the constraint legend in figure A.1⁶⁷:

⁶⁶ Exclusion constraints, equality constraints and subset constraints together are referred to as set comparison constraints (Leung and Nijssen, 1988:35)

⁶⁷ We note that this sentence refers to the Universe of Discourse of an information analyst, who has knowledge of NLM and has access to an accompanying constraint legend.

The constraint c1 implies that there can not exist an extension of the basic information model in which the same name in the role enrolled student appears more than one time.

If we inspect this example we can conclude that the addition of a population state constraint onto a (basic) information model actually eliminates those extensions from the set of extensions that do not comply with the proposition. In this example we have shown that the example extension: {student V1234 majors in major science, student V1234 majors in major economics} does not comply with the proposition of population constraint *c1*.

The population state of a basic information model (BIM) will change over time because fact instances can be added or removed from the application's information base at any time. We will define the addition or deletion of fact instances to or from the application's information base a state transition from a *population before* (the addition or deletion has taken place) to a *population after* (the addition or deletion has taken place)

Definition A.2 (=5.12). A *population state transition constraint* *q* in a basic information model BIM is a proposition that limits the before-after extension combinations of the basic information model BIM to those combinations for which the proposition of *q* is true.

A population state transition constraint is a set valued function into the set of before-after extensions of a basic information model of a universe of discourse.

$$PTC: \quad \{EXT_j(BIM)\} \times \{EXT_j(BIM)\} \rightarrow \{EXT_j(BIM)\} \times \{EXT_j(BIM)\}$$

In the student enrollment example we have illustrated how population state constraint *c1* will guarantee that only those extensions of the BIM can potentially exist as population states that are allowed with respect to the business rule *that a student can be enrolled in at most one major*. In some UoD's business rules might exist that can not exclusively be modelled as (a combination of) population state constraints. Consider the following business rule in the university enrollment UoD: *A student can not major in Economics after he/she has majored in Science*. This business rule prohibits for example the following before-after combination of fact type extensions: {student V1234 majors in major science, student V1234 majors in major economics} although this before/after extension combination is allowed according to constraint *c1* and , therefore, must be encoded as a population transition constraint *c14*:

The population constraint c14 implies that there can not exist a before/after combination of extensions of the basic information model in which the extension for the role chosen major for a given student in the before state is equal to 'science' and in the after state is equal to 'economics'.

The addition of the population state transition constraints to the basic information model reduces the set of allowed before-after extensions in the UoD. We have provided a legend for the state transition constraints in figure A.2.

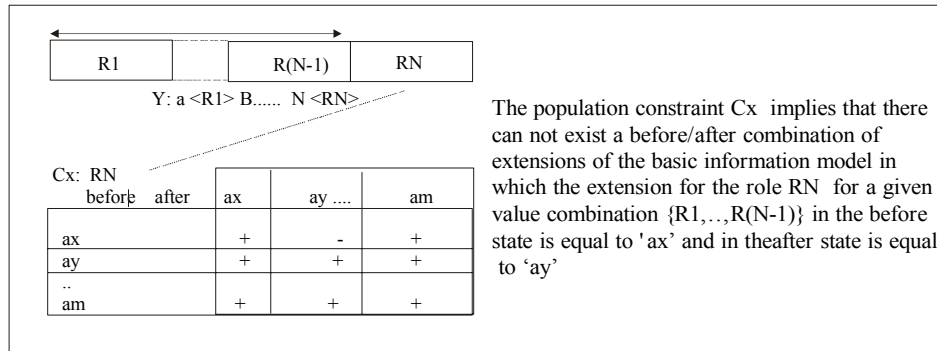


Fig. A.2 Example legend for population state transition constraints.

A.4 DERIVATION RULE CONSTRAINTS

In addition to the population constraints, constraints can exist between the values that particular fact instances must hold when other fact instances are given. For example, in part 2 of our University Enrollment, if we want to record the total number of credits for a person in his/her freshmen year (fact type *FT13*) in addition to the number of credits for every individual course (fact type *FT12*) we know that whenever instances of *FT12* are known, the instances of *FT13* can be 'computed'. If instances of fact type *FT13* are known, however we can not infer or 'compute' the instances of *FT12*. This means that there exists a type of constraint that determines a specific fact instance when instances of other fact types are known to exist. This type of constraint we will call a *derivation rule*.

In this section we will define the modeling constructs that will allow us to precisely specify derivation rules on fact types in the basic information model (defined in section 5.7). Derivation rules specify how fact instances of a given fact type in the BIM are composed of fact instances of other fact types in the application's basic information model.

Example 5.1 part 2 (ctd):

Consider the following fact instance on the example form from figure 5.5:

Fact 1:

The total numbers of credits for the student O 5678 in his/her freshmen year was 24

It may be created in the following way:

The total number of credits for student O 5678 is created by adding all the credits for the courses for student O 5678.

using the following ingredient facts:

Fact 2: Student O 5678 earned 8 credits for the course macro economics.
Fact 3: Student O 5678 earned 8 credits for the course micro economics.
Fact 4: Student O 5678 earned 8 credits for the course finance.

The creation of fact instance *fact 1* is a function defined on the *ingredient fact instances fact 2, fact 3 and fact 4*. The fact type(s) of the fact instances created in (an) instance(s) a derivation rule will be referred to as the *resulting fact type(s)* for the derivation rule. An (the) *ingredient fact type(s)* of a derivation rule specifies what the fact instances serve as an input for the derivation of a fact in a derivation rule.

Definition A.3 (=5.13). A derivation rule (constraint) is a function defined on instances of the ingredient fact types. The function range is a set of resulting fact instance(s) from the resulting fact types of the derivation rule.

Let FT_1 through FT_N be ingredient fact types for the derivation rule CP
Let FT_M be the resulting fact type for the derivation rule CP

$CP: EXT(FT_1) \times \dots \times EXT(FT_N) \text{ -----} \rightarrow EXT(FT_M)$

Example 5.1 part 2 (continued):

Ingredient fact type FT12: *Student [identified by the combination of <university code> and <student ID>] gained a number of credits <course credits> for the course <credited course>.*

Resulting fact type FT 13: *Student [identified by the combination of <university code> and <student ID>] gained a total number of credits <total credits> in his/her freshman year*

Function DF_1 : $FT13.<total credits>(FT13.<university code>.<student ID>):=$

$$\sum_{j^{68}=1}^M FT12.<course credits> [FT12.<university code>.<student ID>=FT13.<university code>.<student ID>]$$

The specification of a *derivation rule* can be considered another semantic bridge in the *natural language modeling requirements determination method*. In this process the variables in the function formula are assigned specific semantics in terms of roles of the basic information model (BIM) of the UoD. The parameters that tell us

⁶⁸ Where j is the index on the instances of courses.

what fact instances will be the 'tangible' end results of the execution of a derivation rule and what other factors constrain the possible outcomes of such a derivation. We will call such a set of parameters: the *derivation rule argument*⁶⁹.

Definition A.4. A *derivation rule argument* specifies the types of values that must be specified for the creation of a *derived fact instance*.

Let CP be a derivation rule
Let FT_M be the resulting fact type for the derivation rule CP
Let dfi be a derived fact instance for CP: dfi ∈ EXT(FT_M)
Let drarg be a derivation rule argument.

$$CP: EXT(FT_1) \times \dots \times EXT(FT_N) \times DOM(drarg) \text{ -----} \rightarrow EXT(FT_M)$$

Example 5.1 part 2 (ctd.):

*Derivation rule constraint c15: **create total number of course credits***
The instances of this derivation rule create instances of fact type FT13
Derivation rule argument: {(arg₁, student)}

We note that derivation rules can exist in which a derivation rule argument does not exist (see the next example).

Example 5.1 part 2 (ctd.):

*Derivation rule constraint c16: **create total number of enrolled students at vandover and Ohoa university combined***
The instances of this derivation rule create instances of fact type FT14 and for this derivation rule no derivation rule argument exists.

⁶⁹ In some application UoD's, the derivation rule arguments and derived fact types can change depending upon the specific set of argument instances that are given. For example in a project management UoD three fact types might exist: FT1: <activity> starts on <time>, FT2: <activity> ends on <time>, FT3: <activity> has <duration>. Sometimes FT3 will be derived based upon FT1 and FT2 as (given) arguments. In other instances FT2 will be derived based upon FT1 and FT3 as arguments and in other instances FT1 will be derived based upon FT2 and FT3 as arguments. In this example, therefore, it is not known in advance precisely which 2 fact instances of the three are known. In this example, therefore, three derivation rules must be created. The modeling provision that allows us to specify when which derivation rule will be executed is the specific event that initiates the triggering of the derivation rule. If the event argument contains instances of FT1 and FT2 then the derivation rule that derives an instance of FT3 will be triggered. This will be modeled in three impulses. See the remainder of this Appendix.

Definition A.5. An *information base condition* (IBC) is a proposition on the information base.

Let $EXT(BIM)$ *be an information base*
Let IBC *be an information base condition*
 $IBC = prop(EXT(BIM))$

Example 5.1 part 2 (ctd):

$ibc_1: \exists f \in EXT(FT12) [f.<student ID>.<University code> = 'O 5678']$

Definition A.6. An *information base condition type* (IBCT) is a set of propositions defined on the information base. An instance of an *information base condition type* is an *information base condition*.

Let $EXT(BIM)$ *be an information base*
Let IBC *be an information base condition*
Let $IBCT$ *be an information base condition type*
 $EXT(IBCT) = \{IBC\}$

An information base condition type in a derivation rule can be instantiated into an information base condition whenever the instantiation values of the derivation rule argument are known.

$DR1: \text{Create total number of credits (arg: student)}$
 $CT: \exists x \in EXT(R_2) [x = DR1.arg]$

If we consider the derivation rule: *create total number of credits* it will only then create (a) fact instance(s) of fact type FT13 when at least one fact instance of fact type FT12 exists in the *application information base*. If we inspect the derivation rule and the instantiation values for the *derivation rule argument* it should be clear whether the execution of the process will lead to a result **before** the derivation rule is actually executed. The *pre-condition* serves as this checking mechanism for the instantiation of a derivation rule. If the *pre-condition* is violated by the actual content of the application information base, (a) derived fact instance(s) will *not* be created.

Definition A.7. A *precondition* in a derivation rule is an *information base condition (type)* that checks whether the required *input fact instances* and derivation rule argument values for the *derivation rule* exist in the *application information base*.

Let $EXT(BIM)$ *be an information base*
Let IBC *be an information base condition*
Let PC *be a precondition*
 $PC \in \{IBC \mid IBC = prop(EXT(BIM))\}$

Example 5.1 part 2 (ctd):

DR₁<{(arg₁,student)}>

IF there exist an instance of FT12

SUCH THAT FT12.<university code>.<student ID>=arg1 {pre-condition}

The post-condition specifies what the fact argument is for the facts that will be created as a result of the execution of the derivation rule. Furthermore, a reference is given on how the fact values for the roles in the facts that will be created in the derivation rule will be obtained. This post-condition should, furthermore, specify what fact instances (in terms of the derivation rule) argument should be instantiated for the resulting fact type(s) when the derivation rule is executed.

Definition A.8. A post-condition specifies (parts of) the fact argument for the instances of the resulting fact type(s) that must be created in the derivation rule.

Let EXT(BIM) be an information base
Let IBC be an information base condition
Let PO be a postcondition
 $PO \in \{IBC \mid IBC = \text{prop}(EXT(BIM))\}$

Example 5.1 part 2 (continued):

C15: Create total number of credits<{(arg₁,student)}>

IF there exist an instance of FT12

SUCH THAT FT12.<university code>.<student ID>=arg1 {pre-condition}

THEN create an instance of fact type FT13

SUCH THAT

FT13.<university code>.<student ID>:= arg1 {this is the fact argument}

FT13.<total credits>:=DF1 {post condition}

DF1:= Σ FT12.<credits> [where FT12.<university code>.<student ID>='arg1']

ENDIF {derivation formula}

C16: Create total number of enrolled students

IF there exist an instance of FT10

THEN create an instance of fact type FT14

SUCH THAT

FT14.<total enrolled students>:=DF2 {post condition}

DF2:= COUNT(Ext(FT10)) {derivation formula}

ENDIF

In figure A.3 an example of a (verbalization) legend for the derivation rule constraint is given.

```

DRX<{derivation rule argument argDRX}>
  IF      {pre-condition Prx1}
  AND ...
  AND    {pre-condition PrxN}
  THEN create an instance of resulting fact type FTX
  SUCH THAT
    {post condition Pox}
    {derivation formula Dx}
  ENDIF

```

The constraint *DRX* implies that whenever the values for the derivation rule argument *argDRX* are known and fulfill the pre-condition '*Prx1*' and.....and '*PrxN*' then (an) instance(s) of fact type *Ftx* will be created in which the fact type argument complies to the post condition '*Pox*' and the formula '*Dx*' will be used to derive the fact value

Fig. A.3 Example legend for derivation rule constraint.

A.5 IMPULSE CONSTRAINTS

In the information systems literature numerous definitions of the *event* concept can be found: “*An event is an occurrence or happening of something in the environment under consideration.*” (De and Sen, 1984:182). “*An event is a noteworthy change of state; all the changes of state of objects are not events.*”(Rolland, 1983:34). We will give the following definitions of *event occurrence* and *event* :

Definition A.9(=5.14). An *event occurrence* is a happening at a certain point in time in the application subject area that can lead to the execution of one or more derivation rules and/or the insertion or deletion of fact instances into/from the application's information base.

Let PH be the set of potential happenings
Let eo be an event occurrence
 $eo \in PH$

From definition A.9 it follows that an event occurrence is a 'one-time' only thing. For example the event occurrence: *student 'V 2345' wants to enroll for major 'science' at 12:45:56 on 01/12/2004*. A different event occurrence is: *student 'V 2345' wants to enroll for major 'science' at 18:45:56 on 03/06/04*. We can group the former two event occurrences into the following event: *student 'V 2345' wants to enroll for major 'science'*.

Definition A.10 (=5.15). An *event* is one or a number of potential happenings in the application subject area that can lead to the execution of one or more derivation rules and/or the insertion or deletion of fact instances into/from the application's information base.

Let PH be the set of potential happenings
Let e be an event
 $e \subset PH$

Examples:

Student requests enrollment for major
Student has earned credits for a course

If we take the University Enrollment example, we can *qualify* the example event from the University Enrollment example: *Student requests enrollment for major* into the following event: *Student V 5463 requests enrollment for Major economics*. If we observe this Universe of Discourse over a certain period of time we can encounter also the following event instances *Student O7564 requests enrollment for major Economics*, *Student with V 4467 requests enrollment for major law*. We can conclude that the former verbalization of events can be further grouped and qualified into the *event type* having an event argument:

Student wants to enroll in major(arg1: student, arg2: major)

Definition A.11(=5.16.) An *event type* is a class of events in the application subject area, each of these events can lead to the execution of one or more derivation rules (of the same type) and/or the insertion or deletion of fact instances (of the same fact types(s)) into/from the application's information base.

Let ET be an event type
Let $E = \{e_i\}$ be the set of events
 $ET \subset E$

Example 5.1 part 2 (ctd.):

Consider following event set E_1 .

$E_1 = \{ \text{Student O7564 requests enrollment for major Economics, Student with V 4467 requests enrollment for major law, Student V 3456 has earned 7 credits for the course behavioral finance} \}$

The event type $ET_1 = \{ \text{Student O7564 requests enrollment for major Economics, Student with V 4467 requests enrollment for major law} \}$.

All events of an event type, therefore, in addition must have the same intentions in the event argument for a universe of discourse U .

Once we have found an event type argument we can add the definition of the concepts that underlies this argument in the list of definitions or *application concept repository* in case it is not yet contained in the ACR.

Definition A.12. An *event type argument* of a given event type specifies all intentions, instances of which must be known at the occurrence of an event instance of the event type.

Let ET be an event type and b be an event type argument then $ET\langle b \rangle$ is the event type ET with event type argument set b .

Example 5.1 part 2 ctd.):

Consider the following event type: $ET1$: *Student wants to enroll in Major* (*arg1*: *student*; *arg2*: *major*)

An instance of this event type is: *Student wants to enroll in major* (*arg1*: 'issn 5678, *arg2*: 'Science')

The derivation rule or insert/delete operation that must be instantiated as a result of this event is the insertion of an instance of fact type $Ft11$. The instances of the intentions in the argument for the event type can be used for instantiating the derivation rule or insert/delete operation (in an *impulse*). The modeling construct of *event* refers to an action that can occur, for example: *student graduates* or it can refer to a more 'static' action, for example the start of a new day when *the clock strikes 12:00 P.M.* We can conclude that events can have different appearances and therefore we will use the *basic information model* and the *derivation rules* in combination as a starting point for 'detecting' events that are relevant for the application subject area. A significant source for potential events is the state change in the application information base (Prabhakaran and Falkenberg, 1988) or database events (Chakravarthy and Mishra, 1994:2).

The derivation and the specification of the event type out of a significant set of event instances (or event sentences) follows the same procedure as the *grouping, qualification and classification* of the user verbalized sentences in the information perspective. The significant difference between the event types in the event perspective and the fact types in the information perspective lies in the *Universe of Discourse* to which they refer. In the information perspective this UoD consist of user examples of *declarative* information. Subsequently these examples are verbalized by the user. The 'UoD' in the event perspective is less tangible and in general can not be traced back as user-example that contains declarative information (see the discussion regarding *status data* versus *event data* in (McFadden et. al, 1994:539)).

A.5.1 Impulse and impulse type

An *event* can start the execution of a derivation rule (in some cases) under (a) condition(s) on the information base (Dayal et al., 1990:106). In the population constraints from the application requirements specification we have modeled the business rules in terms of the state and state transition of the application information base. In the pre-conditions of the derivation rules, the business rules are modeled that

specify what ingredient fact instances should be available in order to ‘compose’ or ‘derive’ the derived fact instance(s). In the event perspective we will model the business rules that contain the knowledge under what condition (on the state of the application information base (Chakravarthy and Mishra, 1994:5)) an event of an event type will trigger a derivation rule or an information base insertion and/or deletion.

Example 5.1 part 2 (ctd.):

Suppose that in the (integrated) University Enrollment UoD the total number of enrolled students must always be up-to-date and be available on the example document from figure 5.5. This means that there exist two event types, that must lead to the (re)calculation of the total number of enrolled students: the event type that designates that a new student is enrolled and an event type that designates that a student has graduated. The *impulse type constraints* for this example will be the following:

C17

ON ET2: Insert (Student ‘x’ wants to enroll in Major ‘y’) into application data
base has succeeded (arg1: ‘x’; arg 2: ‘y’)
DO Create total number of enrolled students

C18

ON ET3: Delete (Student ‘x’ wants to enroll in Major ‘y’) from application data
base has succeeded (arg1: ‘x’; arg 2: ‘y’)
DO Create total number of enrolled students

We can see that there exist two different types of events that can lead to the execution of (different) instances of the same derivation rules (in this case *create total number of enrolled students*). In this example events of two **different** event types will lead to the triggering of the **same** derivation rule (see figure A.4).

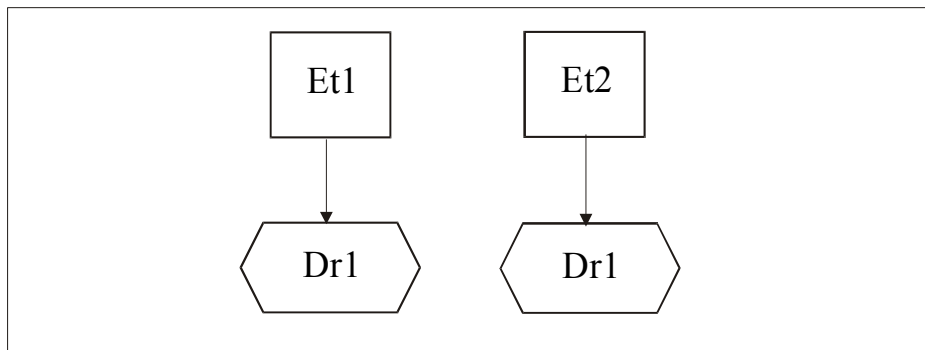


Fig. A.4 Two different event types that trigger the same derivation rule.

On the other hand it is possible that two or more different derivation rules exist in the application subject area that derive (different) instances of the same fact type.

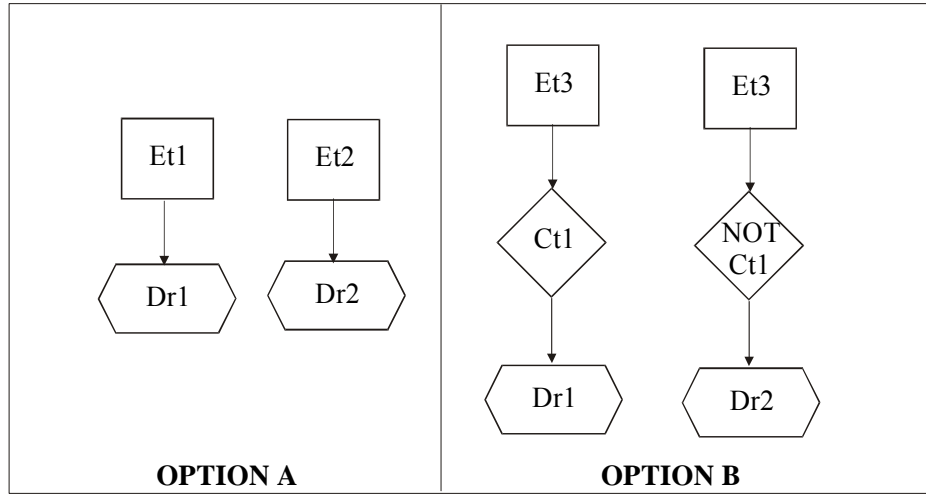


Fig. A.5 Two derivation rules that create instances of the same fact type.

In order to be able to decide which derivation rule will be used for the creation of a specific fact instance we either need two *different* event types that unconditionally instantiate and trigger these two process types on the discretion of the domain user (option *A* in figure A.5) **or** we need one event type and a condition that tells us under what condition on the information base the derivation rule *DR1* will be used or derivation rule *DR2* will be used (option *B* in figure A.5). The reason for the existence of these two sets of impulse types is that each user group within the application area exclusively has knowledge of the fact creation processes that are executed by that user group. If all user groups would have knowledge of all business rules in the integrated application subject area then the condition *CT1* most likely would be incorporated into the description of one single derivation rule.

An *impulse condition* is an *information base condition* for the execution of a derivation rule or information base insertion/deletion that is ‘triggered’ by a specific event. In option B from the application event description in figure A.5 we see that when an event instance of event type *ET3* occurs and the proposition in condition *CT1* evaluates to true then derivation rule *DR1* will be instantiated. If the proposition in condition *CT1* evaluates to false then a derivation rule *DR2* will be instantiated. In addition to the conditions that are given in the *pre-condition* of the derivation rule or a condition that is enforced by the *population constraints* in the application information grammar, the *information base condition* that is specified in the *impulse* is defined in terms of a proposition on the *information base state* at that moment in relative time in which the *information base condition* is checked (see figure A.6).

Definition A.13. An *impulse* is the occurrence of an event leading to the instantiation of a derivation rule and or insert/delete process, eventually under some condition on the application’s information base.

The abstraction of a set of impulses that have events of the same event type, have conditions of the same condition types and trigger the same derivation rule (constraints) we will call an impulse type (constraint).

Definition A.14 (=5.17). An *impulse type (constraint)* is an ordered triplet that contains an event type, a condition type under which an event occurrence of an event of a given event type can lead to the execution, of a specified derivation rule constraint or insert/delete operation.^{70,71}

Let IT be an impulse type

Let SET be the set of event types

Let SCT be the set of condition types

Let SDR be the union of the set of derivation rule constraints and the set of insert/delete operations

$$IT = (A, B, C) \mid A \in SET, B \in SCT, C \in SDR\}$$

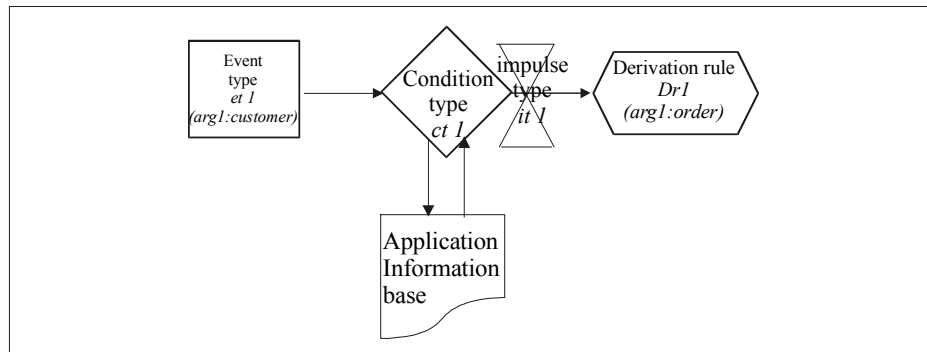


Fig. A.6 Event triggering a derivation rule when a condition is satisfied (impulse).

⁷⁰ In this thesis we have defined an impulse type as a triplet. We thereby assume that there exist some form of ‘event-processor’ that is able to process the occurrence of two events simultaneously and also is able to invoke two different derivation rules and/or insert/delete operations that must be excuted in parallel in which the sequence in which derivation rules and/or impulse/delete operations are executed is determined by a prioritizer that checks the pre-conditions of these derivation rules. An additional complicating factor is the situation in which the joint occurrence of two events of different event types will lead to execution of a possibly different derivation rule and/or insert/delete operation under a possibly different condition (type) than the condition types that are given in the two ‘individual’ impulse types. This complication can be circumvented by labelling the joint occurrence of these two events as a different event and subsequently consider it as a different impulse type.

⁷¹ In this thesis we will only consider elementary updates: adding or deleting a fact (we have called this insert and delete operations) in line with Nijssen and Halpin (1989:17). This means that for these operations we will allow a ‘compound transaction’ in the impulse in which an ‘old’ fact will be deleted and a ‘new’ fact will be inserted.

Example 5.1 part 2 (ctd.):

Event type ET1 *student requests enrollment in major* (arg1: student, arg2: major)
Event instance *student requests enrollment in major* (arg1: "O 7689";
arg2: Economics)

Under the condition that the number of credits that the student has gained in his/her freshmen year is greater or equal than 24 and the student has at least 8 credits for the course macro economics this event should lead to the instantiation of the following fact insertion rule: *Insert (student<university code><student ID> has chosen major <chosen major>)*. The occurrence of this event instance will lead to the following instantiation of this fact insertion rule: *Insert (student 'O 7689' has chosen major 'economics')*. A different event occurrence of the same type is *student requests enrollment in major* (arg1: "O 7689"; arg2: "Psychology"). Given the fact that this student requests to be enrolled in a major that does not exist within the Ohoodover University, the event occurrence will not lead to the instantiation of an insertion rule.

C19

ON ET1: student requests enrollment in major (arg1: student, arg2: major)
IF [FT13.<total credits>
(Where FT13.<university code>.<Student.ID>='ET1.arg1')] > 24
AND [IF ET1.arg2='science' THEN(mathematics ∈ EXT (FT12.<credited
course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND
FT12.<course credits>[where FT12.<university code> .
<Student.ID> = 'ET1.arg1' AND where FT12.<credited course >
= 'mathematics']>8)
OR⁷²
[IF ET1.arg2='history' THEN (language and culture ∈ EXT (FT12.<credited
course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND
FT12.<course credits>[where FT12.<university code> .<Student.ID>
= 'ET1.arg1' AND where FT12.<credited course>= 'language & culture']>5)
OR
[IF ET1.arg2='economics' THEN(macro economics ∈ EXT (FT12.<credited
course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND
FT12.<course credits>[where FT12.<university code> .
<Student.ID> = 'ET1.arg1' AND where FT12.<credited course >
= 'macro econ.']>8)
OR
[IF ET1.arg2='medicine' THEN (biology ∈ EXT (FT12.<credited
course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND
FT12.<course credits>[where FT12.<university code> .
<Student.ID> = 'ET1.arg1' AND where FT12.<credited course >
= 'biology']>5)
OR

⁷² In the sense of an exclusive OR


```
[ IF ET1.arg2='law' THEN (finance ∈ EXT (FT12.<credited
course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND
FT12.<course credits>[where FT12.<university code> .
<Student.ID> = 'ET1.arg1' AND where FT12.<credited course >
= 'finance' ]>5) ]
DO Insert (student'Et1.arg1' has chosen major 'ET1.arg2').
```

An example legend for the *impulse constraint* is given in figure A.7.

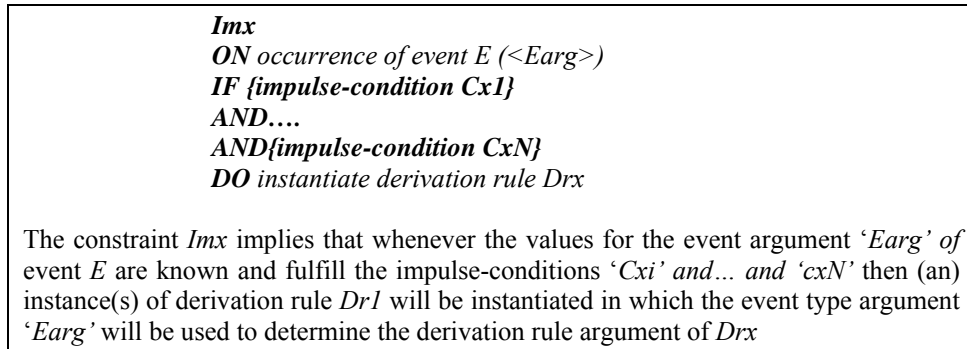


Fig. A.7 Example legend type constraints.

So far, we have assumed that the occurrence of the event, the check of the impulse-condition and the execution of a (the) derivation rule(s) will take place instantaneously. In many real-life business UoD's some form of delay or date-constraint exists that puts a temporal constraint on the event-occurrence, the check of the impulse condition and the execution of the derivation rules (and/or insertion/deletion rules). Consider the diagram in figure A.8.

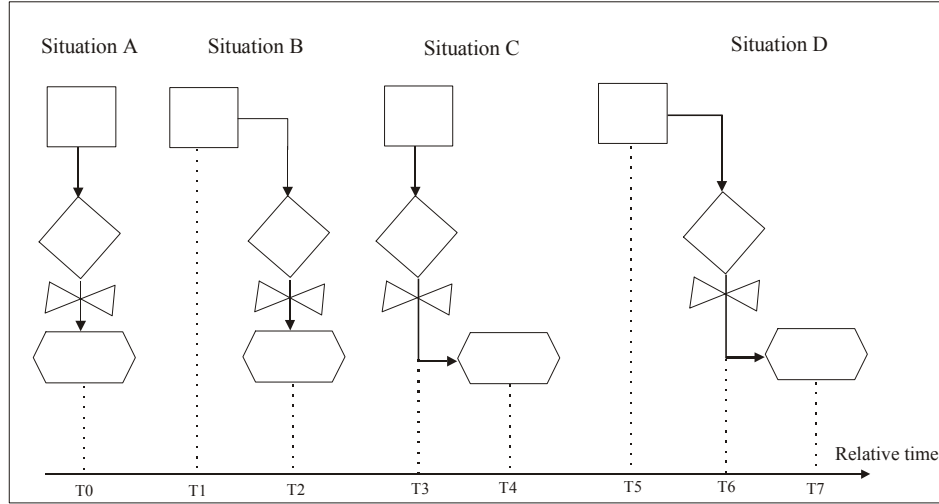


Fig. A.8 The possible temporal characteristic of impulse(s) (types)

In the University enrollment example it could be decided to perform the check on the major availability not on a first come, first served basis but to accumulate the requests on a monthly or bi-monthly base to make up for time-preferences but to evaluate the requests on additional student criteria (for example average grades for freshman courses) if the aggregate demand for enrollments exceeds the capacity at that point in time. In this case we need to qualify the existence of an impulse with a construct that controls the moment in which a condition is checked whenever an event takes place. We also need to define the moment in relative time in which a conceptual process is executed whenever the impulse condition evaluates to true. We will have to introduce the concept of (relative) time. In figure A.8 we have illustrated all the situations that can exist for the moments in relative time in which an event can occur, subsequently a condition can be checked and finally a derivation rule or insert/delete operation can be executed. In situation A in figure A.8 the occurrence of an event, the condition check and the (potential) execution of the conceptual process will take place at the same moment t_0 in relative time. In situation B we see that an event occurs at relative time t_1 where the condition check in the impulse and the potential execution of the conceptual process in the impulse will take place at t_2 ($>t_1$). In situation C the occurrence of the event and the condition check in the impulse will take place at the same moment in relative time t_3 . The (potential) execution of the conceptual process will take place at t_4 ($>t_3$). Finally in situation D we have the general case where we see the event occurrence at relative time t_5 . The condition in the impulse will be checked in t_6 ($>t_5$) and the (potential) execution of the conceptual process will take place at relative time t_7 ($>t_6$). We can conclude now that in addition to the concepts of event (type), information base condition (type) and impulse (type) we need two time variables that can encode the potential time delays between on the one hand the occurrence of an event and the check on the condition in the impulse and on the other hand the check on

the condition and the execution (if any) of the conceptual process in the impulse. If these time variables are necessary in order to be able to model the dynamic constraints that exist in an application UoD, the impulse type legend in figure A.7 can be adapted accordingly.

Example 5.1 University Enrollment part 2 (ctd.):

We will now give the remaining impulse types for our University Enrollment example

C20

ON ET4: Insert (Student 'x' has gained the number of 'y' course credits for course 'z') into application data base has succeeded (arg1: 'x';
arg 2: 'y'; arg3: 'z')
DO Create total number of credits (arg1:= 'Et4.arg1')

C21

ON ET5: Credits granted to student (arg1: student; arg2:course; arg3:credits)
IF ET5.arg1 ∈ EXT (FT11.<Universitycode>.<student ID>)
DO Insert (Student 'Et5.arg1' has gained the number of 'Et5.arg2' course credits for course 'Et5.arg3')

C22

ON ET6: Student graduates (arg1:student)
IF ET6.arg1 ∈ EXT (FT11.<Universitycode>.<student ID>)
DO Delete (Student 'Et6.arg1' wants to enroll in Major 'y')

C23

ON ET1: student requests enrollment in major (arg1: student, arg2:major)
IF [FT13.<total credits>
(Where FT13.<university code>.<Student.ID>= 'ET1.arg1')] > 24
AND [IF ET1.arg2= 'science' THEN (mathematics ∈ EXT (FT12.<credited
course>[where FT12.<university code>.<Student.ID>= 'ET1.arg1'] AND
FT12.<course credits>[where FT12.<university code> .
<Student.ID> = 'ET1.arg1' AND where FT12.<credited course >
= 'mathematics']>8)
OR⁷³
[IF ET1.arg2= 'history' THEN (language and culture ∈ EXT (FT12.<credited
course>[where FT12.<university code>.<Student.ID>= 'ET1.arg1'] AND
FT12.<course credits>[where FT12.<university code> .<Student.ID>
= 'ET1.arg1' AND where FT12.<credited course>= 'language & culture']>5)
OR
[IF ET1.arg2= 'economics' THEN (macro economics ∈ EXT (FT12.<credited
course>[where FT12.<university code>.<Student.ID>= 'ET1.arg1'] AND
FT12.<course credits>[where FT12.<university code> .

⁷³ In the sense of an exclusive OR

```

<Student.ID> = 'ET1.arg1' AND where FT12.<credited course >
= 'macro econ.' ]>8)

OR

[ IF ET1.arg2='medicine' THEN(biology ∈ EXT (FT12.<credited
course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND
FT12.<course credits>[where FT12.<university code> .
<Student.ID> = 'ET1.arg1' AND where FT12.<credited course >
= 'biology' ]>5)

OR

[ IF ET1.arg2='law' THEN(finance ∈ EXT (FT12.<credited
course>[where FT12.<university code>.<Student.ID>='ET1.arg1'] AND
FT12.<course credits>[where FT12.<university code> .
<Student.ID> = 'ET1.arg1' AND where FT12.<credited course >
= 'finance' ]>5)
AND IF EXT (FT10.<chosen major>|where Ft10.<university cod>.
<student ID>='ET1.arg1' ) ≠ ∅ ]
DO Delete (student'Et1.arg1' has chosen major 'z')
Insert (student'Et1.arg1' has chosen major 'ET1.arg2').

```

A.6 REFERENCES

- Chakravarthy, S., Mishra, D. (1994): Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering* **14**:1-26
- Dayal, U., Hsu, M., Ladin, R. (1990): Organizing Long-Running Activities with Triggers and Transactions. *ACM-SIGMOD international conference on management of data 1990*: p. 204-214.
- De, P., Sen, A. (1984): A new methodology for database requirements analysis. *MIS quarterly*, september:179-193.
- Leung, C. , Nijssen, G. (1988): Relational Database design using the NIAM conceptual schema. *Information Systems* **13** : 219-227.
- McFadden, F., Hoffer, J. (1994): *Modern database management*, 4th edition, Benjamin/Cummings.
- Nijssen, G., Halpin, T. (1989): *Conceptual schema and relational database design: A fact based approach*, Prentice-Hall, Englewood Cliffs.
- Prabhakaran, N., Falkenberg, E. (1988): Representation of Dynamic Features in a Conceptual Schema. *Australian Computer Journal* **20** (3): 98-104
- Rolland , C. (1983): Database dynamics. *Data base*, spring 1983: 32-43.

APPENDIX B:

THE META MODEL FOR THE NLM REQUIREMENTS SPECIFICATION LANGUAGE

In this appendix we will narrow down the 'real' or 'constructed' world of interest (Universe of Discourse) to the UoD of a Natural Language Modeling analyst. The result of applying the NLM way of working from chapter 6 on a significant set of examples from this UoD will be called the NLM information meta model (Dedourek et al., 1989). Firstly, the list of concepts and their definitions for the complete NLM UoD are summarized.

| Concept | Definition |
|---|--|
| <i>Sentence group template</i> | <i>The ordering of fixed and variable parts of a group of sentences that reflect domain semantics</i> |
| <i>Role</i> | <i>a variable part in one or more sentence group template (def. 5.2)</i> |
| <i>Role code</i> | <i>a name class</i> |
| <i>Intention</i> | <i>the meaning or the definition of a concept in a real or abstract world (def. 5.3)</i> |
| <i>Intention name</i> | <i>a name class</i> |
| <i>Verb</i> | <i>the parts of a sentence group template that are not variable</i> |
| <i>Fact type</i> | <i>a set of roles (def. 5.5)</i> |
| <i>Fact type code</i> | <i>a name class</i> |
| <i>Basic information model</i> | <i>the union of intentions and their definitions of a UoD, a set of roles, a set of fact types, a set of sentence group templates for every fact type... (def. 5.7)</i> |
| <i>Population state constraint</i> | <i>a proposition that limits the allowed extensions of a basic information model to those extensions that comply to the proposition in the population state constraint. (def. 5.11)</i> |
| <i>Population state transition constraint</i> | <i>a proposition that limits the combinations of before-after extensions of a basic information model to those before/after extension combinations for which the proposition is true (def. 5.12)</i> |
| <i>Population constraint</i> | <i>a population state constraint or a population state transition constraint</i> |
| <i>Derivation Rule constraint</i> | <i>a function defined on instances of the ingredient fact types resulting in instances of the derived fact type (def. 5.13)</i> |

| Concept | Definition (ctd.) |
|--|---|
| <i>Derivation Rule argument</i> | <i>specifies the types of values that must be specified for the creation of a derived fact instances (def. A.4)</i> |
| <i>Information base</i> | <i>A proposition on the information base (def. A.5)</i> |
| <i>Information base condition type</i> | <i>A set of propositions defined on the information base (def. A.6)</i> |
| <i>Pre-condition</i> | <i>is an information base condition (type) that checks whether the required input fact instances and derivation rule argument values for the derivation rule exist in the application information base (def A.7)</i> |
| <i>Post-condition</i> | <i>specifies (parts of) the fact argument for the instances of the resulting fact type(s) that must be created in the derivation rule. (def A.8)</i> |
| <i>Event occurrence</i> | <i>a happening at a certain point in time in the application subject area that can lead to the execution of one or more derivation rules and/or the insertion or deletion of fact instances into/from the application's information base (def 5.14)</i> |
| <i>Event</i> | <i>is one or a number of potential happenings in the application subject area that can lead to the execution of one or more derivation rules and/or the insertion or deletion of fact instances into/from the application's information base. (def. 5.15)</i> |
| <i>Event type</i> | <i>a class of events in the application subject area, each of which lead to the execution of one or more derivation rules (of the same type) and/or the insertion or deletion of fact instances into/from the application's information base (def 5.16)</i> |
| <i>Event type argument</i> | <i>specification of the intentions, instances of which must be known at the occurrence of an event instance of the event type (def. A.12)</i> |
| <i>Impulse</i> | <i>The occurrence of event leading to the instantiation of a derivation rule and or insert/delete process eventually when an information base condition evaluates to true (def. A.13)</i> |
| <i>Impulse type(constraint)</i> | <i>An ordered triplet that contains an event type, an Information base condition type under which the occurrence of an event of an event type can lead to the execution, of a derivation rule constraint or insert/delete operation (def. A.14).</i> |
| <i>Constraint</i> | <i>A population constraint, derivation rule constraint or impulse constraint</i> |

Concept**Definition (ctd.)***Constraint code**a name class*

Requirements specification a basic information model for that UoD together with all population constraints for which a legend is defined, derivation rule constraints and impulse type constraints, that reflect the business rules in that UoD(def. 5.19)

Secondly, we will apply the way of working of the NLM requirements determination method on the ‘real-life’ examples for the NLM analyst, those real life examples can be verbalized in a number of ways. In chapter 5 we have provided a verbalization legend for the graphical or diagrammatic NLM format. In that legend we considered the fact type template as a ‘string’ which contains no internal structure. However, in the analyst’s UoD that we will focus on in this appendix the ‘analyst’ (in the sense of an information systems developer) is a meta-analyst (in the sense of an information systems development process developer⁷⁴) and this ‘meta-analyst’ wants to make a distinction into the intentions, roles and naming conventions that are embedded in the NLM fact type or sentence group templates. This means that the following ‘pseudo’ example verbalization by the analyst:

The sentence group template x of fact type FtX is ‘A1 c1<RI>.....AN cN<Rn>’

Will be translated into real declarative natural language sentences by the ‘meta-analyst’. We will now show the results of the meta-analysis applied on a simple example when the verbalization, grouping, classification and qualification and atomization transformations have been applied:

The sentence group template ‘x’ of fact type ‘FtX’ has in position ‘1’ the verb ‘A1’

The sentence group template ‘x’ of fact type ‘FtX’ has in position ‘2’ the verb ‘c1’

The sentence group template ‘x’ of fact type ‘FtX’ has in position ‘K-2’ the verb ‘aN’

The sentence group template ‘x’ of fact type ‘FtX’ has in position ‘K-1’ the verb ‘cN’

The sentence group template ‘x’ of fact type ‘FtX’ has in position ‘3’ the role ‘R1’

The sentence group template ‘x’ of fact type ‘FtX’ has in position ‘K’ the role ‘RN’

The intention ‘c1’ for fact type ‘FtX’ has an identification structure that contains the role ‘R1’

The intention ‘cN’ for fact type ‘FtX’ has an identification structure that contains the role ‘RN’

The information meta model for the partial model in natural language modeling that refers to the basic information model and the population constraints contains the following NLM specific intentions: Fact type, fact type code, role, role code sentence group template, verb-part, intention, intention name, together with following fact types in which these intentions play roles: VERB IN TEMPLATE , ROLE IN TEMPLATE

⁷⁴ See the Universal framework for information activities in Auramäki et al. (1987).

C27 to illustrate the applicability of NLM on itself. Constraint *c1* expresses that a sentence group template can not have a role and a verb-part in the same position at the same time. Constraint *C2* expresses that every sentence group template of every fact type has at least one verb-part and at least one role or no verb and no role at all. Constraint *C3* expresses that a constraint predicate can not have a reference to a role and a verb part in the same position. Constraint *c4* expresses that a constraint has at least one reference to a role and at least one verb part or no verb part and no role at all. Constraint *C5* expresses that every role that is referenced in a constraint predicate has to be defined as a role in a fact type template of the basic information model. Constraint *C6* expresses that in a specific sentence group template of a specific fact type in a specific position there can exist at most one reference to a role. In figure B.1 a part of the information meta model for NLM is shown. We note that we have chosen an atomic reference scheme for roles. Furthermore, we have specialized the names from the archetype into nominal names and ordinal names. In the latter group or scale type the names have an ordering.

B.1 REFERENCES

- Auramäki, E., Leppänen, M., Savolainen, V. (1987): Universal framework for information activities. *Data Base*. Fall/Winter 87/88: 11-20
- Dedourek, J., Sorenson, P., Tremblay, J. (1989): Meta systems for information processing system specification environments. *INFOR* **27**(3): 311-337

SUMMARY (in dutch)

Dit proefschrift handelt over informatiebehoeftebepalingsmethoden ten behoeve van het ontwikkelen van informatiesystemen. In de afgelopen 35 jaar hebben deze methoden zich ontwikkeld via structured analysis and design tot de object-georiënteerde methoden zoals de Unified Modeling Language (UML). Het toepassingsdomein van deze methoden is echter ook veranderd in die 35 jaar. De belangrijkste ontwikkeling in dit toepassingsgebied is de verschuiving van 'maatwerk' naar implementaties van 'standaard' product-software, zoals ERP. Voorbeelden van ERP pakketten die geparameteriseerd, kunnen worden voor een specifieke implementatie zijn SAP/R3 en BAAN. In dit proefschrift tonen we aan dat een goede informatiebehoeftebepaling nog steeds maatgevend is voor de kwaliteit van het uiteindelijke informatiesysteem. We laten eveneens zien dat de belangrijkste bestaande informatiebehoeftebepalingsmethoden niet aan alle kwaliteitseisen die men aan deze methoden dient te stellen, voldoen. Dit derhalve leidt tot de hoofdvraagstelling in dit proefschrift in hoofdstuk 1:

Bestaat er een informatiebehoeftebepalingsmethode die kan worden toegepast in een breed scala van organisaties voor het specificeren van de volledige informatiebehoefte voor een bepaald deelgebied van zo'n organisatie, in het kader van het ontwikkelen en implementeren van informatiesysteem voor (een deel van) zo'n organisatie.

In hoofdstuk 2 van dit proefschrift worden een aantal criteria gegeven waaraan een informatiebehoeftebepalings methode dient te voldoen.

In hoofdstuk 3 van dit proefschrift worden de belangrijkste informatiebehoeftebepalingsmethoden geanalyseerd en worden een aantal tekortkomingen van deze methodieken blootgelegd.

In hoofdstuk 4 worden de criteria die in hoofdstuk 2 zijn gegeven verwerkt met de tekortkomingen van de bestaanden methodieken. Dit resulteert in een 19-tal ontwerp-eisen waaraan een te ontwikkelen methodiek dient te voldoen.

In hoofdstukken 5 en 6 van dit proefschrift wordt een nieuwe informatiebehoeftebepalingsmethodiek beschreven: Natuurlijke taal Modelleren (NLM). Deze methodiek kenmerkt zich doordat er een volledig gespecificeerd stappenplan wordt gegeven dat precies aangeeft hoe een informatie-analist in samenspraak met een domeingebruiker kan komen tot een specificatie voor een applicatie. De NLM methodiek neemt gebruikersvoorbeelden als een startpunt en kenmerkt zich doordat op het gehele informatiebehoeftebepalingstraject, in het jargon van de gebruiker wordt gecommuniceerd. De NLM methodiek voldoet tevens aan de 19 ontwerp-eisen, die we in hoofdstuk 4 aan een dergelijke methodiek hebben gesteld.

In hoofdstuk 7 wordt geconcludeerd dat NLM (onder meer) een informatiebehoeftebepalingsmethode is die een aantal tekortkomingen van bestaande methodieken heeft verbeterd en voldoet aan de door ons verkregen 19 ontwerp-eisen.

CURRICULUM VITAE

Peter Bollen werd op 12 Juli 1959 te Maastricht geboren. Hij doorliep van 1971 tot 1977 het VWO aan het Stedelijk Lyceum aldaar, waarna hij tot 1978 Wiskunde studeerde aan de Technische Universiteit Eindhoven. Van 1978 tot 1984 studeerde hij Bedrijfskunde eveneens aan de Technische Universiteit Eindhoven. Van 1984 tot 1986 was hij werkzaam bij Mars chocoladefabriek b.v. te Veghel waar hij binnen een aantal functies op het gebied van logistiek, productie en operations research werkzaam is geweest. Vanaf 1986 is hij werkzaam als universitair docent bedrijfsinformatiekunde binnen de vakgroep Managementwetenschappen van de faculteit Economie en Bedrijfskunde aan de Universiteit Maastricht, waar hij vanaf 1994 heeft gewerkt aan het onderzoek dat in dit proefschrift beschreven wordt.