Software Specification Based on Re-usable Business Components

PROEFSCHRIFT

Ter verkrijging van de graad van doctor aan de Technische Universiteit Delft, op gezag van de Rector Magnificus prof.dr.ir. J.T. Fokkema, voorzitter van het College voor Promoties, in het openbaar te verdedigen op 26 september 2005 om 15:30 uur

door

Boris Blagovestov SHISHKOV
Master of Science in Management Information Systems
Sofia University St. Kliment Ohridski, Bulgaria

geboren te Sofia, Bulgarije
Dit proefschrift is goedgekeurd door de promotor:

Prof.dr.ir. J.L.G. Dietz

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter
Prof.dr.ir. J.L.G. Dietz, promotor
Prof.dr.ir. J. Filipe

Prof.dr. K. Liu

Prof.dr. R. Wieringa
Prof.dr. W. Gerhardt
Prof.dr. H. Koppelaar

Technische Universiteit Delft
Escola Superior de Tecnologia de Setúbal, Portugal
University of Reading, United Kingdom
Universiteit Twente
Technische Universiteit Delft

Published by:
B.B. Shishkov
P.O. Box 104
1618 Sofia
Bulgaria

SIKS Ph.D.-thesis series, no. 2005-17
SIKS P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

ISBN 90-9019924-1

Cover Design: C.P. Shishkova

Copyright © 2005 by B.B. Shishkov

All rights reserved. No part of the material protected by this copyright notice may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without written permission from the author.

Printed in The Netherlands by Sieca Repro, Delft
Dedicated to my father
Summary

The current Information and Communication Technology (ICT) is of significant importance for the proper flow of processes belonging to a number of business domains. Software applications are to facilitate the utilization of ICT for the mentioned purpose. Thus more and more research takes place on application development methodologies, and also more and more industrial projects appear related to this issue. However, often such projects are characterized by unrealized goals, low user satisfaction, and increasing budgets. It is claimed that one frequent cause of software project failure is the mismatch between the business requirements and the actual functionality of the delivered application. This problem relates to the misconception that a Business System (process) is a kind of Information System (process). Instead, they should be considered in different ways. Although they both are basically Social Systems, they differ in the kind of the production: business services and (internal) information services, respectively. Therefore, in order to adequately reflect the requirements in the software system-to-be, one needs to soundly align the business process modeling and software specification, mapping a pure business-oriented model towards the specification of a software system. Realizing such an alignment in a component-based way seems feasible and beneficial because component-based business and software models would allow for re-use, and also for good modeling traceability and ease of modifiability. Next to that a business component would concern business process modeling concepts while a software component would concern software concepts; hence a rigorous mapping between the two would be a good foundation for a business-software alignment. However, to date the component paradigm has only really penetrated the implementation and deployment phases of the software life-cycle, and does not yet play a major role in the earlier analysis and design activities of large software projects. In the software context, components are associated mainly with the current 'physical' component technologies (e.g., .NET, CORBA, EJB). The SDBC approach will be presented (SDBC stands for Software Derived from Business Components), which aims at addressing this issue by considering 'logical' components that represent the logical building blocks of a software system. From this position, SDBC proposes a mechanism for business-software alignment. In particular, the approach allows for deriving a pure business process models (Business CoMponents) and reflecting them in conceptual (UML-driven) software specification models (Software CoMponents). In the Business CoMponent identification, SDBC follows a multi-aspect business perspective guaranteeing completeness. In the Business CoMponent – Software CoMponent mapping, SDBC follows rigorous rules, guaranteeing adequate alignment. Being UML-driven, SDBC is in tune with the current software design standards. The application of SDBC has been explored in a large Dutch insurance company.
# Contents

**CHAPTER 1: INTRODUCTION** ................................................................. 1

1.1 Background ....................................................................................... 1

1.1.1 The technical progress, ICT, and their impact on the business area 2

1.1.2 Utilizing ICT and the role of ICT applications ............................... 2

1.2 The Research Problem ....................................................................... 4

1.3 Identified Promising (Research) Perspectives .................................... 5

1.3.1 Component-based system development ....................................... 8

1.3.2 Language-action perspective ....................................................... 8

1.4 Research Goal and Scope ................................................................ 9

1.4.1 Research goal ............................................................................. 9

1.4.2 Research scope and characteristics ......................................... 10

1.5 Motivation ......................................................................................... 10

1.6 The Research Questions ................................................................... 11

1.7 The Research Strategies and Approach ........................................... 12

1.7.1 Research strategies and justification ......................................... 12

1.7.2 Research approach .................................................................... 14

1.8 Outline of the Thesis ....................................................................... 17
# Contents

## CHAPTER 2: ESSENTIAL CONCEPTS .................................................. 18

2.1 Systems ................................................................................ 18
2.2 Business Systems ................................................................. 20
2.3 Information Systems ............................................................... 25

## CHAPTER 3: ANALYSIS OF TOOLS .............................................. 29

3.1 Business Process Modeling Tools ............................................. 30
  3.1.1 Semi-formal Business Process modeling tools: analysis ......... 34
    3.1.1.1 The Communication perspective, LAP, and DEMO ......... 34
    3.1.1.2 The Structural perspective, OS, and Semantic/Norm Analysis .... 37
    3.1.1.3 The Dynamic perspective, Workflow Management, and Petri Net 40
    3.1.1.4 The Data perspective, LFP, and ORM ................................ 42
    3.1.1.5 The Simulation perspective, Discrete Event Sim., and ARENA .... 44
  3.1.2 Business Process modeling tools: conclusions .................... 46
3.2 Software Specification Tools .................................................... 46
  3.2.1 Software specification tools: analysis ................................. 47
    3.2.1.1 UML ........................................................................ 47
    3.2.1.2 Other relevant tools – KobrA, Catalysis, Tropos .................. 52
  3.2.2 Software specification tools: conclusions ............................ 57
3.3 Summary ................................................................................. 58

## CHAPTER 4: THE SDBC APPROACH ......................................... 60

4.1 Essential Issues Behind SDBC ............................................... 61
  4.1.1 Integrated view over BP modeling and software specification .... 63
  4.1.2 Business Process modeling embracing the communicative aspects 69
  4.1.3 Component-based business-software alignment .................... 71
  4.1.4 Re-use issues .................................................................... 72
4.2 SDBC - Outline ................................................................. 81
4.3 Concluding Remarks ................................................................ 87
### CHAPTER 5: MAKING SDBC OPERATIONAL

- 5.1 Selecting Appropriate Tools - General Issues ........................................ 89
- 5.2 Focusing the Selection ........................................................................ 90
  - 5.2.1 DEMO – suitabilities for SDBC .................................................. 92
  - 5.2.2 Petri Net – suitabilities for SDBC .............................................. 95
  - 5.2.3 ARENA – suitabilities for SDBC ................................................. 96
  - 5.2.4 ORM – suitabilities for SDBC ...................................................... 97
  - 5.2.5 Use cases – suitabilities for SDBC .............................................. 97
- 5.3 Concluding Remarks ........................................................................... 98

### CHAPTER 6: APPLYING SDBC

- 6.1 Information Analysis and Structuring ................................................ 99
  - 6.1.1 Identifying actor-roles ................................................................... 100
  - 6.1.2 Elaborating on the inter-role relations ......................................... 101
  - 6.1.3 Summarizing the achieved output ............................................... 104
- 6.2 Identification of Business Components ............................................. 106
  - 6.2.1 Business Process patterns ............................................................ 106
  - 6.2.2 Mapping towards Business Components ..................................... 107
- 6.3 Deriving a Software Specification Model ........................................... 109
  - 6.3.1 Deriving use cases from Business Components ............................ 109
  - 6.3.2 Further specification and elaboration .......................................... 116
- 6.4 Business/Software Models’ Validation ............................................... 118
- 6.5 Concluding Remarks .......................................................................... 124

### CHAPTER 7: CASE STUDY

- 7.1 Background ......................................................................................... 125
- 7.2 Icomp ............................................................................................... 127
- 7.3 Applying SDBC .................................................................................. 129
  - 7.3.1 From the case information to Business Components .................... 129
  - 7.3.2 Elaborating a Business Component ........................................... 143
  - 7.3.3 Towards software specification .................................................... 160
- 7.4 Concluding Remarks .......................................................................... 165
Contents

CHAPTER 8: CONCLUSIONS ......................................................... 166

8.1 The Research Questions .................................................... 168
8.2 SDHC ................................................................. 172
8.3 Outlook for Further Research ............................................ 174

BIBLIOGRAPHY ............................................................ 175

SUMMARY IN DUTCH ....................................................... 187

ACKNOWLEDGEMENTS ..................................................... 189

ABOUT THE AUTHOR ....................................................... 190
Chapter 1

INTRODUCTION

Truth is ever to be found in the simplicity, and not in the multiplicity and confusion of things.

Isaac Newton

The research reported in this thesis concerns the alignment between the creation of a software artefact and the consideration of its corresponding Business System. The thesis introduces innovative ideas about that by proposing a software specification approach.

Being the thesis' introduction, the current chapter consists of eight sections. The first one introduces the research background. This is done by considering consequently the impact of the current information/communication technology on the business area and the role of software applications in utilizing the technology. The second section specifies and elaborates the (identified) research problem: the (frequently observed) mismatch between the functionality of a delivered application and the original (business) requirements. The third section considers two research perspectives, namely the Component-Based (system) Development and the Language-Action Perspective, which we claim to be adequate for approaching the identified problem. On this basis, the fourth section derives the research goal (concerning the identification of business components and their use for specifying applications) and sub-goals, and also defines the research scope. Then the fifth section analyzes the relevance of the (proposed) research. All this is reflected in the sixth section which formulates the research questions. The main research question is: How can (re-usable) business components fill the gap between Business Process modeling and software specification? The seventh section introduces and elaborates the (adopted) research approach. And finally, the eighth section gives an outline of the entire thesis.

1.1 BACKGROUND

A proper utilization of the current Information and Communication Technology (ICT) could improve business effectiveness considerably [Lin and Chlamtac 2001; Wanyembi 2002]. This could be seen not only from the digital multimedia supporting Business Processes to date [Laudon and Laudon 2005] but also from the powerful
telecommunication facilities (such as GSM, GPRS, UMTS/3G [GSM], and Wi-Fi [Wi-Fi]) which significantly improve business possibilities and quality.

It is widely accepted that adequately utilizing ICT comes through providing an appropriate ICT application support [Atkinson et al. 2001; Shishkov 2002-1]. That is not only because the utilization of digital information (characterizing the current ICT) requires processing (which means in turn software support [Laudon and Traver 2004]) but also because the complexity of current technological platforms [AWARENESS] requires a software intermediary [BETADE] that bridges the technology and its users.

This section will elaborate firstly the impact of ICT on the business area and secondly the role of applications for the utilization of ICT.

The latter directly concerns the essential goal in this PhD project, namely – to propose improvements in the way software is specified, particularly focusing on the need to bridge the design of software and a prior Business Process modeling.

1.1.1 The Technical Progress, ICT, and their Impact on the Business Area

Throughout the human evolution, marked by the ‘Industrial Revolution’ [Landes 2003], the ‘Rockefeller Era’ [Knowles 1973], and the ‘Digital Era’ [Laudon and Laudon 2005], information sharing and communication have been issues of constantly increasing importance [Barjis et al. 2000]. Presently we witness significant relevant developments – mainly associated with the current global telecommunications and the digital multimedia [Laudon and Traver 2004]. Businesses are using information and information technology to gain and sustain a competitive advantage [Haag et al. 2003].

These possibilities have led to the emergence of new ways of collaborating [Barjis and Shishkov 2001-2; Cross and Dublin 2002], doing business, and working. E-Business appeared as a way of doing business relying completely on ICT [Laudon and Laudon 2005; Laudon and Traver 2003; Shishkov 2002-2; Shishkov and Barjis 2002; Shishkov and Dietz 2002-1]. Tele-Work appeared as a way of working and sending work results, being independent of place and time [Laudon and Traver 2004; Shishkov and Dietz, 2002-4; Shishkov and Barjis 2001]. Telecenters appeared to facilitate people in learning about the latest technological possibilities and using them in the everyday life [Shishkov and Barjis 2000].

Therefore, if properly utilized, current ICT could create huge improvements in the way people realize business activities.

1.1.2 Utilizing ICT and the Role of ICT Applications

A motivation for a further utilization of ICT relates to the increasing public demands towards business. They result from people’s being informed about possible technology-driven improvements. For instance, a bank customer can easily imagine how convenient it would be to manage his/her bank account through a mobile telephone or Internet [Haag et al. 2003]; a trade customer can easily see that, if properly applied, ICT could allow for increasing the speed and quality of business transactions as well as for decreasing the occurrence of errors.

Hence, by realizing all this, people actually increase their demands towards business, requiring high level of speed and quality, low level of errors, mobility, information security, and so on.

Thus, as mentioned before, businesses should embrace current technology (ICT) in order to stay competitive, meeting the mentioned increasing demands.
However, the new technology alone is not always applicable in a straightforward way. For example, the mere existence of Third-Generation Mobile Services [Lin and Chlamtac 2001] does not directly lead to their appropriate usage in realizing business activities. What is required for this aim, in general, is [Shishkov 2002-1]:

(1) A proper business (re-) design so that the technical support is realizable in a straightforward way.
(2) Intermediary between the technical possibilities and the particular business activities.

Commenting on (1):
Designing a Business Process includes designing its essence firstly and secondly, designing its realization [Dietz 2003-1; Dietz 1994]. The essence of a Business Process stays the same no matter what kind of technical support is or is not applied. For instance, the essence of transferring money from bank account ‘A’ to bank account ‘B’ is the same irrespective of whether it is realized through visiting a (bank) branch office or triggering the transaction via the Internet. The role of a technical support is to bring about some improvements in the realization of a Business Process (in this case – the possibility to realize a business activity from distance, using a computer). Therefore, the technology usually facilitates another way of doing the same thing; it, consequently, relates to the realization (not to the essence) of a Business Process.

Nevertheless, one could find examples where the existence of particular technical facilities motivates some evolution in the essence of Business Processes. For instance, the project AWARENESS [AWARENESS] considers technology-driven monitoring of patients, called ‘tele-monitoring’. Unlike the usual monitoring of patients, which takes place within a hospital, tele-monitoring is characterized by the possibility for both patients and caregivers to be independent in terms of location. As a result of this, Business Processes should be designed differently in tele-monitoring and usual (patient monitoring) cases. In a usual patient monitoring situation, in case of emergency, the alarm signal is sent to the alarm desk of the hospital where the patient is while in a tele-monitoring case, the alarm signal is sent to the hospital closest to where the patient is. This means that Business Process re-design activities need to be performed (in such cases) before introducing a (new) technical support. This might include making a Business Process distributed, bringing it in tune with some current distributed technologies [Visser et al. 2000; BETADE; Simon 1996; Shishkov et al. 2002-2].

Taking into account that we address (within the current research) particularly questions concerning the alignment between Business Process modeling and Software Design, the (above-discussed) Business-Process-re-design-related issues are considered to have just indirect influence on our research activities. Moreover, a number of research studies have already been realized in this direction. Existing studies on Business Process re-engineering [Dietz 1994; Giaglis and Paul 1996] and distribution [Shishkov and Barjis 2000] are examples of this.

Commenting on (2):
Conducting a proper Business Process (re-) design and supplying an appropriate technology is not enough for adequately employing it in support of a Business Process. It is also necessary to bring together the technical (supportive) facilities and the realized business activities – without linking the two, one cannot speak of
technical support to a Business Process [Shishkov and Dietz 2005-1; Dietz 2004]. As already stated, current ICT is characterized by digital representation of information; utilization of digital information requires processing which means in turn software support. Therefore, ICT applications are to be the intermediary between a particular technical possibility and a business activity [Shishkov 2002-1]. In other words, ICT applications are about to allow the user take full advantage of the particular technical possibility in realizing business duties. This is how business activities could be put on advanced technological foundation [Jacobson 1995; Liu 2000].

For this reason, there is a great demand nowadays for applications which facilitate effectively target (business) processes, by allowing them fully utilize the possibilities of ICT [WARENNESS; BETADE]. That is why more and more research takes place on application development methodologies [Shishkov 2002-1].

Summarizing: in order to make current Business Processes adequate to user demands, it is essential allowing these processes utilize properly ICT. In order to achieve this, it is necessary creating an intermediary between them and the technical possibilities. ICT applications are to play the role of such an intermediary. For this reason, the design of ICT applications, as a bridge between the business world and the technical possibilities, is of importance for the business development.

1.2 THE RESEARCH PROBLEM

The development of ICT (software) applications has been put in a bad light for decades already [Dietz 2004]. This could be seen from the great sums of money wasted in such developments because of projects which are either cancelled or fail to fulfil time and budget requirements [Liu 2000]. Among the challenges concerning current application development is the (frequently observed) mismatch between the original (business/user) requirements and the actual functionality of the delivered application [Shishkov and Dietz 2005-1]. We actually observe two opposite phenomena [Shishkov 2002-1]:

On the one hand, we (often) observe software being designed without prior adequate consideration of the Business Processes to be supported by it [Shishkov and Dietz 2004-1]. In such cases, therefore, the application design does not stem from a higher-level (conceptual) Business Process model capable of precisely grasping the (business) requirements. Instead, such a design is conducted in an ad hoc intuitive way, using some best practices and/or existing technological solutions. This means that the business requirements are poorly determined and the Software Design model is incompletely addressing the original business information. An application, designed in such a way, would thus support the Business Processes inadequately; although the application’s quality might be high from a software point of view, the effectiveness of the support offered to the target Business Processes would remain low.

On the other hand, there are examples of ‘business-modeling-driven software specification’, relating to Business Process management [Aalst 2004] or business data modeling [Halpin 1998]. In these cases, it is claimed that a workflow/data-driven Business Process model could be straightforwardly mapped towards a software specification model. However, the lack of appropriate transformation mechanisms supporting such a mapping leads to just a partial reflection of the Business Process model in a Software Design model [Shishkov and Dietz 2004-2]. Next to that, both Business Process management and business data modeling
Introduction

Concern just particular business modeling viewpoints, missing other ones (such as the structural perspective, for example); actually, the software community misses examples of business-modeling-driven software specification, which are characterized by a complete Business Process modeling input [Shishkov and Dietz 2003-1]. Hence, relying on the current knowledge, it is a great challenge to adequately map a complete Business Process model towards software specification [Shishkov and Dietz 2004-1]. For this reason, although it might be possible to conduct sound Business Process modeling prior to the design of an application, the Business Process model would be used partially, since it is not straightforwardly transformable into a relevant input for the application design. This would not allow for full utilization of ICT possibilities in solving the particular business problem(s).

Therefore, the two tasks: the Business Process modeling and the specification of applications for the support of the Business Processes, need to be better aligned. They should be considered as one integrated task.

Many researchers have addressed issues related to this problem. Dehnert and Rittgen (2001) present a formal notation for describing Business Processes. This would be a step forward if further related to Software Design. Olivera, Filho and Lucena have also contributed in this direction, by investigating the design of software on the basis of business requirements analysis [Olivera et al. 2001]. Their suggested approach is also a promising achievement even though it does not yet offer a straightforward mapping of a Business Process model to a Software Design model. Hikita and Matsumoto have studied how the appearance of additional requirements could be reflected in the system's construction [Hikita and Matsumoto 2001]. This is also a useful result accomplished so far, although not completely solving the problem. Kruchten suggests (based on the existing use case concepts [Jacobson et al. 1992]) the 'Business use case' concept – considered useful in bridging Business Process modeling and Software Design [Kruchten 2000]. However, the question remains of how to consistently identify such use cases.

These are only some of the examples of research activities addressing the problem mentioned above (others are discussed in Chapter 3), and none have yet reached a complete and convincing solution. Therefore, it might be concluded that further knowledge is still required relating to the issue of basing application design soundly on Business Process modeling.

1.3 IDENTIFIED PROMISING (RESEARCH) PERSPECTIVES

Approaching the identified research problem intuitively, we discover two layers to be considered, namely Business Process modeling layer and Software specification layer. The significant issue concerning the problem is that adequate alignment mechanisms are needed between these two layers. And also, in aligning them, it is essential that the original business information, including the domain-imposed business requirements (to be discussed in Chapter 4), is precisely and completely grasped.

As for the first of the above mentioned issues: in aligning the mentioned two layers, we should rely on compatible concepts to be defined in both layers. In other words, we should be able to address Business Process modeling and software specification in such a way that we could use compatible business and software concepts. Further, the achieved modeling output should be consistent with some of the existing software standards; otherwise our research results would have no
practical value. A conclusion drawn out of the literature survey was that Component-Based Development - CBD (rooted in Object-Orientation – OO [Booch 1994]) is the only popular and widely used software-related paradigm which is reflectable also in Business Process modeling. The CBD way of thinking [Szymerski 1999] and also the essential CBD/OO principles could be much useful not only in designing software but also in conducting Business Process modeling, as advocated by Jacobson [Jacobson et al. 1995]. Such claims have inspired research activities aiming at considering the component concept (to be studied in the following chapter) not only in Software Development but also in Business Process modeling [Fowler 1997; BETADE]. Therefore, the component concept is to be addressed as a good candidate for a business/software-compatible concept. Another advantage regarding components is that they could be re-used and considered as self-contained units (separately from their environment) – all this reduces complexity in analysis/design activities and allows for flexibility [Atkinson and Muthig 2002]. Thus, we give the following hypothesis (illustrated on Figure 1.1):

- Component-based alignment between Business Process modeling and software specification is possible and useful.

As seen from the figure, mapping is to be achieved between business components (Business Process modeling concepts) and software components (Software Design concepts). The terms business component and software component will be elaborated in Chapter 2.

![Figure 1.1: First research hypothesis](image)

As for the issue of guaranteeing Business Process modeling completeness, we claim (being inspired by the ‘LAP research community’ [Winograd 1988]) that considering just the structural, dynamic, and data perspectives (as it is usually done today [Booch 1994; RMODP Part 1]) over a business reality is insufficient; it is necessary to capture as well the communicative actions concerning the Business Process actors [Shishkov and Dietz 2005-1]. This claim will be motivated further in this section. Thus, we give the following hypothesis (illustrated on Figure 1.2):

- In order to completely grasp a business reality, it is necessary to consider not only the structural, dynamic, and data views (perspectives) over it but also a communication view.
Further elaborating the hypothesis, we claim that both business and software analysis/modeling are to include a consideration of at least three perspectives, namely structural, dynamic, and data, as shown on the figure. This corresponds to the intuitive way of perceiving any system: viewing firstly its structure, secondly its behaviour, and thirdly its factual issues. Moreover, this is in tune with the DEMO methodology [DEMO] (widely considered in this research) and also with the latest software design facilities such as UML/MDA [OMG 2000; MDA 2001] and ODP [RMODE Part 1,2,3]. Hence, it is widely accepted that approaching a business/software modeling task, one should consider these three perspectives. However, what most of the current Business Process modeling tools miss are mechanisms for addressing the ‘communication’ perspective (as above mentioned), capturing in this way the communicative actions characterizing a particular Business Process. We are going to consider this Business Process modeling perspective in the current research, which we claim would contribute to achieving a more complete Business Process modeling view. Therefore, we find it sensible approaching the Business Process modeling task through four perspectives, namely Structural Perspective, Dynamic Perspective, Data Perspective, and Communication Perspective.

As also seen from Figure 1.2 (indicated by the dashed line), we envision an indirect influence of the communication issues concerning the Software specification layer; this is in tune with the currently widely used (mentioned) Software Design facilities. Moreover, Software Systems abstract partially from the complexity of real-life (business) systems [Winograd and Flores 1986], which mainly concerns the business communicative actions (Software Systems are not characterized by such communicative actions; instead, they are characterized by rules (norms) which indirectly reflect such actions). This makes the Dynamic perspective sufficient (to some extent) for describing all actions and behaviors concerning a Software System [Aalst et al. 2000]. However, although indirect, the ‘effect’ of the Communicative perspective over the Software specification layer is acknowledged and it could be found, for example, in the structure and derivation of use cases; as will be seen in the following chapters, providing a sound (Communication-perspective-driven) Business Process modeling foundation for the use case derivation process, would guarantee that the resulting software is adequately based on the original business reality.

Further on in the current section, the two research perspectives (corresponding to the defined research hypotheses), claimed to be useful with regard to this research (the Component-Based (system) Development and the Language-Action Perspective - LAP), will be addressed consequently.
1.3.1 Component-Based System Development

The Component-Based Development (CBD) [Jacobson et al. 1992; Gamma 1994] is considered to be a promising contemporary approach for application design and development, founded on the principles of Object-Orientation (OO). As it is well known, the OO paradigm (characterized by the fundamental concepts of encapsulation, classification, inheritance, and polymorphism) is widely considered as a special approach to the construction of models of complex systems, in which a system consists of a large number of objects. According to some researchers, the CBD/OO could be applied not only to Software Systems but also to Business Systems [Jacobson 1995]. Thus, it seems feasible to expect that software specification and Business Process modeling could be bridged by basing the specification on software components which are derived from some business components. Such components should fill the gap between the two mentioned tasks. If re-usable components are identified, they could be used many times for designing different applications. Next to that, Component-Based Development seems beneficial for the application design itself. By basing application development on encapsulated, individually definable, re-usable, replaceable, interoperable and testable (software) components, developers could build applications which possess durable configuration and a high degree of flexibility and maintainability. The process of application development would also be improved because building new applications would include using already developed components. This reduces development time and improves reliability. The performance and maintenance of developed applications would be enhanced because changes could occur in the implementation of any component without affecting the entire application. All this makes the component-based application development much more effective than the traditional way of application development.

For all these reasons, the proposed research, focusing in general on the alignment between Business Process modeling and software specification, focuses in particular on realizing this on the basis of (re-usable) business components identified from target Business Processes. By basing the design of applications on such components, it is expected that the application support to Business Processes can be improved considerably.

1.3.2 Language-Action Perspective

As it was stated already, people demand currently more and more from Business Processes. For example, it is often required that such processes are modified and/or connected to other Business Processes [AWARENESS]. The increasing demands towards Business Processes cannot be met adequately anymore by relying only on best practices [Dietz 2003-2]. A promising perspective, convincingly claiming to soundly approach the modeling of Business Processes is the Language-Action Perspective (LAP). LAP is a theoretical orientation towards approaching the modeling of Business Processes by emphasizing the importance of interaction and communication. The theory recognizes that language is not only used for exchanging information, as in reports, for example. Language is used, however, also to perform actions, as in promises or orders, for instance. Such actions are claimed to represent the foundation of communities and organizations, and must be understood to create effective Information Systems (the Information System concept will be introduced in Chapter 2). For this reason, we claim that adequately capturing the communication aspects characterizing a considered Business System (the Business System concept will be introduced in Chapter 2), would contribute to the creation of sound and complete Business Process models.
1.4 RESEARCH GOAL AND SCOPE

Considering the introduced research background, and based on the identified research problem and promising research perspectives, we are going to define in this section the research goal and also determine the research scope.

1.4.1 Research Goal

As discussed earlier, the principles underlying the CBD paradigm appear applicable to the task of aligning Business Process modeling and software specification. Regarding the Business Process modeling: the Communication perspective characterizing the Business System(s) under consideration, is claimed to be crucial for the creation of a sound and complete Business Process model. Hence, the main research goal is summarized as follows:

To investigate the (LAP-driven) identification of (re-usable) business components and their use for specifying ICT applications which effectively support Business Processes.

In order to achieve the main goal, four basic sub-goals have to be realized:

**sub-goal 1**
To develop a tool-independent conceptual framework for specifying software based on (re-usable) business components.

Two requirements are identified in this regard:
- It is essential that the framework offers mechanisms for identifying business components (as will be seen in the following chapter, in our view, a business component is a Business Process concept) based on business reality description and requirements. Here, it is crucial to correctly and completely grasp the requirements in identifying the components. It is important also to have the possibility of identifying (whenever it is possible) re-usable business components - to be used for specifying different software artefacts.
- The framework should also offer mechanisms for reflection of the components (as Business Process models) into the specification of software. This means soundly basing the specification model on a Business Process model and thus achieving an adequate mapping.

**sub-goal 2**
To develop an approach through which the framework could be applied.

A set of appropriate modeling tools should be suggested and step-by-step guidelines should be specified, through which the framework could be applied in solving concrete problems. Hence, the guidelines should represent a precise outline of the activities to be realized in specifying software based on (re-usable) business components as well as in validating the created models.

**sub-goal 3**
To identify (within a business domain) several (re-usable) business components, following the framework and approach.
A target business domain should be selected. An investigation of the characteristics of Business Processes within the domain is to take place. Based on this, several business components (including re-usable ones) are to be identified. This identification should be done according to the developed framework and approach.

**sub-goal 4**

To demonstrate, by applying the suggested approach, how software specification could be realized based on the identified (business) components.

A software specification model should be built up, using as a basis a Business Process model which includes the identified (re-usable) business components. The specification model is to be used to demonstrate the advantages of specifying software by soundly deriving it from component-based Business Process model(s). This derivation should be realized according to the developed framework and approach.

### 1.4.2 Research Scope and Characteristics

Focusing on the identification of business components and their use for specifying software, the current research appears to concern the synthesis of concepts from two disciplines, namely Business Processes and Software Design. The direction is towards a better understanding of what should be the role of a Business Process model in the design of software. Further, it appears that the research is characterized by yielding theories which are themselves constructions, generated from an insider perspective (as opposed to yielding objective and independent theories generated from an outsider perspective). Said otherwise, the current research contributes to innovative theories (which are themselves under development) rather than further developing existing and well-established ones.

### 1.5 MOTIVATION

As mentioned before, in this section we are going to motivate the current research by analyzing its **relevance** - scientific and societal:

**Scientific relevance**

In order to achieve a correct reflection of the business requirements in specifying software artefacts, it is necessary to adequately align Business Process modeling and software specification. An important scientific contribution of this research is to fill in this gap; a correct consideration of the business requirements should allow for building software that adequately supports Business Processes, allowing them to properly utilize ICT.

Bringing about some additional scientific benefits concerns the consideration of the Component-Based (system) Development - CBD. Following the principles of the CBD paradigm is claimed to be most beneficial in approaching the mentioned research problem. The purpose is that the research framework (to be suggested) be based on re-usable business components being identified from target Business Processes and further reflected in the design of software. The business components themselves, identified in a domain (for example, e-Business), would be useful for other investigations and research activities related to the particular domain. Further, the knowledge developed concerning the identification of business components could
also be used in identifying components in other domains. And also, this research is expected to demonstrate how Business Process modeling and Software Design could be aligned in such a way that component-based application development is supported in a straightforward way. Achieving results in this area would be particularly helpful for current Software Design since this promising way of designing software applications is comparatively new and still not thoroughly explored.

Societal relevance
By contributing to knowledge on how to realize effective software support for Business Processes, this research is expected to be a useful societal contribution. The effective software support is crucial especially for those businesses which are heavily dependent on ICT. Examples of such business areas are: banking, brokering, and e-business. The areas mentioned are among the most critical business areas for the current global prosperity. Improvements there would create substantial benefits for Society.

1.6 THE RESEARCH QUESTIONS

In order to grasp correctly the concepts which are fundamental to the current research, the following two research questions (derived out of the research goal) need to be answered:

**question 1**  
What benefits can the CBD way of thinking bring to the modeling of Business Systems?

**question 2**  
What must be understood by ‘business component’?

The final (research) output within this thesis should supply an answer to the main research question:

**question 3**  
How can (re-usable) business components fill the gap between Business Process modeling and software specification?

In relation to this question as well as to Sub-goals 1 and 2, we identify the three following research questions:

**question 4**  
How can business components be re-used for the purpose of specifying different software applications?

**question 5**  
How can identified business components be validated in terms of structure and dynamics?

**question 6**  
Which of the existing modeling environments and tools could be employed for effectively aligning Business Process modeling and software specification in a component-based way?

In answering the questions formulated above, knowledge of the two main research disciplines (mentioned in Section 1.4), namely Business Processes and Software Design, should be acquired. Figure 1.3 represents them together with the
Chapter 1

particular questions related to each of them. It is seen also that a synthesis between these disciplines should be accomplished, in answering questions 3 and 4. The star symbol next to question 6 indicates that although the question is fundamentally related to Software Design, it also concerns Business Processes.

![Diagram showing the relationship between Business Processes, Software Design, and questions 1 to 6.](image)

*Figure 1.3: Addressed research areas*

1.7 THE RESEARCH STRATEGIES AND APPROACH

In order to find answers to the research questions and accomplish the research goal, it is essential to base the research activities on sound methodological foundation. This means to delimit the research domain and select appropriate strategies to be applied in accordance with the adopted research approach. All this is discussed below; the research strategies and approach are considered in Sub-sections 1.7.1 and 1.7.2, respectively.

1.7.1 Research Strategies and Justification

In general, there are several ways of doing scientific research. Among them are: Literature Study, Case Study, and Field Experiment. These are called 'research strategies' in the terminology of Yin [Yin 1994]. Each strategy represents a different way of collecting and analyzing (empirical) evidence, following its own logic, and possesses peculiar advantages and disadvantages. Each of the (mentioned) strategies could be used for all three purposes – exploratory, descriptive, or explanatory. However, it is not always easy to decide which strategy to use for any particular purpose, basing the decision on clear criteria. Even though each strategy has its distinctive characteristics, there are large areas to overlap among them [Sieber 1973]. Thus, the goal should be to avoid 'gross misfits': when one type of strategy is planned to be used but another is really more advantageous.

According to Yin, there are three conditions, distinguishing the strategies:

- the type of research questions;
- the control the explorer has over actual behavioral events;
- the degree of focus on contemporary as opposed to historical events.

This could determine the choice of proper strategies in planning a research.

Considering the taxonomies of scientific research strategies, introduced and discussed in [Galliers 1992; Yin 1994], and taking into account the characteristics of the current research, it is suggested to base it on a compilation of several research strategies which complement each other: Literature Study, Case Study, Survey Research (interviews with experts), and Archival Research. These strategies are briefly outlined and discussed below.
Literature Study is considered to be an important part of any research because this strategy is essential for defining a conceptual framework at the outset of the research, for refining this framework in the course of the work as well as for formulation of hypotheses. It also helps for gaining new insights into the problems under investigation and for ensuring that the research being developed would not be just a replication of a past research project [Miles 1979].

Literature Study will be an important strategy to be used in the current research since this strategy could contribute to:

- finding out the state-of-the-art of the research area, investigating concepts and theories;
- basing the research activities on results already available, by exploring the findings of other researchers and their achieved results;
- orientation about tendencies and development perspectives in the explored domain;
- refining the research goals and questions throughout the research.

The Case Study research strategy contributes (in general) to capturing some practical perspectives of the investigated problems. It is helpful for considering the knowledge of the practitioners in exploring the research area. According to Yin [Yin 1994], a Case Study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident. The Case Study as a research strategy comprises an all-encompassing method – with the logic of design incorporating specific approaches to data collection and data analysis. Although in the past Case Study had been considered only as an exploratory tool [Platt 1992], they have proved to be more than just an exploratory strategy. Some of the best and most famous Case Studies have been both descriptive and explanatory [Yin 1994].

Usually, Case Studies are the preferred strategy when "how" or "why" questions are being posed, when the investigator has little control over events and when the focus is on a contemporary phenomenon within some real-life context [Yin 1994].

In order to realize successfully this strategy, it is essential to design properly the particular Case Study, to collect precisely consistent evidence and to analyze it.

Particularly in the current research, the Case Study strategy is proposed being applied following a specific goal, namely to test the applicability of (proposed) research framework and approach.

Surveys (interviews with experts) usually contribute to acquiring specific knowledge, related to the explored area. Such interviews could also be used to provide the researcher with feedback (e.g. concerning a developed framework and its potential effectiveness).

By Archival Research we mean study of documentation and archival information. When Business Systems and processes are investigated, it is often necessary to explore such documentary information in order to get an extended insight.

The set of the selected strategies needs to be justified, taking into consideration the overall features of this research.

The current research faces areas, not enough explored, e.g. component-based software design. This requires mastering new theoretical knowledge. At the same time, the achieved results are applied in a concrete business domain and the validation of these results is related to the consideration of real business activities. Thus, it is necessary to investigate existing business processes and specify software that is to support them. This requires some concrete results. All these project
features make it relevant to use the research strategies outlined above, paying special attention to Literature Study and Case Study. A match between the features of the research and the relevant research strategies is represented in Table 1.1. The star symbol on the second row (from the bottom up) indicates that more explanation is needed about the particular role of the Case Study strategy within the 'validation and evaluation' task. This role is suggested to be: testing the applicability of the (proposed) research framework and approach.

<table>
<thead>
<tr>
<th><strong>Features of the Current Research Project</strong></th>
<th><strong>Relevant Strategy/ies</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Necessity to face areas, not enough explored</td>
<td>Literature Study, Survey</td>
</tr>
<tr>
<td>Necessity to investigate particular b. processes</td>
<td>Archival Res., Survey, Lit. Study</td>
</tr>
<tr>
<td>Necessity to specify software</td>
<td>Literature Study</td>
</tr>
<tr>
<td>Necessity for validation and evaluation</td>
<td>Case Study*, Survey</td>
</tr>
<tr>
<td>Necessity for generalization of the res. results</td>
<td>Literature Study</td>
</tr>
</tbody>
</table>

*Table 1.1: Justification of the research strategies*

### 1.7.2 Research Approach

A key element to a successful development in any study is a sound and precise step-by-step outline of the research activities to be realized. It is essential to identify a set of interrelated tasks which need to be performed for fulfilling the research goals.

The suggested research approach is depicted in Figure 1.4, in a flow-chart like diagramming technique. It is suggested, as shown in the figure, the research tasks to be realized in five phases which are outlined below.

**Phase 1** is devoted to theoretical study of the considered research area; in particular, the two disciplines, fundamentally essential for the research, are addressed, namely: Business Processes and Software Design. The basic purpose is to become familiar with the area of study. Analysis of the two mentioned disciplines (especially the first one) should provide the answers to (research) Questions 1 and 2. As proper research strategies for this phase are suggested Literature Study and interviews with experts.

In **Phase 2**, a target domain should be selected and investigated, where to apply the created deliverables, in validating the research results. Also, based on the results of Phase 1, Phase 2 should provide two important research outputs (reflected also in Sub-goals 1 and 2, respectively):

1) A tool-independent conceptual framework that allows for identification of (reusable) business components, addressing properly some business requirements, and also for reflection of the (business) components into the specification of software (within this task, the focus is on the synthesis between the two disciplines of the research area).

2) An integrated approach which allows for the application of the proposed framework. In this regard, the current research considers in particular the UML-based software specification, not only because of the completeness of the Unified Modeling Language [OMG 2000; Booch et al. 1999; Kruchten 2000] but also because it turns out to be de facto the standard language for modeling Software Systems [Mallens et al. 2001], widely accepted by both researchers and practitioners. Additional arguments in favor of selecting UML are to be found further on, in the chapters to come.

With respect to founding the UML-based software specification on Business Process modeling, a fundamental goal should be to find out how to consistently derive use cases from a Business Process model (it is well-known that use cases are
modeling constructs that serve to link the application domain (the business world) to the software domain, regarding the UML-based software specification). It should be taken into consideration that the software community still misses adequate guidance in identifying use cases. Methods for construction of UML Use case diagram [Jacobson et al. 1992; Fowler and Scott 2000; Shishkov and Dietz 2001] on the basis of Business Process modeling are still needed. Therefore, an essential issue to be provided by the approach should be derivation of use cases from Business Processes. Another important issue should be to relate the activities (discussed above) to prior Business Process modeling, adopting a Business Process modeling tool. DEMO [Dietz 1999; Dietz, 1994] is considered to be a proper tool in this regard, for a number of reasons. Among them are:

- the consistency of DEMO with the selected (Section 1.3) LAP perspective;
- the possibility to capture the essence of the explored Business Processes, completely abstracting from all realization issues; this would give the software designer the right (re)design freedom;
- the good potentials for consistently basing a use case model on a DEMO Coordination Structure Model [Shishkov and Dietz 2002-3,4];
- the completeness of DEMO, resulting in its wide applicability and use.

More arguments in this direction will follow further on in the thesis.

As seen from the figure, the three outlined tasks, namely: domain investigation, framework construction, and approach design, are carried out in parallel, and only after each of them is completed, Phase 3 can be entered. Relevant research questions are 3, 4, 5, and 6. Basic research strategies to be applied are Literature Study and interviews with experts.

**Phase 3** should provide a repository of (re-usable) business components. The components should be identified from Business Processes within the domain investigated in Phase 2 (refer as well to Sub-goal 3). The identification process should be realized through the approach constructed in Phase 2, and based on a Case Study. The purpose is to validate the approach in some aspects (and its underlying framework), demonstrating how it could be helpful in the process of identifying business components.

The repository of business components needs to undergo analysis resulting in conclusions about its consistency. If the repository proves to be inconsistent, the constructed framework and approach should be investigated and improved. All this is important because of the crucial role of the business components for aligning Business Process modeling and software specification. Basic research strategies to be applied are Case Study, Literature Study, interviews with experts, and for some specific information concerning the target domain - Archival Research might be applied as well.

Relevant research questions are 2 and 5.

**Phase 4** is about software specification based on the identified business components (refer to sub-goal 4). In particular, following the approach created in Phase 2, software models should be derived on the basis of the business components, founding in this way the design of software components, as a final step in the process of aligning Business Process modeling and software specification. The resulting software specification model should be evaluated.

Relevant research questions are 3 and 4.

And finally, after the results have proven the usefulness of basing the specification of software on business components, in the last phase (**Phase 5**), conclusions should be drawn.
Figure 1.4: Overview of the research approach
1.8 OUTLINE OF THE THESIS

The content of the following seven chapters develops as follows.

Chapter 2 will address the essential concepts and considerations regarding the current research. This includes concepts adopted from existing theories and such ones proposed especially for the purpose of this research.

Based on that, we will analyze (in Chapter 3) the existing knowledge that relates to the issue on (component-based) alignment between Business Process modeling and software specification. The (widely used) relevant modeling tools will be considered, including DEMO and UML. They will be briefly introduced in the chapter, as tools which are very relevant to the addressed research problem.

In Chapter 4, we will introduce a tool-independent conceptual view on the way in which we suggest aligning Business Process modeling and software specification. We will accomplish this by proposing a software specification approach.

Chapter 5 will relate the introduced conceptual framework to appropriate concrete modeling tools. The chapter will present arguments in favor of choosing particular tools as suitable ones with regard to application of the conceptual framework.

Based on the conceptual framework (Chapter 4) and the theories behind the selected modeling tools (Chapter 5), Chapter 6 will introduce step-by-step methodological guidelines concerning the application of the introduced framework.

In Chapter 7, the proposed approach will be demonstrated by means of a case study. This is considered helpful for validating the approach, relating it to some real-life situations as well as for evaluating its quality, and drawing conclusions about some of its strong and weak points.

Chapter 8 presents the conclusions and suggestions for further research.
Chapter 2

ESSENTIAL CONCEPTS

A short saying often contains much wisdom.
Sophocles

Following Chapter 1 which presented the framework of the current research, we consider it necessary to introduce the concepts which have fundamental relevance to it. Hence, in this chapter, we will identify these basic concepts; all other relevant concepts will be considered in the chapters to come.

Taking into account that in any scientific discipline (purely scientific or applied), it is necessary to study a system of some kind, we will start our conceptual exploration by clarifying what we mean by 'system'. Then we will consider the system concept through the perspectives of the two disciplines characterizing the current research, namely Business Processes and Software Design (we envision Software Design as relating to Software Development which represents one of the phases of the Software Engineering process; in Chapter 4, we will discuss this further). This is expected to lead to an adequate delimitation of the notions of 'Business System' and 'Information System'. Such a delimitation is essential because of the currently observed misconception that a Business System is a kind of Information System. Instead, they should be considered in different ways. Although they both are basically Social Systems, they differ in the kind of the production: business services and (internal) information services, respectively [Dietz 2003-1].

And finally, based on all this, we will identify and specify essential concepts relevant to the research goals and questions (Chapter 1), in the context of either Business Systems or Information Systems.

The first section of this chapter concerns systems in general. The second section is about Business Systems and related concepts. The third section is about Information Systems and related concepts.

2.1 SYSTEMS

As mentioned above, considering a scientific discipline includes addressing a system of some kind: a Physical System, a Social System, a Biological System, and so on. Hence, we aim at defining 'system' not only in general but also with regard to
the two above mentioned disciplines, namely Business Processes and Software Design. The General Systems Theory [Bertalanffy 1976; Boulding 1956] proposes a unified approach in considering a system, based on the (justified) claims that:

- there are some concepts and structural principles that seem to hold for systems of many kinds;
- there are some modeling strategies – in particular the State Space Approach – that seem to work everywhere.

Hence, Bunge (1979) defines Systemics as 'this set of theories that focus on the structural characteristics of systems and can therefore cross the largely artificial barriers between disciplines'.

The cognitive rationale of Systemics is the wish to discover similarities among systems of all kinds despite their specific differences. The practical motivation for Systemics is the need to cope with the complex systems that exist in industrial societies. A Systemics ‘specialist’ would therefore de-emphasize the particular scientific discipline's aspects and will focus instead on the structure and behavior of the considered system(s). This would include imitating (modeling or simulating) the system’s behavior. In relation to this, Bunge defines also Systems Analysis as being wider in sense than Systemics. Unlike Systemics, System Analysis is not particularly interested in de-emphasizing the peculiarities and building extremely general models. It aims instead at drawing flow-charts, network diagrams, and occasionally specific mathematical models. The essential goal is to enable one understand how a system operates.

Because of our considering a two-discipline research perspective (within the current thesis), the (above) described perspective on systems (which is basically inspired by the views of Bunge) is claimed being adequate to follow. For this reason, we have adopted the 'system' definition which has been proposed by Bunge (1979):

**Definition 2.1** Let $\mathcal{T}$ be a nonempty set. Then the ordered triple $\sigma = \langle C, E, S \rangle$ is system over $\mathcal{T}$ if and only if $C$ (standing for Composition) and $E$ (standing for Environment) are mutually disjoint subsets of $\mathcal{T}$ (i.e. $C \cap E = \emptyset$), and $S$ (standing for Structure) is a nonempty set of active relations on the union of $C$ and $E$. The system is *conceptual* if $\mathcal{T}$ is a set of conceptual items, and *concrete* (or material) if $\mathcal{T} \subseteq \Theta$ is a set of concrete entities, i.e. things.

Hence, we have defined system, positioning it in the Systemics perspective. On this basis, we claim that it is possible doing further system categorization depending on the research area of interest. This would lead to a clearer delimitation of the studied area. Some examples of such categories are:

- Legislative System – a system concerning legal norms and acts;
- Planet System – a system concerning planets;
- Political System – a system concerning political subjects.

As already stated, our interest relates mainly to the disciplines characterizing this research, namely Business Processes and Software Design. That is why we are particularly interested in two system categories. These are:

- Business Systems;
- Information Systems.

As for the Business System concept, it should correspond to our view on business, in general. By 'business things' we do not mean only things concerning trade/commerce but we do also include in that notion (applying a broader perspective) all things that refer to any organized activity which is driven by a particular goal. Next to that, we envision businesses as human-driven – humans are those through whom businesses operate. This is in tune with the LAP/DEMO concepts [Dietz 2003-2] which prove to be adequate to current business modeling
demands. Based on these considerations as well as on Definition 2.1, we propose the following definition [Shishkov and Dietz 2005-1]:

**Definition 2.2** A system should be considered being a *Business System* if and only if it is composed of human beings collaborating among each other through actions which are driven by the goal of delivering products to entities belonging to the environment of the system.

Following the mentioned LAP/DEMO concepts, we mean by *product* anything that is or can be delivered to a customer. It includes material things (often called 'products') and immaterial things (often called 'services').

As for the 'Information System' concept, it should be considered not only in an ontological but also in a functional perspective, because the functional aspect is essential concerning the way in which an Information System supports informationally a Business System. Therefore, we will propose an ontological as well as a functional definitions of *Information System*.

The ontological Information System concept should correspond to our viewing Information Systems as composed of humans facilitated by ICT applications (to be further defined), who collaborate in realizing internal informational support to organizational processes. Based on these considerations as well as on Definition 2.1, we propose the following definition [Shishkov and Dietz 2005-1]:

**Definition 2.3** A system should be considered being an *Information System* if and only if it is composed of humans (often facilitated by ICT applications as well as technical and technological facilities) collaborating among each other driven by the goal of supporting informationally a corresponding Business System. Usually the Business System and the Information System belong to the same organization.

The functional Information System concept should correspond to the basic (as it is well-known) functions characterizing a (current) technological support: functions related to the creation of data, its processing, distribution, and so on. For this reason, we have adopted the following definition [Simon 1996]:

**Definition 2.4** Concerning its functional characteristics, an Information System is a system which manipulates data and normally serves to collect, store, process and exchange or distribute data to users within or between enterprises or to people within wider society.

In the following two sections, we will subsequently consider Business Systems and Information Systems, staying in the context of the current research.

### 2.2 BUSINESS SYSTEMS

Before considering concepts/definitions in the context of Business Systems, we will elaborate on how we suggest distinguishing such systems.

An essential criterion (in consistency with Definition 2.2) on whether a human/action belongs to a Business System should be his/her having (or not) a relation to the goal that drives the system. For example, a company's goal could be the delivery of high-quality consultancy products to a target market segment. In such a case, among the humans and actions having any relation to the goal, would be the analytical activities and their corresponding executors, the financial issues connected to the company-customer relation, and so on. If, in parallel with this, the same company is involved in another business (for example, accommodation providing),
then the actions and humans related to this business (such as maintenance of the rooms offered for rent and those who are responsible for it, advertising of the accommodation services, and so on) should not be considered belonging to the Business System, since they are irrelevant with respect to the mentioned goal.

Thus, although it does not directly concern the composition and structure of a Business System, the goal driving it has to be taken into account when considering such a system.

In identifying Business Systems, it is important having a clear understanding also about the actions and humans concerning it.

As for the actions, we view two types of actions that could take place in a Business System, namely production and communicative ones. Production Actions do concern a particular output (a material/immaterial product) while Communicative Actions concern the collaboration (within the Business System) that is in support of the realization of the Production Actions. This view is in tune with the concepts within the DEMO methodology (to be presented in Chapter 3).

As for the humans, we claim that it is easier describable if considering the particular actor-role (for example the role ‘Information provider’) associated with an action rather than the particular human executing the action. Our reason for claiming this is motivated by the observation that in current business activities often a particular human could be involved in the execution of different types of actions; hence, it would be complicated describing one and the same executor involved in a number of actions. For this reason, we propose viewing a Business System as a collection of actions and corresponding actor-roles. This is depicted in Figure 2.1 where actor-roles are actually the elements of the composition and the actions concern the structure:

![Figure 2.1: Simplified view on a Business System](image)

As seen from the figure, within a Business System, we view a number of actions whose realization relates to corresponding actor-roles.

In order to bring deeper clarification regarding Business Systems, we need to further elaborate on the actions. As already mentioned, we distinguish between Production Actions and Communicative Actions; thus, we need a concept for reflecting this as well as the ‘action-role’ link. Since the ‘Transaction’ concept (a concept from the methodology DEMO [DEMO]) fulfils this demand and also because of its full consistency with the current modeling tendencies (as it is widely agreed upon), we have adopted the ‘Transaction’ concept. It is claimed to have a good capacity for grasping the essential issues concerning the actions taking place within a Business System. In connection with this concept, some clarification should be made regarding the actor-role concept. In DEMO, actor-roles are considered as being fulfillable by particular humans. Hence, an actor-role could be fulfilled by a number of humans and also (usually) a human could fulfil a number of actor-roles. If we take the situation of an actor-role being fulfilled by a particular human, then the combination of the human and the actor-role is called ‘actor’. Hence, we propose considering the following definition [Dietz 2003-2]:

21
**Definition 2.5** A *Transaction* is a finite sequence of coordination acts between two actors, concerning the same production fact. The actor who starts the Transaction is called the *Initiator*. The general objective of the initiator of a Transaction is to have something done by the other actor, who therefore is called the *Executor*.

Hence, Transactions should be considered as the *elementary building blocks* of a Business System. As studied by Dietz (2004), Transactions are related to each other in a tree-structure. The top of the tree is called the *Starting Transaction* [Shishkov and Dietz 2004:1]. It is a Transaction that is not caused directly by another Transaction (from the particular tree) but triggers the execution of other Transactions (within the tree).

The Transaction concept will be discussed further in the following chapters.

We claim that considering Transaction trees rather than Transactions is more appropriate to be done in the modeling of Business Systems because in considering Transactions, the complexity is rather big: even a simple Business System would contain a great number of Transactions, making it difficult for modelers to precisely grasp and describe these Transactions. Therefore, the consideration of Transaction trees would help partitioning somehow the multitude of Transactions, grouping them into segments. We introduce the 'Business Process' concept in this regard [Shishkov and Dietz 2004:1]:

**Definition 2.6** A *Business Process* is a structure of (connected) Transactions that are executed in order to fulfill a Starting Transaction.

Thus, in our view, the operation of Business Systems concerns Business Processes (which are driven by the goal characterizing the system). Each Business Process consists of Transactions, including a Starting Transaction. Transactions relate to Initiators and Executors. This view is presented in Figure 2.2.

The figure shows a particular example of a Business System operation. It concerns many Business Processes. Four of them are depicted in the figure, namely \(bp_1, bp_2, bp_3, \text{ and } bp_4\). As seen from the figure, each of the Business Processes (driven generally by the goal \(<G>\)) consists of Transactions (with a starting transaction on top). The Transactions are presented by diamonds; the Starting Transactions are represented by diamonds colored black. A Starting Transaction could be activated in any of the following three ways: outside cause (activation from a customer), periodic activation (usually concerning payment activities), and activation resulting from a waiting relation (a Transaction could start only after another one is completed) [Dietz 2003:1].

Summarizing so far: we have presented our viewing the operation of Business Systems as concerning a number of Business Processes driven by a common general goal. We have also elaborated on our defining a Business Process.

A further consideration of Business Systems should include the question on the decomposition of such systems: firstly, because decomposition reduces complexity in the consideration of any system (as it is well known) and secondly, because considering particular parts of a Business System could allow for addressing them separately and also for re-using them. For this reason, we will consider Business Sub-systems, by proposing the following definition:

**Definition 2.7** A *Business Sub-system* is a system which is a part of a Business System.
Based on the definition, it becomes clear that if $W$ is the set containing all the Transactions and actors included in a Business System, any sub-set $W' \subseteq W$ which satisfies the system definition, would represent a Business Sub-system.

Nevertheless, using this concept without any other restrictions makes little use because of the undeterministic nature of the concept: any combination of Transactions and actor-roles could be a Business Sub-system.

Making use of Business Sub-systems means, in our view, that we must have clear criteria what types of Business Sub-systems to use and how could we take advantage of their re-use. This includes a clear granularity positioning of the Business Sub-systems which we are to consider.

We claim that a possible and logical way to define a particular useful type of Business Sub-systems is selecting such Business Sub-systems which are related to corresponding Business Processes. We have arguments in support of that claim:

- The issues related to a particular Business Process are distinguishable from all other issues that belong to the particular Business System.
- Business Processes relate to a useful granularity level ('between' the 'Transaction level' and the 'Business System level').

For these reasons, we will consider particularly those Business Sub-systems which relate to particular Business Processes. We will call these sub-systems 'Business Components', proposing the following definition [Shishkov and Dietz 2005-1]:

**Definition 2.8** A Business Component is a Business Sub-system that comprises exactly one Business Process.

We have now introduced and clarified some basic concepts for this research, paying special attention to the 'Business Component' concept. Our defining it positions this concept within the Business Engineering area unlike most of the (current) definitions according to which 'Business Component' is a Software Engineering concept [Abolhassani 2003; Atkinson et al. 2001].

After having introduced the Business Component concept (and other relevant ones), we should reflect this into the (current) research context. It is featured by the goal of aligning Business Process modeling and software specification in a component-based way. As we have discussed already, this means to derive software specification models on the basis of re-usable models of Business Components. For this reason, we should clarify what we mean by 'model'. We should also explain how we will consider Business Components in this perspective, how we would build
models of Business Components. And finally, we should discuss how such models could be useful in aligning Business Process modeling and software specification. Thus, we will (below) define 'model' and 'Business CoMponent', and explain how Business CoMponents could be used for the (mentioned) goal.

Our view on what is a model is in tune with the generally accepted attitude (which is widely agreed upon) according to which a model of System A is a system being used in order to acquire knowledge about System A. What modelers usually do is to make a representation of the system they need to understand [Hommes 2004] — such a representation is their resulting system. Hence, in defining 'model', we follow the definition of Apostel (1980):

**Definition 2.9** Any subject using a system A that is neither directly or indirectly interacting with a system B, to obtain information about the system B is using A as a Model for B.

Because of our claiming the significant role of four perspectives (Chapter 1), namely Structural Perspective (over Transactions and actors, and their relations), Dynamic Perspective (over the flow of Transactions), Data Perspective, and Communication Perspective (over the collaborative acts realized in support of the realization of Transactions), we will use the term 'Complete Model' to denote a model elaborated in all these four perspectives. This is defined as follows:

**Definition 2.10** A Complete Model is a model that is elaborated at least in four perspectives, namely Structural Perspective, Dynamic Perspective, Data Perspective, and Communication Perspective.

We will also introduce (below) the 'Business CoMponent' concept denoting a (complete) model of a Business Component (the word 'CoMponent' is derived from the word 'Component' being with capital 'M' to indicate the relation to the word 'model'):

**Definition 2.11** A Business CoMponent is a Complete Model of a Business Component.

Hence, our goal (within the current research) should be to investigate how (re-usable) Business CoMponents could be used for aligning Business Process modeling and software specification in a component-based way. In the Figure 2.3, we have shown how do the concepts 'Business System', 'Business Component', and 'Business CoMponent' relate, according to the above definitions and explanations.

![Figure 2.3: The Component-CoMponent relation](image)

As seen from the figure, we view a Business System as composed of Business Components. We would represent such components in terms of Business CoMponents via modeling. The CoMponents are to be used as input for further software specification tasks. That is discussed in the following section.
2.3 INFORMATION SYSTEMS

As mentioned in Section 2.1, the current section will consider Information Systems and related concepts. As also mentioned there, these issues are to be addressed from the perspective of the current research (defined in the Introduction). This includes:

- the software specification task and its role in the creation of Information Systems;
- the link between Business CoMponents and (software) specification.

Before proceeding towards these issues, we need to relate precisely Information Systems to Business Systems, in conformity to the 'system' definition as well as definitions 2.2 and 2.3. As it could be seen from them, in both cases:
- the compositional elements are humans;
- actor-roles could be considered (instead of particular humans);
- the structure concerns inter-role relations driven by a goal.

However, in Business Systems, the goal is the delivery of business products to entities which belong to the environment of the system while in Information Systems, the goal is the informational support of a corresponding Business System. As for environments: the environment of a Business System usually consists of actor-roles and actions external to a corresponding organization and/or team while the environment of an Information System, in contrast, (usually) consists of actor-roles and actions which are internal for the particular organization/team.

In considering the functional relation between Business Systems and Information Systems, we have been following (because of its wide popularity) the 'Management Paradigm'. It is explained and discussed in [Galagan 2004]. According to the paradigm:

- a Business System exploits an Information System (by benefiting from services that the Information System provides to it);
- an Information System supports a Business System by providing services to it.

As already explained, such sort of support to current Business Systems is usually realized using ICT applications which allow Business Systems to utilize the current (ICT) technology. Our defining an ICT application is as follows:

**Definition 2.12** An ICT application is an implemented software product which realizes a particular functionality supporting in this way the humans who are elements of the composition of a Business System or an Information System.

Therefore, ICT applications are the main instrumentarium through which the informational support to Business Systems is realized. In the majority of cases, the goal of an ICT application is to automate some Business Process(es) which belong to the Business System being supported. Thus an ICT application would (then) relate to a particular Business Component. The component, therefore, should be precisely reflected in the specification of the (ICT) application; otherwise the application support would be inconsistent with the broader Business System context. Thus, it is logical to propose using Business CoMponents as a source for deriving the specification of ICT applications. We have depicted all these issues on Figure 2.4.
As seen from the figure, the support (indicated by the dashed line) that an Information System realizes to a Business System is facilitated (actually driven) by ICT applications. As seen also from the figure, a Business Component might support the specification of an ICT application. Hence, of particular interest are the relations:


Otherwise said, we are interested how a (re-usable) Business Component could be identified and also how it could be reflected in the specification of an ICT application.

Therefore, we would need to discuss the role of specification in the design and development of an ICT application and also possibilities of decomposing the specification model.

We will first position the 'specification' task, considering the three-phase software creation process (we will discuss it in Chapter 4). These three phases [Atkinson and Muthig 2002] are:

- **specification**, addressing the functionality of the software artefact-to-be;
- **realization**, addressing the specification's (further) refinement and also technological aspects;
- **implementation**, addressing the model-based coding bringing about the final software application output.

The modeling support that is provided by a Business Component affects the specification phase as depicted in Figure 2.5.

As far as an ICT application is concerned, we have to take into account the (current) software development standards according to which it should be allowed for using components in implementing software.
Usually, within the software community, the term *Software Component* is associated with the Component-Based Development of ICT applications (mentioned in the Introduction), which is characterized by assembling re-usable Software Components. They represent prefabricated, configurable, and independently evolving building blocks which provide some functionality that can be used separately or in composition with the functionality provided by other Software Components.

According to the ‘Middleware Perspective’ [Guareis de Farias 2002], which does not necessarily envision a Software Component in the context of the development of an ICT application, Software Components are blocks of code ready to be deployed on top of a suitable execution environment (often called 'Container') which provides a number of generic services for the execution of components, such as Event Notification, Authentication, and so on.

Therefore, we could conclude about several essential characteristics of Software Components, relevant to the Software Engineering domain:

1. Any Software Component is to be characterized by a particular functionality driven by its goal to provide some service to its environment.

2. In its providing a service, a Software Component could collaborate with other Software Components.

3. The environment of a Software Component may consist of other Software Components, ICT applications, supporting platforms [MDA 2001], and so on.

Hence, in addressing Software Components, we consider it necessary paying attention to the interface specification, component dependencies, deployment, and granularity – these issues are briefly discussed below. This is in tune with related studies reported in [Guareis de Farias 2002].

An *interface specification* can be seen as a contract which is established between a Software Component providing (implementing) a service and the component’s environment using (invoking) it.

The *component dependencies* comprise the events that can be either produced or consumed by a Software Component, in its providing a service(s).

Given its binary representation, a Software Component is a self-contained building block which could be *independently deployed* in a variety of environments.

Being 'composable', a Software Component should not usually be a complete ICT application; it should be a part of the whole. It is well known, however, that there are examples of large Software Components which could be envisioned either as components or as applications. Thus, considering the *granularity* of a Software Component under development is of significant importance. In our view, in specifying the size of a Software Component, the modeler should take into account the fundamental requirement that a Software Component should be general enough to be re-used in a number of ICT applications [Shishkov 2002-1].

Hence, on the basis of the above analysis, we propose our relevant Software Component ontological definition [Shishkov and Dietz 2004-1]:

**DEFINITION 2.13** Software Components are implemented pieces of software, which represent parts of an ICT application, and which collaborate among each other driven by the goal of realizing the functionality of the application.

Since the Software Component concept concerns the implementation phase, we would need to propose also a functional definition, inspired by Szyperski (1998):

**DEFINITION 2.14** A *Software Component* is functionally a part of an ICT application, which is self-contained, customizable, and composable, possessing a clearly defined function and interfaces to the other parts of the application, and which also can be deployed independently.
Thus, by creating an instance of a Software Component, we do actually deploy it. We could view, therefore, such a component instance as an object. However, there is little agreement to date on the differences between components and objects [Guareis de Farias 2002]. For this reason, we will not enter now this discussion.

Since any support from a Business CoMponent would concern the specification phase, we should (thus) consider another relevant concept. Such a concept must refer to the logical building blocks of an ICT application (in contrast: the (already introduced) term 'Software Component' refers to the 'physical' building blocks of an application, in the sense of contemporary 'physical' component technologies, such as CORBA [CCM], .NET [Atkinson et al. 2001], EJB [EJB], and so on). We introduce the term 'Software CoMponent' to reflect the above mentioned logical aspect:

**DEFINITION 2.14** A Software CoMponent is a conceptual specification model of a Software Component.

Summarizing our views:
- A Business System consists of Business Components.
- An ICT application consists of Software Components.
- The creation of a Software Component is supported conceptually by a Software CoMponent.
- The identification of the Software CoMponents is supported conceptually by a Business CoMponent.

Figure 2.6 illustrates this:

![Figure 2.6: Business CoMponents and Software Components](image)

As seen from the figure and as already stated, a Business CoMponent supports conceptually the identification of at least one Software CoMponent. A Software CoMponent, further on, supports conceptually the creation of a Software Component.

Thus, the concepts which have been introduced in this chapter allow for considering the component-based alignment between Business Process modeling and software specification. The following chapter is going to analyse the existing relevant tools and knowledge. Then, based on the contents of the first three chapters of the thesis, Chapter 4 is to propose the conceptual framework of an approach allowing for the specification of software based on Business CoMponents.
Chapter 3

ANALYSIS OF TOOLS

(The world is full of obvious things which nobody by any chance ever observes.
Sir Arthur Conan Doyle)

Based on the introduced research background (Chapter 1) and considered concepts (Chapter 2), the current chapter will analyze relevant modeling tools (and related knowledge) which concern either Business Process modeling or software specification.

We have been interested particularly in such tools which consider both aspects in an integrated way (as according to the ‘business-software-alignment’ requirements stated in the Introduction). However, in the analysis we carried out, we did not come upon such tools that fully meet the mentioned requirements. We came instead either upon Business Process modeling tools which relate to software specification or upon software specification tools which relate to Business Process modeling.

Although such tools do concern the ‘business-software-alignment attitude’, we consider them being either ‘Business Process modeling tools’ or ‘software specification tools’; we therefore address them separately – in Sections 3.1 and 3.2, respectively.

We do not claim that the set of tools which we have analyzed is fully representative – taking into account the observed boom of modeling tools/techniques/languages [Hommes 2004], we realize that it is quite challenging to aim at grasping all existing tools, following a particular criterion. We claim for representativeness of our analysis, however, because we have selected for consideration some of the most popular and proven tools which are relevant to our formulated (in Chapter 1) research hypotheses and questions.

In our study we have stressed particularly on the strengths of the analyzed tools so that we would be able to use these strengths in our further research activities.

Therefore, the current analysis should provide the following benefits:
1. Concluding about to what extent do the existing tools allow for adequately bridging Business Process modeling and software specification, the analysis might motivate the need for introducing a new software specification approach.
2. Addressing particular relevant tools, the analysis would support the further steps of the current research, especially related to the (eventual) usage of any of these tools.

All these issues will be reflected in the last section of the current chapter (Section 3.3) which contains concluding remarks.
3.1 BUSINESS PROCESS MODELING TOOLS

By Business Process modeling tool we mean a modeling method/technique which could be useful in the analysis and/or (re-)design of Business Processes. In our view, it is logical to expect that any Business Process modeling tool is rooted in a particular theory(ies). If this is not the case, it would be difficult bringing sound evidence about the adequacy and consistency of proposed modeling steps and/or notations.

For this reason, in considering a Business Process modeling tool, we pay particular attention to its underlying theory(ies). With respect to this, we argue that it is hardly possible grasping all essential Business Process modeling perspectives (Figure 1.2) if using only one theory. In support of this claim is our observation that some of the major Business Process modeling theories [Hommes 2004] are mainly related to one perspective or another. For example, Organizational Semiotics [Liu 2000] is mainly concerned with the Structural perspective over a Business System, Workflow Management [WMC] – with the Dynamic perspective, the LAP theory [Winograd 1998] – with the Communication perspective, and so on. A conclusion drawn out of the literature survey was that these three theories have essential importance concerning their particular corresponding perspectives. Hence, we identify three major categories of Business Process modeling tools: *semiotic tools, workflow management tools*, and communicative analysis (LAP) tools. As for the Data perspective (Figure 1.2) we would not distinguish any particular theory because the database domain is undergoing significant developments currently, after the ‘ERP era’ [Batini et al. 1991]. That is why we consider in general factual (database-related) tools.

Besides these four main types of Business Process modeling tools (according to our categorization) we consider also the simulation-driven Business Process modeling tools. The reason is that great (and currently increasing) attention is paid to simulation within the modeling community [Ilkov 2004]. We have not categorized simulation tools as a sub-type of any of the four major types identified because different simulation tools might be concerned with different perspectives.

As concerns their formality, most tools belonging to these five categories are considered to be neither purely formal (strongly relying on mathematical apparatus) nor purely informal (lacking any structural/formal notations and/or guidelines). For this reason, we qualify these tools as *semi-formal*; it should be noted here that although there are some formal simulation tools, we qualify simulation tools as semi-formal because the majority of simulation tools are semi-formal according to our observations.

As for the purely informal Business Process modeling tools, we leave them beyond the scope of our consideration because a purely informal Business Process modeling output would be hardly translatable to software notations, given the main focus of the current thesis, namely business-software alignment.

Therefore, we would distinguish two main categories of Business Process modeling tools to be considered in this section, namely formal and semi-formal tools. As already stated, we identify five major sub-types of semi-formal tools for consideration, namely semiotic tools, workflow management tools, communicative tools, factual tools, and simulation tools. Hence, we propose the following classification where we have added also particular tools in the different categories:
Analysis of Tools

* Business Process modeling tools
  ** formal tools
  *** ...
  ** semi-formal tools
  *** communicative analysis tools
  **** DEMO
  ***** ...
  *** semiotic tools
  **** SEMANTIC/NORM ANALYSIS
  ***** ...
  *** workflow management tools
  **** PETRI NET
  ***** ...
  *** factual tools
  **** ORM
  ***** ...
  *** simulation tools
  **** ARENA
  ***** ...
  *** ...
  ** ...

We do not claim for exhaustiveness of the above classification – our goal was not to cover all popular Business Process modeling tools and make a plausible classification. Instead (as mentioned already), we have aimed at just identifying several major categories of Business Process modeling tools which are relevant to the current research. Logically, we approached this task by considering the four Business Process modeling perspectives (Figure 1.2) essential for the research.

As for the particular tools presented above:
DEMO [DEMO] is considered as a communicative analysis tool;
Semantic Analysis, Norm Analysis [Liu 2000] are considered as semiotic tools;
Petri Net [Petri Nets] is considered as a workflow management tool;
ORM [ORM] is considered as a factual tool;
ARENA [ARENA] is considered as a simulation tool.

Through analyzing these tools, we will realize a view over their corresponding underlying theories. By doing this, we are to achieve the following:

- justification of our claim that none of the currently popular Business Process modeling theories (and corresponding tools) is capable alone of adequately spreading over the mentioned four perspectives (Figure 1.2);
- conclusions about the relatability of each of these representative tools to software specification;
- identification of relevant values for the current research;
- ideas of possible synergies resulting from joint application of different tools.

Regarding our choice of tools:
DEMO was selected not only because it is a representative LAP-driven tool but also because the current research is related to the on-going development of DEMO.
Semantic Analysis and Norm Analysis have been selected as well-established semiotic tools which are also usefully relatable to DEMO [Shishkov et al. 2003-1].
Petri Net has been selected as one of the most popular and proven workflow management tools [Aalst and Best 2003].
ORM has been selected as a well established factual tool which is also adequately combinable with DEMO [Dietz and Halpin 2004].
Therefore, by analyzing this set of Business Process modeling tools, we hope not only to achieve a representative view over the state-of-the-art but also to inherit knowledge useful for the current research.

As for formal Business Process modeling tools, they are outside the direct scope of the current research, as it could be inferred from Chapter 1. For this reason, we have not elaborated on them in our classification. However, we acknowledge their great relevance to the Business Process modeling domain; moreover, we envision considering formal tools in our future work, following this research. For this reason, we will briefly present our view on some relevant values of formal Business Process modeling tools.

This outline follows in the paragraphs below while the (more-detailed) consideration of semi-formal Business Process modeling tools (which are to actually concern the further content of this thesis) is in Sub-section 3.1.1.

**Formal Business Process modeling tools**

Applying mathematical methods in support of activities concerning Business Process modeling has been considered to be promising mainly because of its reliability resulting from the rigorous foundations of such methods; this can be seen from diverse areas of such application, including Functional Analysis, Statistical Synthesis, Model Checking, Constraint-Based Design, and so on [Cramer 1999; Thierauf 1988; Bordini et al. 2003; Lakmazaheri and Rasdorf 1989; Levin and Rubin 1997; Freund 2003; Shishkov 1999]. In order to briefly elaborate on the role of formal tools in current Business Process analysis and modeling, we will outline below the basic features and advantages of three of the formal approaches (relating to the above-mentioned applications) most relevant to this research: Statistical Analysis and Synthesis, Functional Analysis, and Constraint-Based Design. As already stated, the role of this outline is just informative since the current research does not directly consider formal tools. However, as stated also, such tools are to be considered, concerning the agenda for further continuation of the research.

_Statistical Analysis and Synthesis_ [Cramer 1999; Levin and Rubin 1997; Freund 2003] are associated with two basic types of problems in examining stochastic signals and systems: problems of analysis and problem of synthesis. A Business/Information System is built in accordance with a determined _criterion of making decision_ and works on an algorithm evolving from this criterion.

The problems of analysis could be solved applying the methods of Probability Theory and Theory of Random Processes [Freund 2003], with the presumption that mathematical probability models of the investigated processes are fully determined (all necessary probability distributions are completely determined). More precisely, Probability Analysis of the output of a system should be performed when both the characteristics of the system and the probability description of the input process are determined.

However, a more complex and frequently arising is the situation concerning a model with a priori uncertainty. This is the case when the probability model of the process (operating within a Business System) is unknown or determined incompletely. Hence, the gist of a number of methods (relating to Mathematical Statistics) for problem solving in business statistics concerns the overcoming of a priori uncertainty. In Statistical Synthesis this issue is approached, by considering the construction of _algorithms_ that process observed experimental data in order to achieve the desired _statistical inferences_. Statistical Synthesis, as a construction of an _optimal_ system or an algorithm for processing, is connected primarily with the concept of _optimization_.

32
Functional Analysis [Thierauf 1988] is associated with the possibility of reducing the solution space by automatically deriving functions relevant to a task of fulfilling a business goal. In Mathematics, the formulation of functionality is via functional equations while in Computer Science - via functional programming.

In conducting Functional Analysis, it is essential to think in terms of the goals; a goal statement should be presented as a functional expression of what is the problem. Then, it is necessary to exhaust all possible consequences, arriving in this way at a business invariance.

In Functional Analysis, one should always start from 'the general' (consider a model bank, for example) and then move to 'the specific' (for instance, a particular function embeddable in the functional class).

Hence, identifying a function from an appropriate class is a crucial issue in conducting formal Functional Analysis. Once we have identified the class of functions (type of functional class) or models to embed the problem in, we could select an appropriate identification method. The method should reconcile the data with the class and identify/propose the most fitting function or model from the class. For example, it might be that in the context of the addition of economic effects, a Cauchy functional is the mandatory identified function type. Then it is necessary to judge relationships which enable the best possible prediction of one or more variables in terms of others.

Experimental mathematics could be useful in this regard as well as relevant mathematical software products such as Maplesoft [MAPLESOFT].

Constraint-Based Design is associated with the constraint-based approach for engineering design [Lakmazaheri and Rasdorf 1989]. The approach has emerged mainly as a consequence of advances in Artificial Intelligence [Thierauf 1988].

Engineering design is viewed there as an information conversion process in which information in the form of requirements is converted into description of an artefact that meets the requirements. The starting point is however the emergence (and realization) of a need. The formulation of requirements comes as a result of this. The requirements themselves describe the desired functional and behavioural characteristics of an artefact. So, according to the principles of constraint-based design, the recognition and formulation of the design requirements, and synthesis of an artefact that satisfies the requirements, together constitute the design process.

The compositional elements of an artefact are objects. Its structure refers to the relations among objects. These relations are considered as constraints in constraint-based design. Design requirements, hence, concern the constraints. Thus, in one sense, the design process can be viewed as a process of identifying, formulating, and satisfying constraints. Constraint-based design actually focuses on the automation of this process. Such an automation requests:

(1) a language for representing constraints, and
(2) a mechanism for satisfying constraints.

Constraint-based design is characterized by orthogonality of the representation of constraints and their processing. This means that modeling a problem (using constraints) does not at all concern the task of satisfying these constraints.

Further on, we continue with the analysis, focusing on semi-formal Business Process modeling tools.
3.1.1 Semi-formal Business Process Modeling Tools: Analysis

We will briefly consider (in this sub-section) each of the (types of) semi-formal Business Process modeling tools mentioned at the beginning of the current section. Then we will conclude in Sub-section 3.1.2 about their strengths concerning the integration between Business Process modeling and software specification.

3.1.1.1 The Communication perspective, LAP, and DEMO

Applying a Communication perspective in approaching a Business System, is motivated by the importance of grasping not only the structural, factual, and dynamic Business System aspects but also the communicative aspect [Searle 1969]. This was already stressed upon in the Chapter 1. It is well known that one of the most sound and popular theories behind this issue is LAP [Winograd 1988]. As mentioned in the Introduction, LAP is a theoretical orientation towards approaching the modeling of Business Processes by emphasizing the importance of interaction and communication. The theory recognizes that language is not only used for exchanging information, as in reports (for example), but that language is used also to perform actions, as in promises or orders (for example). Such actions are claimed to represent the foundation of communities and organizations, and must be understood to create effective Information Systems. For this reason, it is claimed by us that adequately capturing the communication aspects, characterizing the considered Business System(s), would contribute to the creation of sound and complete Business Process models.

We consider in our analysis a Business Process modeling tool which is not only consistent with the LAP theory but also capable of providing other aspect models expressed in sound graphical notations. This tool is DEMO (Design and Engineering Methodology for Organizations). DEMO [DEMO; Van Reiswound et al. 1999] is a methodology for understanding, analyzing, (re)designing and (re)engineering Business Processes. Among the reasons for claiming that DEMO is among the most representative and practically useful LAP-based Business Process modeling tools, are: the completeness of DEMO, its ability to capture the essence of Business Processes [Dietz and Habing 2004], its sound graphical notations. This sub-section elaborates on these issues after considering briefly the conceptual background of DEMO.

Fundamental concepts of DEMO

DEMO’s underlying theory about organizations is rooted in the Language/Action Perspective [Winograd 1988; Flores and Ludlow 1980], Organizational Semiotics [Liu 2000] and Philosophical Ontology [Bunge 1979]. DEMO reveals the ‘construction’ and ‘operation’ of an organization, contrary to the current function and behavior-oriented approaches. It is characterized by three major features:

- a white-box architecture of actors, production and coordination;
- the extraction of the essence of business processes from their realization;
- the Transaction pattern.

Actors, production, coordination

Like every other system (e.g. an alarm clock or a racing car), the functional behavior of an organization is brought about by the collective working of the constructional components. The construction and the working of a system are most near to what a system really is, to its ontological description [Bunge 1979]. An organization is defined as a (discrete dynamic) system in the category of Social Systems. This
means that the elements are social individuals or actors, each of them having a particular authority to perform production acts (P-acts) and a corresponding responsibility to do that in an appropriate and accountable way. The structure of an organization consists of coordination acts (C-acts), i.e. the actors enter into and comply with commitments regarding the performance of P-acts. The generic white-box organizational model (Figure 3.1) consists of: the actors, the P-world, and the C-world [Dietz 1999].

![Figure 3.1: The white-box model of an organization](image)

By performing P-acts, the organization does what it is supposed to do according to its function. C-acts serve to coordinate and control the performance of P-acts.

**Essence and realization**

In DEMO, three perspectives on an organization are distinguished, called essential, informational, and documental [Dietz 1994], as exhibited in Figure 3.2:

![Figure 3.2: The three perspectives on organizations](image)

- **essential** - the organization viewed as a system of authorized and responsible actors that create new original facts;
- **informational** - the organization viewed as a system of information processors that remember facts and derive new facts from existing ones;
- **documental** - the organization viewed as a system of formal operators that collect, transport, store, copy and destroy representations of facts.

Take for example the process of withdrawing money from a bank account using an ATM machine. Think of observing this process through essential, informational or documental ‘glasses’ as a metaphor. Looking through documental ‘glasses’ we see someone inserting a card into a machine, pushing buttons on a keyboard and finally getting out the card and other pieces of paper. Nothing with respect to the information on it or the purpose for which they are used, is seen. Looking at the same process through informational ‘glasses’, we see someone providing information to an ATM system: a PIN code and specification of an amount of money. Also, the machine provides information if withdrawal is possible to the customer. We see that the machine outputs money and receipts. Looking through essential ‘glasses’ shows responsible actors, their actions and interactions. A customer requests a bank to withdraw money from an account. The bank decides to do this and states that the money is withdrawn. Further on, the customer accepts it.
The Transaction pattern

Production acts and Coordination acts appear to be performed in particular sequences or chains that can be viewed as paths through a generic pattern called the (business) Transaction [Dietz 1999]. It is exhibited in Figure 3.3:

![Diagram of the Transaction pattern]

**Figure 3.3: The Transaction pattern**

A Transaction is a finite sequence of C-acts between two actor-roles, the customer and the producer. It takes place in three phases: the order phase (O-phase), the execution phase (E-phase), and the result phase (R-phase). O-phase is a conversation that starts with a request by the customer and that, if successful, ends with a promise by the producer. E-phase basically consists of the performance of the P-act by the producer. R-phase starts with the statement by the producer that the requested act is performed and ends, if successful, with the accept by the customer. The whole pattern of a transaction is represented by one symbol in the so-called Coordination Structure Diagram (CSD). Figure 3.4 exhibits CSD for the money withdrawal example. The two boxes represent the two actor roles involved: A0(A1) is the customer(producer). The small black box indicates that A1 is the producer of T1 (and consequently A0 is the customer). The successful result of a transaction T1 is the P-fact ‘withdrawal W is performed’ where W is constituted by the account, the amount and the time.

![Diagram of the money withdrawal example]

**Figure 3.4: CSD of the money withdrawal example**

**Strengths and limitations of DEMO**

Based on our studying DEMO, we conclude that among its relevant strengths are the LAP theoretical foundation, the consideration of more than one Business System perspectives, its powerful graphical notations as well as the potentials for an indirect mapping towards a UML-driven software specification model; the studies of Shishkov
and Dietz [Shishkov and Dietz 2004-3], directed towards the derivation of use cases based on DEMO models, are a manifestation of the tendency towards further developing DEMO, making it usable in a broader perspective. As for the limitations of DEMO in the context of the current research, we claim that DEMO alone is not capable of soundly and completely aligning Business Process modeling and software specification. Our arguments are in two directions. Firstly, DEMO is a Business Process modeling tool and is hardly adaptable for software specification purposes (it does not support the modeling of complex interactions, for example). Secondly, given the requirement for a component-based software specification (refer to the Introduction) DEMO’s output is not straightforwardly transformable into a useful software specification input, mainly because DEMO is not founded on the OO/CBD paradigm.

3.1.1.2 The Structural perspective, Organizational Semiotics, and Semantic/Norm Analysis

It is considered useful applying the Semiotic theory [Coble and Jansz 2001], regarding issues connected with the analysis and modeling of Business Processes. Actually, a branch of Semiotics is considered, namely – Organizational Semiotics (OS), and in particular two OS methods: Semantic Analysis and Norm Analysis [Stamper 2000; Stamper 1996; Liu 2000]. OS focuses on the nature, characteristics and behavior of signs. The term ‘Organizational Semiotics’ was officially coined in the first international workshop in Twente, The Netherlands, in 1995 after a long time of research by colleagues who have worked in the many fields of organizational studies and information systems. This sub-section considers briefly some essential issues related to the OS theory.

OS – basic concepts

Peirce founded Semiotics as the ‘formal doctrine of signs’ [Peirce 1998]. A sign is defined as something that stands to someone for something else in some respect or capacity. OS and the analytical methods [Stamper 1992,6,7; Stamper 2000; Stamper et al. 1997, 2000; Liu 2000; Liu et al. 1999, 2001] offer a theory to understand business organizations, with or without the computerized Information Systems. Organizations are deemed as systems where signs are created, transmitted, and consumed for business purposes. Stamper and his school of OS argue that in contrast to the concept of information, signs offer a more rigorous and solid foundation to understand Information Systems. For example, within a business context, a bank note is much more than a piece of colored paper with digits on it. It stands for the bank note holder’s wealth and ability to pay, as well as the issuing bank’s authority and credibility, and much more. Large quantity of underlying social relationships and behavior possibilities are attached to these business concepts; oversimplifying them into pure digits would be dangerous. On one hand, computers can only process and manipulate such digits; on the other hand, the underlying meanings and possibilities must be exposed to enable the correct processing. Adopting the concept of sign enables us to study the organization in a more balanced way, taking account of both the technological issues, and the human and social aspects of information resources, products, and functions.

OS adopts a subjectivist philosophical stance and an agent-in-action ontology. This philosophical position states that, for all practical purposes, nothing exists without a perceiving agent and the agent engaging in actions.
Chapter 3

Ronald Stamper adopts the concept of affordance from the perceptual psychologist James Gibson, who defined 'the affordances of the environment' as 'what it offers the animal, what it provides or furnishes, either for good or ill...' [Gibson 1979]. Based on the theory, since a person perceives things by recognizing what he can do with them or to them, a thing can be defined as an 'invariant repertoires of behaviors, either substantive affordances or social norms' [Stamper 2000] that are available to the responsible person. For example, in the context of a university library, a book affords to be borrowed by a library user.

Borrowing a book is a potential ability, which may or may not be implemented in the reality. However, once it is implemented, new possibilities may emerge. For example, a borrowed book may be retained or returned to the library by the user. Under certain circumstances, the library may also call it back. This shows that affordances have dependency relationships among them. In Organizational Semiotics such a relationship is called Ontological dependency.

We may schematically show this relationship as following, with the antecedents on the left side and the dependencies on the right, and the solid line denotes the ontological dependency:

\textit{book} \rightarrow \textit{borrow} \rightarrow \textit{return}

Ontological dependency does not only show the logic relationship between the concepts. What's more important is that it shows the dependencies get their meaning from the existence of the antecedents. Since the existence of the dependencies would not be possible without the existence of the antecedents, the lifecycle of the dependencies is always included by that of the antecedents. The existence of the antecedents thus forms a context for the dependencies. For example, talking about returning a book without referring to the fact that the book was previously borrowed from the library would be off the topic.

Two essential OS methods considered in this research are the \textit{Semantic Analysis Method} and the \textit{Norm Analysis Method}. These methods are briefly discussed below.

\textit{Semantic Analysis}

The Semantic Analysis Method is fundamentally based on the Semiotic theory that has been discussed above. This method is a method for elicitation and specification of user requirements. It considers the signs created by members of an organization. Semantic Analysis is theoretically founded in OS [Stamper 1997] and the semiotic framework. The method has been applied in many fields such as user requirements for Business Systems, organizational analysis, legal document design, and analysis and design of Computer Systems [Liu 2000; Liu et al. 1999; Shishkov et al. 2004, 2003-1]. The Semantic Analysis is conducted usually in four steps, outlined below, and the final result is a semantic schema, called 'Ontology chart'.

Taking into account that Semantic Analysis deals with analysis of documents and conversations, the first step that is to be realized, is to gather relevant data and understand the problem. This can be called 'problem statement'.

The second step is to produce a list of semantic units such as verbs, nouns, adjectives and adverbs. These semantic units may be used to describe human agents and their respective patterns of behavior.

The third step is to further analyze the semantic units by linking them together according to their relationship in terms of generic-specific positioning. This is shown graphically from the left to right on the Ontology chart.

The fourth step should bring together all the linked semantic units into a coherent whole, which produces a complete semantic model. The model is represented graphically through the Ontology chart.
Norm Analysis

When studying organizations from the perspective of agents' behavior it is necessary to specify the norms based on which this behavior is realized. Norms [Stamper et al. 1997] are the rules and patterns of behaviour, either formal or informal, explicit or implicit, existing within a society, an organization, or even a small group of people working together to achieve a common goal [Liu et al. 2001].

Norms are determined by Society or collective groups, and serve as a standard for the members to coordinate their actions. An individual member uses the knowledge of norms to guide his or her actions. If the norms can be identified, the behaviours of the individuals, hence their collective behaviours, are mostly predictable. From this perspective, to specify an organization can be done by specifying the norms [Stamper 1992].

Four types of norms exist, namely evaluative norms, perceptual norms, cognitive norms and behavioural norms. Each type of norms governs human behaviour from different aspects. In Business Process modeling, most rules and regulations fall into the category of behavioural norms. These norms prescribe what people must, may, and must not do, which are equivalent to three deontic operators 'is obliged', 'is permitted', and 'is prohibited'. Hence, the following format is considered suitable for specification of behavioural norms:

whenever <condition>
  if <state>
    then <agent>
    is <deontic operator>
    to <action>

It is essential to recognise that norms are not as rigid as logical conditions. If a person does not drink water for certain duration of time he cannot survive. But an individual who breaks the working pattern of a group does not have to be punished in any way. For those actions that are 'permitted', whether the agent will take an action or not is seldom deterministic. This elasticity characterises the Business Processes, therefore is of particularly value to understand the organizations.

A Norm Analysis is normally carried out on the basis of the results of the Semantic Analysis (for information on Semantic Analysis interested readers are referred to [Liu 2000]). The semantic model delineates the area of concern of an organization. The patterns of behavior specified in the semantic model are part of the fundamental norms that retain the ontologically determined relationships between agents and actions without imposing any further constraints. However, Norm Analysis could be successfully related also to other modeling tools, e.g UML activity diagram, Petri net, etc.

In general, a complete Norm Analysis can be performed in four steps: 1) Responsibility analysis; 2) Proto-norm analysis; 3) Trigger analysis; 4) Detailed norm specification. Responsibility Analysis enables one to identify and assign responsible agents to each action. The analysis focuses on the types of agents and types of actions. In an organization, responsibilities may be determined by the organizational constitution or by common agreements in the organization. Proto-norm Analysis helps one to identify relevant types of information for making decisions concerning a certain type of behavior. After the relevant types of information are identified, they can be used as a checklist by the responsible agent to take necessary factors into account when a decision is to be made. The objective of this analysis is to facilitate the human decisions without overlooking any necessary factors or types of information. Trigger Analysis is to consider the actions to be taken in relation to the
absolute and relative time. The absolute time means the calendar time, while the relative time makes use of references to other events. The results of trigger analysis are specifications of the schedule of the actions. The detailed Norm Specification concerns the specification of norms in two versions, a natural language and a formal language. The purposes of this are (1) to capture the norms as references for human decision, and (2) to perform actions in the automated system by executing the norms in the formal language.

For those norms identified in the Business Processes, some refers to the major authorities and responsibilities of the major figures in the organizations. These norms govern some trivial, relatively less important norms or those of lower priorities, from the perspective of organizational functionalities [Shishkov et al. 2002-1]. This strongly suggests the possible hierarchies exist not only in the organization structure, but also in the norms. The terms framing norms and contractual norms are used to express the hierarchies [Liu et al. 2001].

**Strengths and limitations of OS**

Studying Semantic/Norm Analysis, we have identified a number of relevant strengths of these Semiotic tools. Among them are the following three strengths: Firstly, Semantic Analysis is powerful in situations in which it is necessary to put some unstructured information in order. This is an unavoidable task in any software project. Secondly, Norm Analysis is powerful in situations in which it is necessary to specify rules and also to relate a number of rules to each other. Hence, Semiotic Norms could be much useful in both Business Process modeling and software specification – both tasks include consideration of rules. Thirdly, Semantic and Norm Analysis are founded in the OS theory; it is a well established theory relevant to both Business Process modeling and software specification. As for the limitations of these two Semiotic tools, we again claim (as in discussing DEMO) that they alone are not capable of soundly and completely aligning Business Process modeling and software specification. Among our arguments are that these tools do not relate to the dynamic Business System perspective (Figure 1.2) and also they do not support a component-based modeling. Therefore, a Semiotic model could not be reflected in a straightforward way in a component-based software specification model (as it is according to the requirements in the context of the current research (Chapter 1)).

**3.1.1.3 The Dynamic perspective, Workflow Management, and Petri Net**

Using workflow-like diagram techniques has been popular with the modeling community for many years. Workflow developed further on to become a sound and useful approach to the modeling of Business Systems, especially their dynamic aspects [Rolland 1996]. According to the Workflow Management Coalition [WMC]: the evolution of Workflow Management consists of the automation of business procedures or 'workflows' during which documents, information or tasks are passed from one participant to another in a way that is governed by rules or procedures. Some key benefits of modeling Business Processes through workflows are:

- A workflow-based Business Process model is easily usable in the automation of a Business Process. As it is well known, automation of many Business Processes results in the elimination of many unnecessary steps.
- Flexibility: if software control is used over the Business Processes, their re-design (if necessary) would be easy (if their model is workflow-based), being in line with changing business needs.
Better process control: improved management of Business Processes would be achieved through standardizing working methods.

Improved customer service: consistency in the processes leads to greater predictability in the levels of response of customers,

Business Process improvement: a focus on Business Processes leads to their streamlining and simplification.

As it is widely agreed upon, Petri Net [Petri Nets] is most popular among the workflow modeling tools. For this reason, we will consider this particular tool.

**Petri Net – basic concepts**

According to the Petri Net concepts [Petri Nets; Peterson 1981; Aalst and Best 2003; Shishkov and Barjis 2002], any Business Process can be viewed as a collection of processes, where a process can be described as ‘a set of identifiable, repeatable actions which are some way ordered and contribute to the fulfilment of an objective’. These processes change as organisations evolve over time in response to their business environments. The focus on the Business Process is important in this context, in order to design, maintain and improve the way business work effectively and efficiently.

So as it becomes obvious, if the business modeling is to be reflected in the modeling of software systems (which are to support the original Business System), there are two aspects to be studied [Shishkov and Barjis 2002]. Firstly, it is essential to model consistently the system, eliciting its static issues. Secondly, it is necessary to capture (via a tool(s)) the system dynamics, representing. As for capturing the dynamics of the Business Processes to be supported by software systems, Petri Net would be of great value [Petri Nets]. Petri Nets have well supported mechanisms to allow modeling, analysis and, if necessary, simulation (execution) of systems. Petri Net is a well known and widely used modeling technique [Aalst and Best 2003]. The Petri Nets are formalism and a graphical language for the design, specification, and verification of systems. In order to better understand Petri Nets, there are some typical examples of Petri Net depicted in Figure 3.5. These examples are especially chosen to demonstrate Petri Nets’ application and way of modeling, while dealing with processes in series, parallel, and conditional or alternative processes. It should be noted that rectangles colored in grey indicate that these transitions are enabled.

![Figure 3.5: Typical Petri Net examples](image)

In Figure 3.5, there are the following situations represented before and after the firing of transitions from the left to the right. The first example shows an ordinary process having one transition. The second example represents parallel processes. When the enabled transition fires, it enables two other transitions in parallel. The
third examples represents alternative processes. This examples shows that only one of the two processes, for which the condition is true, will be undertaken. The fourth and last example represents synchronization or AND - join processes. In this situation it is necessary to finish both processes completely before starting the following one.

We will not deepen anymore regarding the consideration of Petri Net because there are numerous literature and other sources (among them the ones cited above) which are informative and well known to the business modeling community.

**Strengths and limitations of Petri Net**

Although Petri Net is a sound workflow tool which is both well-founded theoretically and well-proven in practice, we claim that Petri Net is not capable alone to adequately solve the business-software-alignment problem. It is well known that Petri Net is usable as a workflow tool both in the Business Process modeling and software specification contexts. However, Petri Net is not soundly concerned with other essential Business System modeling perspectives, such as the structural and the communication perspectives (mentioned before). Petri Net's lacking the capability to consider the structural Business System modeling perspective means thus that the tool is incapable of considering components, which is another related limitation.

### 3.1.1.4 The Data perspective, LFP, and ORM

The stress upon information, and especially its factual aspects, relates to the so called 'Fact perspective' in considering a Business System. Currently, there is hardly any general consensus about the meaning of the term 'information' [Dietz 2004]. However, there are some widely agreed upon perspectives regarding this. The prevailing perspective on information is the one that is taken by the conceptual modeling approaches to databases and Information Systems. The Entity–Relationship (ER) Model [Batini et al. 1991] is the most popular one of these. The model has further evolved through approaches like NIAM [Leung and Nijsen 1988] and ORM [Halpin and Wagner 2003; Halpin 1998]. The mentioned approaches have mechanisms for normalization directed towards correction of the errors that people inherently make when applying the ER-modeling. These approaches are language based and according to Jan Dietz [Dietz 2004; Dietz and Halpin 2004] are defined to belong to the so called Language-Fact Perspective (LFP). We are going to consider the ORM approach as the LFP-based one that is closest to the current modeling achievements. This is not only because ORM is language based but also because it has been recently developed, taking account on the latest modeling accomplishments.

**ORM – basic concepts**

ORM (Object-Role Modeling) is a fact-oriented approach for performing information analysis at the conceptual level [ORM]. ORM is used mainly in the context of the design of information systems. However, ORM could have much wider application potential. Applying ORM in the modeling of Business Processes is considered to be useful [Dietz and Halpin 2004]. ORM could be of use in Business Process modeling, particularly when the factual issues are to be precisely addressed.

To develop (through ORM) a conceptual design, seven steps should be taken.

Firstly, examples of the kinds of information required are verbalized in natural language. This results in a set of ordered elementary facts. As a quality check, it is to be ensured that objects are well identified. Basic objects are either values or entities.
For example, if R. Adams (emp. Nr. 128) works for the Department of Computer Science at a university, and if we have declared reference schemes as follows: Academic (empNr), Dept (name), then we arrive at the following fact (f1): f1 Academic 128 works for Dept 'Computer Science'. This fact is an instance of the following fact type (F1): F1 Academic works for Dept.

Secondly, a draft diagram of the fact types is to be drawn and a population check is to be applied (Figure 3.6). The diagram is much helpful for structuring in a sound way the initial input data.

![Diagram](image)

*Figure 3.6: ORM draft diagram of fact types*

Thirdly, some checks are to be performed, for examples checking for entity types that should be combined. If, for instance, we have three fact types: Professor obtained Degree from University; SeniorLecturer obtained Degree from University; Lecturer obtained Degree from University, the common predicate suggests that the entity types Professor, SeniorLecturer, Lecturer should be collapsed to the single entity type Academic. Hence, we introduce the fact type: Academic has Rank. All these realized updates should be reflected in the diagram.

Fourthly, uniqueness constraints must be added. Uniqueness constraints are used to assert that entries in one or more roles occur there at most once. For example, the internal uniqueness constraints on the binary fact types assert that each academic has at most one rank, holds at most one chair (and vice versa), works for at most one department, and has at most one employee name.

Fifthly, mandatory role constraints should be added. A role is mandatory for an object type if and only if every object of that type which is referenced must be known to play that role [ORM].

Sixthly, value, set comparison, and subtyping constraints are to be added. For example, Rankcode is restricted to { 'P', 'SL', 'L' } (coming from Professor, Senior Lecturer, and Lecturer, respectively).

Seventhly, a final check is to be performed. Additional constraints might be added during the check. For example, if university departments are considered and if each of them has (compulsory) two units within its budget – teaching and research ones (called here for convenience 'teaching budget' and 'research budget'), we might need to schematize this. This is to be done by indicating that each department that is included in the population of that role (role, related to the budget) must appear there twice. In conjunction with the other constrains, this ensures that each department has both its teaching and research budgets recorded.

Once the global schema is drafted, various optimizations can usually be performed to improve the efficiency of the logical schema. And further on, simple algorithmization is used to group the fact types into normalized tables.

We are not going to further deepen in discussing the ORM approach since the goal of this section is to analyze a number of approaches related to business process modeling, and draw conclusions. However, more information on the ORM approach can be found in [ORM].
Strengths and limitations of ORM

ORM is sound and complete as far as we consider the data perspective. The data-related issues could be soundly matched, using ORM, between Business Process and software specification models. Nevertheless, we claim that the lack of adequate consideration of other essential Business System modeling perspectives (such as the dynamic (workflow) perspective) makes ORM unusable (alone) for adequately bridging Business Process modeling and software specification.

3.1.1.5 The Simulation perspective, Discrete Event Simulation, and ARENA

An essential issue in the Business Process modeling task is the realization of sound validation of the dynamics of developed business process models. Simulating the dynamic issues concerning a Business System is claimed to be useful in this regard [Shishkov et al. 2003-2]. Simulation not only allows for good illustration and visualisation of the dynamics of a Business System but also is a good basis both for system developers and users to grasp the system under development: the developers could be facilitated by the visualisation in discovering some shortcomings of their models (under development), also – they could identify other modelling options; users would be facilitated to easier understand the model. Simulation is even further essential in cases when a business model is to be used as a basis for software specification. Then, it is of significant importance to trace precisely all essential dynamic issues related to Business Process model in order to make sure that the performance of the software system will be properly designed, avoiding the risk of a software systems functioning inadequately. A way to reliably and illustratable trace the interactions within a Business System is via Discrete Event Simulation [Shishkov et al. 2003-2]. Analyzing systems in such a way has proved its methodological advantages [Barjis et al. 2002], mainly related to the easy way (in this case) to model and execute (simulate) discrete dynamic events (Discrete Event Simulation is well known to the public and for this reason we will not deepen further in introducing it). Next to that, the possibility to represent and visualise the system under study in sound graphical notations is of particular importance since the developers would have the chance to grasp the peculiarities of the system as a whole. Another advantages feature regarding Discrete Event Simulation models is that they (as most of the types of simulation models) are easily understandable not only for the system developers but also for potential users. For all these reasons, we claim that it is feasible to expect that such models can be helpful for grasping the dynamics of a business system being modeled.

We consider in particular, the ARENA [ARENA] simulation tool as one of the appropriate tools that can be used for business management simulation systems [Shishkov et al. 2003-2].

ARENA – basic concepts

Based on the above considerations, we regard the ARENA simulation tool to be one of the most appropriate tools that can be used for Business Processes simulation. An advantageous feature of ARENA is its good alignability to workflow modeling tools, such as Petri Net and Activity diagram [Barjis and Shishkov 2001-1]. ARENA has also rich and well-formulated animation facilities that make simulation models interesting, attractive, and easy for training people and demonstration purposes [Barjis and Ilkov 2000].
ARENA is a general simulation tool which can be used for Business Process simulation as well. There is an ARENA business edition which is made especially for this purpose. The way that ARENA represents an activity, action, or atomic process is as follows. Each activity (action or process) in ARENA can be represented as three interrelated elements like Arrive, Server and Depart; or Enter, Process and Leave.

In order to introduce the ARENA simulation tools, there are some logical models of the ARENA discussed in this section. These logical models represent typical features of process modeling such as causal and conditional interrelations, synchronization, parallelism, sequence and choice. The fragments in Figure 3.7 show how these situations can be captured by the ARENA modeling tools. The first fragment (Figure 3.7 a) represents a single and simple process having a set of consecutive activities. The second fragment represents a process consisting of two parallel activities shown by ‘Duplicate’ and synchronization shown by ‘Batch’. The third fragment represents a relatively more complex process consisting of three activities in parallel shown by ‘Duplicate’ and a conditional activity shown by ‘Choose’. This fragment also contains a synchronization ‘Batch’.

The Choose module provides entity branching based on the ‘If’ conditional rule in conjunction with the deterministic ‘Else’ and ‘Always’ rules. Branch destinations are defined by connectors or by specifying a label destination.

![Diagram of process simulation models using the ARENA notations](image)

Figure 3.7: Examples of process simulation models using the ARENA notations

Actually, the above fragments include the basic options of a workflow such as causal, conditional and optional links, and also parallel initiation of several actions and synchronization of several actions before succeeding further. These are just the basic workflow patterns; more workflow patterns have been identified [Aalst et al. 2000]. However, we are not going to discuss them because the (introduced) three patterns appear most frequently in real-life situations.

**Limitations of ARENA**

Being a Discrete Event Simulation tool, ARENA could be useful in modeling and demonstrating the dynamics of a Business System. However, other Business System perspectives (such as the structural one) are inadequately considered by ARENA. Moreover, ARENA’s modeling part lacks sound methodological guidelines on how to analyze a business reality and how to reflect this in an ARENA model. Therefore, we claim that using ARENA alone would not bring solution to the problem of aligning Business Process modeling and software specification.
3.1.2 Business Process Modeling Tools: Conclusions

It could be seen, from what has been discussed so far in this section, that each of the analyzed Business Process modeling tools is built over a particular Business System modeling perspectives (for example, Petri Net is built on the foundation of the dynamic perspective); our observation is that this is valid for Business Process modeling tools in general. Furthermore, neither of the analyzed tools has been designed with the purpose of its being directly relatable to software specification. Thus, we have resulting from this:

- Lack of a thorough (multi-perspective) scope: a business model which is constructed considering just a particular perspective, would in any case be incomplete as a foundation for a further specification of software. For instance, Petri Net is not exhaustive regarding structural and data-related issues, ORM misses the dynamic perspective, ARENA does not provide at all a structural view, and so on. DEMO, however, spreads across different perspectives. This is an advantage even though DEMO could be more powerful with regard to the dynamic perspective (improving the graphical representation and adding simulation facilities would be a solution). Therefore, applying any of these currently popular Business Process modeling tools (analyzed in this chapter) would in any case result in a model which is incomplete with respect to basing on it a further software specification.

- Lack of a 'bridge' towards the current software specification standards (such as UML and XML), which makes the mentioned Business Process modeling tools practically hardly usable for founding software design, because their Business Process modeling output is not translatable in a straightforward way into (currently used) software specification notations.

Next to that, none of the studied Business Process modeling tools is fully consistent with the OO/CBD paradigm (in our view, this applies to most of the current Business Process modeling tools). This is an obstacle with regard to the goal (Chapter 1) of further mapping a Business Process model towards component-driven software specification models.

Hence, what is offered by the currently popular and widely used Business Process modeling tools is insufficient with regard to the goal of providing a Business Process modeling output that is adequately mappable towards further component-based software specification models.

3.2 SOFTWARE SPECIFICATION TOOLS

Currently, software systems are built by a multitude of designers/developers distributed across different locations and possessing different types of knowledge and expertise. Therefore, it is crucial that the software artefacts are being designed using common, unified, standard tools and environments that are known by as broad circles as possible [Shishkov and Dietz 2002-4]. The Unified Modeling Language – UML is becoming de facto the standard language for modeling software systems, widely recognized by both researchers and practitioners [Shishkov and Dietz 2002-3]. For this reason we will consider basically in this section the UML. Another argument in this direction is that the software community has succeeded to integrate (through UML) the most successful results from the last years, bringing together some of the outstanding representatives of the community. Therefore, considering the Unified Modeling Language, one guarantees a sufficiently complete grasp over the 'software design arena'. Anyway, for the sake of completeness, we have included in our
analysis also some other representative tools (some of them related to UML) concerning essential aspects of the software specification tasks, possessing prestige and popularity in the mentioned community, and having proven practical applicability. These tools are Tropos [Tropos], Catalysis [D'Souza and Wills 1998], and KobrA [Atkinson et al. 2001].

3.2.1 Software Specification Tools: Analysis

Further on, we will firstly consider UML, and then discuss briefly Tropos, Catalysis, and KobrA (above mentioned). After this, we will conclude (in subsection 3.2.2) about their capabilities in the direction of soundly deriving a software specification model based on Business Process modeling.

3.2.1.1 UML

UML is a language for specifying, visualizing, and documenting models of software systems, including their structure and design [OMG]. Several issues are considered below in this regard:

- the fundamental concepts and ideas behind UML as well as its application through a development process;
- the essential UML diagrams representing the fundamental issues in UML;
- since UML offers an appropriate 'interface' between requirements engineering [Loucopoulos and Karakostas 1995] and software design, namely the Use Case Model, this model is especially elaborated.

UML - background

Object-oriented analysis and design methods appeared in the late '80s [Shlaer and Mellor 1989]. UML is the successor of this multitude of methods. However, it should be taken into consideration that most methods consist of both a modeling language and a process [Fowler and Scott 2000] as well as that UML is just a modeling language, UML has no notation of process. Therefore, UML cannot be considered as a method. As a modeling language, UML is well-defined, expressive, powerful, and applicable to a wide spectrum of problem domains [Booch et al. 1999]. UML is independent of development process – actually, any process can be used with the UML. However, the latest developments of the Rational Unified Process – RUP [Kruchten 2000], position it as the process usually applied with UML. For this reason, the software community considers the UML-RUP environment as standard as long as UML-based software design is concerned.

As already stated, UML can be used for specifying, visualizing, and documenting models of software systems, including their structure and design. UML defines twelve types of diagrams (some of them are considered further on in this section), divided into three categories: four diagram types represent static application structure; five represent different aspects of dynamic behavior; and three represent ways in which application modules can be organized and managed [OMG].

The primary design goals [UML Resource Center] of the UML are as follows:

- Provide users with a ready-to-use, expressive visual modeling language to develop and exchange meaningful models.
- Furnish extensibility and specialization mechanisms to extend the core concepts.
- Support specifications that are independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of the object tools market.
Chapter 3

- Support higher-level development concepts such as components, collaborations, frameworks and patterns.
- Integrate best practices.

Further on, it is elaborated on the essential UML diagrams as well as on the use case model considered to be of crucial role, especially in the context of the realization of a UML-driven business/software alignment (what is our goal in the current thesis).

**UML diagrams**

The UML notation includes the Use case diagram, Class and Package diagrams, Collaboration and Sequence diagrams, State diagram, Activity diagram, and Deployment diagram. Below, these diagrams are briefly introduced, excluding the use case diagram which is especially elaborated afterwards.

**Class diagram and Package diagram**

Class diagrams show the static structure of classes and their relationships expressed with the UML notation. Each class on a diagram can optionally show its name, attributes with or without data types, and/or operations with or without arguments. Class diagrams also show the constraints that apply to the way objects are connected [Fowler and Scott 2000]. A class can appear on several diagrams with different presentations; the designer may choose to display only those attributes and operations significant to the context of a diagram. Classes can be grouped and dependencies among such groups — specified. The Package diagram should be used for such purposes.

**Collaboration diagram and Sequence diagram**

These UML diagrams describe how groups of objects collaborate in some behavior [Fowler and Scott 2000]. Collaboration and Sequence diagrams should be used when the developer wants to look at the behavior of several objects within a single use case (the use case concepts are considered further on in this sub-section). Although Sequence diagram and Collaboration diagram play one and the same role in UML, actually developers have to choose which of these diagrams to use, they have specific peculiarities motivating such a choice. Using Sequence diagram, allows modelers to easily see the order in which things occur. The Collaboration diagram is more focused on indicating how objects are statically connected.

**State diagram**

This UML diagram represents the behavior of entities capable of dynamic behavior, be specifying its response to the receipt of event instances. Typically, it is used for describing the behavior of class instances [UML Resource Center]. However, State diagram may also describe the behavior of other entities such as use cases, actors, subsystems. The diagram describes all the possible states a particular object can get into and how the object's state changes as a result of events that reach the object.

**Activity diagram**

An Activity diagram is a special case of a State diagram in which all (or at least most) of the states are action or sub activity states and in which all (or at least most) of the transitions are triggered by completion of the actions or sub activities in the source states. The entire Activity diagram is attached (through the model) to a class or to
the implementation of an operation, or a use case. The purpose of this diagram is to focus on flows driven by internal processing, as opposed to external events [OMG 2000].

Activity diagram is based on ideas from different techniques, for example - state modeling techniques. Unlike the other UML diagrams, Activity diagram is not founded on object-oriented ideas. It is particularly useful in describing behavior, providing a possibility to capture some dynamical aspects of behavior.

One of the strengths of Activity diagram lies in the fact that it supports not only sequential but also parallel behavior. This makes it a great tool for workflow modeling [Fowler and Scott 2000].

Another strength of Activity diagram is that it could be used as a starting point for conducting computer simulation. Models represented with the means of Activity diagram could be easily simulated, using different tools [Barjis and Shishkov 2001-1].

**Deployment diagram**

A Deployment diagram shows the physical relationships among software and hardware components in the delivered system [Fowler and Scott 2000], elaborating on the configuration of run-time processing elements.

**UML - the Use Case Model**

In order to solve the main research problem approached in this thesis, namely the mismatch between the business requirements and the actual functionality of the delivered software application, it is necessary to find out how to consistently base the specification of software on a Business Process model. These issues were already considered in Chapter 1. However, addressing the mentioned problem from the perspective of UML leads directly to the notion of Use Case because, as it is well-known, use cases are modeling constructs that serve to link the application domain (the business world) to the software domain (regarding the UML-based software development).

Ivar Jacobson introduced use cases in 1986, to be applied to requirements analysis [Jacobson et al. 1992]. This was an essential contribution to UML where the use case concept plays a fundamental role. According to the concept, in a use case, a user performs a behaviorally related sequence of transactions in a dialogue with the (software) system [Fowler and Scott 2000], as illustrated in Figure 3.8. Thus, a use case is a typical user / computer system interaction. A use case captures some user-visible function. This view suggests that developers of good use cases identify the users' goals, not the system functions.

![Figure 3.8: Use case concept of Jacobson](image)

In incorporating the use case concept in UML, Jacobson provides description for several use case formalisms. Four essential formalisms are outlined below:

- The basic use case consists of structured text description, including alternative and exceptional behavior. Scenarios may also be used to explain different perspectives of use.
There are two use case stereotypes - <<include>> and <<extends>>, and actor inheritance hierarchies.

There is a contract, specifying an object's interface in detail. Actors may provide a contract that involves multiple use cases. Conversely, a use case may provide a contract that multiple actors use.

A stimulus from an actor will cause the system to leave its current state and carry out some tangible amount of work which is associated with pre- and post-conditions.

Based on these UML-related concepts, Alistair Cockburn further considers the use case concepts, discussing how use cases should be developed and documented. Cockburn discusses the way in which use cases can be represented with varying levels of formality [Cockburn 2000]. In his study, Cockburn further develops some of the concepts concerning use cases. According to Cockburn, a use case describes a system's behavior, as illustrated on Figure 3.9.

![Diagram](image)

*Figure 3.9: A use case as seen by Cockburn*

In his model, Cockburn defines at the most generic level a *system* and *actors* (both having responsibilities and behavior). He is interested just in these actors, situated outside the system, and from them chooses those whose interests should be protected by the system. Cockburn calls these actors *stakeholders*. The *primary actor* (the term "primary actor" is equivalent to the term "user" used by Jacobson) is one of the stakeholders, the one who has the goal and who initiates the system's activity. The system should satisfy this goal but at the same time it should protect the interests of the stakeholders.

The primary actor's *goal* drives the use case. The primary actor should be determined as well as the goal level - a goal may contain sub- and sub-sub-goals. Three most widely used goal levels are suggested – user goal (the goal the primary actor has in trying to get work done), summary goals (involving multiple user goals), sub-function goals (those required to carry out user goals).

Besides determining the goals, it is essential to determine exactly what is the system under discussion. This is called *scope*. Functional scope refers to the services that the system offers. Design scope is the set of systems, hardware and software, that the developer is charged with designing and discussing. It is important that the writer and reader are in agreement about the design scope for a use case. Cockburn suggest as fundamental:

- **Enterprise scope**: the use case describes a person's interaction with an organization.
- **System scope**: the use case describes a person's interaction with hardware/software.
- **Subsystem scope**: refers to situations where we describe how a piece of a system works.

According to the vision of Cockburn, use cases may include also other elements – action steps, scenarios, and preconditions.
The concepts of Jacobson and Cockburn were thoroughly investigated by Shishkov and Dietz [Shishkov and Dietz 2001] from the point of view of their actuality for the development of UML. These two perspectives need to be compared considering the purposes behind them.

Aiming at looking inside a use case, Cockburn has built a concept that allows developers to keep interest not only in the user of the system (the primary actor) but also in the other stakeholders. For this reason, the model of Cockburn seems more complete than the model of Jacobson. At the same time, the graphical representation of the Cockburn model is unsuitable for presenting a multitude of use cases.

The latest developments in Jacobson’s use case concept are put in the perspective of UML. For this reason, instead of focusing on the complete representation of a use case, the concept emphasizes on features that allow developers to show relationships which cover a large number of use cases and actors. This is the main function of the Use case diagram. Thus, Jacobson extracts the gist – actors, pieces of functionality and their relationships.

Jacobson and Cockburn form their use case perspectives, starting from different angles. This indicates that the use case theory needs further exploration in order to provide options for both complete insight and flexible multi-use-case representation.

In UML, use cases (representing text documents) are implemented through the Use case diagram which shows actors and use cases together with their relationships [OMG 2000]. The diagram itself is a graph of actors, a set of use cases, and the relationships between these elements (associations, generalizations, etc.). It might include also some interfaces. By representing the potential use cases for the system to be built and relevant actors, the diagram provides the starting point in system modeling. Therefore, the proper derivation of use cases and the construction of use case diagram are crucial concerning the task to place consistently the (UML-based) specification of software on prior business process modeling.

**UML - limitations**

Taking into account that UML is the *de facto* standard for designing software systems, we are convinced that any ambitious methodological contribution to the current software design area should be consistent with the UML. We have identified though some limitations of the RUP-UML environment, mainly related to its consideration of the target business reality and also its relatability to simulation models as a way to soundly verify especially some dynamic modeling issues.

As for the first identified limitation, it mainly concerns the fact that UML is about designing software, not about Business Process modeling. Hence, in tune with our claim (stated already) that a sound business/software alignment could be realized only if the target business reality is approached through pure Business Process modeling tool(s), we conclude that UML alone is not capable of realizing a sound and complete business/software alignment. However, this issue has been tackled through the 'use case' and (further on) 'business use case' concepts (Chapter 1). This is a valuable additional contribution to the UML conceptual background. Nevertheless, since it is still a question how use cases are to be identified, based on a consideration of the original business reality and following rigorous methodological guidelines, we are still considering the business/software alignment issue to be not fully solved (yet) by the RUP-UML.

As for the second identified limitation, it concerns the relatability of UML models to simulation. There is consensus in the business/software modeling community that simulation could be a valuable extension to any dynamic system model, mainly because simulation could extend a model with sound graphical and visual notations.
This could be valuable not only for modelers but also for potential users (in understanding the (on-going) modeling activities). A possible relevant improvement could be the combination of UML dynamic models with simulation tools. A step forward in this direction represents the study of Shishkov, Barjis, and Dietz, proposing a combined application of the UML Activity Diagram and the ARENA discrete event simulation tool [Shishkov et al. 2003-2].

3.2.1.2 Other relevant tools - KobrA, Catalysis, Tropos

We will consider below KobrA [Atkinson et al. 2001; Atkinson and Muthig 2002], Catalysis [D’Souza and Wills 1998], and Tropos [Tropos] not only because they are (widely agreed upon) among the well established Software Design methods besides UML+RUP but also because they are useful (though insufficiently, as it will be discussed) in the task of aligning Business Process modeling and software specification. Next to that, these three mentioned tools are popular and successfully applied in practice [Shishkov and Dietz 2004-2]. We will briefly discuss each of them and will analyze their limitations.

KobrA

Our analysis on KobrA has been supported mainly by the following two sources: [Atkinson et al. 2001; Atkinson and Muthig 2002]. Interested readers could find there information about all concepts related to KobrA, which have not been considered in Chapter 2.

The KobrA method is a state-of-the-art approach to component-based product line engineering with the Unified Modeling Language (UML). Among the key characteristics of KobrA are: architecture-centricity; systematic COTS component re-use; integrated quality assurance. The major strengths of KobrA are its overall consistency, the embracement of the component concept in all phases of the software life-cycle, and the UML-based graphical specification of components. The main limitation is that there are no clear guidelines how to relate the specification of software to a prior business analysis and modeling.

A complementary workbench has been developed to support the use of the KobrA method in conjunction with commercial CASE tools. A test bed for the approach has been provided in the domain of Enterprise Resource Planning.

KobrA is conceptually based on the foundation of ‘Product-Line Engineering’. Hence, before proceeding further, we would briefly introduce it. Product Line Engineering is an inherent part of the KobrA method. When pursuing a product line approach in KobrA, the overall software life cycle consists of two basic Product Line Engineering activities:

- **Framework engineering.** This applies the Komponent (‘Komponent’ means ‘Component’ as seen from the perspective of the method KobrA) modeling and implementation activities, accompanied by additional sub-activities for handling variability and decision models, to support a family of similar applications (i.e. development for reuse). A framework therefore contains a generic Komponent tree that captures the common and variable characteristics of a product line.

- **Application engineering.** This uses the framework developed during framework engineering to build particular applications. Since one of the goals of application engineering is to remove the variabilities in the framework, and resolve the decisions in the decision model, Komponent containment trees for applications are very similar to those for a single system. The only difference is that Komponents are
accompanied by a decision model instance, which captures the decisions made in resolving the decision model for a particular Komponent.

Based on the (above outlined) brief information about KobrA, we will (below) come through some basic principles and issues characterizing the method.

A core principle of KobrA is the strict and systematic separation of concerns, so that at all times during a development project developers are aware of what they should be attempting to do and what ‘concern’ they are working on. A manifestation of this principle in KobrA is in the separation of the ‘product’ from the ‘process’ (contrary to methods which arbitrarily mix the description of what engineers should be trying to produce with the definition of how they should produce). Another fundamental separation of concerns in KobrA is the organization of the method in terms of three orthogonal dimensions of development: one dealing with the level of abstraction; one dealing with the level of genericity; and one dealing with composition.

At the largest level of granularity, the Product Line paradigm takes precedence in KobrA. This splits the overall development cycle into two parts, one dealing with the development of a framework – a re-usable set of software artefacts whose core is embedded within all products developed by the enterprise; and the other concerned with the development of an application – a concrete instance of the framework, adapted and extended to meet the needs of a specific customer.

At the intermediate level of granularity, KobrA is driven by the component paradigm. KobrA frameworks and applications are all organized in terms of hierarchies of components. However, the components in KobrA represent the logical building blocks of a software system (not ‘physical’ for example CORBA components).

A central goal of KobrA is to enable the full expressive power of the UML to be used in the modeling of components. To this end, the use of the UML in KobrA is driven by four basic principles:

• Uniformity. Every behavior-rich entity is treated as a Komponent, and every Komponent is treated uniformly, regardless of its granularity or location in the containment tree.

• Encapsulation. The description of what a software unit does is separated from the description of how it does it.

• Locality. All descriptive artifacts represent the properties of a Komponent from a local perspective rather than a global perspective.

• Parsimony. Every descriptive artifact should have ‘just enough’ information, no more and no less.

As for the life-cycle of a KobrA, at the highest level of granularity, this life-cycle is composed of a sequence of phases in which new versions of the central framework are developed and new applications are instantiated from it to meet the expectations of new customers.

In summary, the strict separation of concerns makes KobrA compatible with a large number of practical implementation and middleware technologies. Its embracing the Component Paradigm allows for adequately benefiting from re-use possibilities. Its being soundly founded on the principles of the Product-Line Engineering provides a good theoretical foundation. Its consistency with UML results in a specification of software, which is fully in tune with the current software design standards.

All these (discussed) characteristics of KobrA motivate our claim that it is one of the best examples of a Software Design method being adequate to the current possibilities and demands. We outline though as a limitation of KobrA its treatment of
Chapter 3

the very early software specification tasks and in particular - the relation to the original business system that is to be supported by the software-to-be. As mentioned before, there are no clear guidelines how to relate the specification of software to a prior business analysis and modeling. This could be improved either by extending KobrA backwards (towards a consideration of very early business modeling activities) or by a combination with a business process modeling tool. This conclusion is shared by Atkinson who has discussed with the author the possibility of achieving such improvements. Another identified limitation of KobrA is connected with its not being founded (in its approaching the target business system) on concepts and theories that allow for a proper consideration of the communicative aspects. We have motivated (Chapter 1) the significance of these aspects and therefore claim that the lack of their particular consideration is a disadvantage.

Catalysis

Our analysis on Catalysis has been supported mainly by the following source: [D'Souza and Wills 1998]. Interested readers could find there information about all concepts related to Catalysis, which have not been considered in Chapter 2.

Catalysis is a method for component-based and OO Software Development, which provides a strongly coherent set of techniques for business analysis (characterized by unambiguity about requirements) and system development using UML as well as a coherent method for OO analysis and design. Catalysis provides also well-defined consistency rules across models and powerful mechanisms for composing different views to describe complex systems.

Catalysis is specifically targeted as a method for component-based development, in which families of products are assembled from kits of components. The method also allows for re-use of other artefacts of the design process, such as frameworks of collaboration between objects.

Catalysis includes techniques to map between (UML-based) system design and an analysis model. The gap and inconsistencies are reduced by:

- Unambiguous interface specification;
- Techniques to define powerful component ‘connectors’ abstracting above the level of OO messages;
- “Retrieval” techniques for relating the differing models that different components (especially bought-in or legacy components) usually have (this might include, for example, different notions of what a customer is).

Use-cases have a central role in Catalysis; they are applied at different abstract levels. With each decomposition, the objects interact to fulfil the goals of the more abstract use cases.

The Catalysis method basically comes through the following phases:

- A model of the domain is produced, specifying first, what objects are there and second, the goals which are associated with the major use-cases.
- Scenarios are drawn on how (certain) component could help realizing the major use-cases, breaking them down into individual steps.
- Viewing a component as a specification (this would be possible because at this stage it is to be known what a component is supposed to do). The component has some defined responsibilities, and defined collaborations with the actors around it.
- Component’s responsibilities are distributed between objects inside it and also, interactions between components are defined (use cases are used for that goal). It is possible (if necessary) defining generic interactions between components, so that they are made ‘pluggable’. This is done through template models.

Thus, essential characteristics of the Catalysis method are:
• Usability of generic chunks of software with robust, well-defined interfaces. Dynamic coupling of components is just one form of re-use. Other forms include the import of a generic chunk of design into many other designs. In this sense, "component" can include any piece of development work (code, models, rules, design patterns, and so on).

• Issues which concern the inter-component connections; 'Connectors' play a significant role in this task. They are specified independently on the specification on (relevant) components. Just like objects, connectors are encapsulated: the specification of what one achieves is independent of its implementation.

• Software Development evolving through: first, rapid assembly of end products from components, and second, high quality component development.

In Catalysis, there are particular validation mechanisms. The validation suite is a set of ancillary components for two purposes: first, one set help test the component once it is installed in a particular context, to provide ensure it is running properly; second, a test version of a component exercises the components it is connected to, to make sure they behave as required.

In our view, Catalysis has certain limitations, particularly in the context of the problem of soundly aligning business process modeling and software specification. We claim first that the method does not offer a solid mechanism for the reflection of the original business requirements in the specification of the software's functionality. This is because Catalysis is not fundamentally based on any Business Process modeling theory (even tool). Second, we claim Catalysis to be unfocused, especially as concerns the re-use question: so many issues (code, models, rules, design patterns, and so on) are considered for re-use; however, in neither of the cases there is a thorough methodological explanation on how the particular re-use should be realized. And finally, similarly to KobRA, Catalysis is not founded (regarding its consideration of the (original) business system) on concepts and theories that allow for a proper reflection of the communicative aspects within the business process modeling activities performed.

Tropos

Our analysis on Tropos has been supported mainly by the following two sources: [Tropos] and [Castro et al. 2002]. Interested readers could find there information about all concepts related to Tropos, which were considered in Chapter 2.

Tropos not only soundly approaches the requirements elicitation and specification but also offers mechanisms for transformation of this output into an input for the further Software Design phases. Some of these mechanisms have been supported by the Tropos-related formal method, Telos [Tropos] (Telos is not going to be considered in this chapter). A brief outline of Tropos follows below.

Some issues related to the early requirements analysis with the modeling framework *i* are of essential relevance for Tropos. Hence, introducing this briefly, as well as *i* is considered useful, before (actually) introducing Tropos.

Early requirements analysis (to be also discussed further below) focuses on the intentions of stakeholders. These intentions are modeled as goals. They lead, through goal-oriented analysis, to the functional and non-functional requirements of the system-to-be. In *i* (standing for 'distributed intentionality' which refers to assigning intentional characteristics (such as goals, beliefs, capabilities, commitments) to actors within a distributed organizational environment), stakeholders are represented as (social) actors who depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. The *i* framework includes the strategic dependency model for describing the network of
relationships among actors, as well as the strategic rationale model for describing and supporting the reasoning that each actor goes through concerning its relationships with other actors. These models have been formalized using intentional concepts from Artificial Intelligence, such as goal, belief, ability, and commitment.

These issues are to be considered as the foundations of Tropos which undergoes five phases.

**Acquisition of Early Requirements** is the first Tropos phase. They are concerned with the understanding of a problem by studying an organizational setting. The outputs of this phase are two models: First, Strategic Dependency (SD) Model capturing relevant actors, theirs respective goals and their interdependencies; Second, Strategic Rationale (SR) Model determining through a means-end analysis how the goals can be fulfilled through the contributions of other actors.

**Definition of Late Requirements in i** is the second Tropos phase. They are concerned with the description of the system-to-be, within its operational environment, along with relevant functions and qualities. The outputs of this phase are revised SD and SR models: First, inclusion in the original Strategic Dependency (SD) Model of an actor to representing the software system-to-be; Second, consideration of this system actor by doing a means-ends analysis to produce a new Strategic Rational (SR) Model; Third, decomposition (if necessary) of the system actor into several sub-actors and revision of the SD and SR Models.

**Architectural design** is the third Tropos phase. It concerns the definition of the system's global architecture in terms of sub-systems, interconnected through data, control and other dependencies. The outputs of this phase are a Non-Functional Requirements (NFR) Diagram and revised SD and SR models, and also, agents are introduced: First, selection of an architectural style using as criteria the desired qualities identified in Phase 2 and producing a NFR diagram representing the selection and design rationale; Second, introducing (in case it is required) new system actors and dependencies, as well as the decomposition of existing actors and dependencies into sub-actors and sub-dependencies, and also revision of the SD and SR Models; Third, assigning actors to agents and roles/patterns to solve actors' goals.

**Detailed design** is the fourth Tropos phase. It is about the definition (in further detail) of the behavior of each architectural component. The outputs of this phase are Agent Class Diagrams, Sequence Diagrams, Collaboration Diagrams and Plan Diagrams: First, construction of a Class Diagram, based on the SD and SR Models; Second, building of Sequence and Collaboration Diagrams capturing inter-actor dynamics; Third, construction of Plan (state-based) Diagrams capturing both intra-actor and inter-actor dynamics.

**Implementation** is the fifth (and last) Tropos phase. Its output is a Beliefs-Desires-Intentions (BDI) agent architecture. From the detailed design, Agents, Capabilities, Database Relations, Events and Plans in JACK are generated.

Our views on the limitations of Tropos (particularly in the context of the current research) are in four directions, namely:

- Limited use of the OO way of thinking and the possibilities for re-use;
- Consideration of actors rather than actor roles;
- Insufficient concern with the communicative aspects in considering a Business System;
- Absence of clear guidelines on how to map towards UML diagrams.

We will briefly comment on the above issues.

Chapter 1 posed the definite claim that applying the CBD/OO way of thinking could bring significant value in the task of aligning Business Process modeling and

56
software specification. In support of this claim is the tendency towards relying on the CBD paradigm in proposing new system specification methods. For this reason, we consider it important judging to what extent Tropos embraces the CBD way of thinking. In this regard one should take into account the 'architectural components' in the Tropos framework. This is an example of the consideration of the CBD way of thinking. Next to that, the consistency with UML is also in support of this. However, no sound theoretical foundation exists in this direction. There are no strictly defined component derivation rules both in the business process modeling task and the software specification task. Something more, the business/software alignment itself is not at all based on components. The Tropos modeling framework does not include the identification of Business CoMponents and their reflection in Software CoMponents. Logically, as a result of this, Tropos does not offer methodological re-use guidelines. Therefore, the CBD way of thinking is limitedly used in Tropos. This is an obstacle for its utilizing relevant modeling benefits, including re-use.

The second discovered limitation of Tropos concerns its considering actors. Our view is that this is a disadvantage because the complexity of the currently produced software systems makes it inadequate approaching them through models consisting of actors/agents. The reason is that quite often an actor could be involved in a number of roles. Modeling this (if using actors) would be much more complicated than modeling roles.

The third discovered limitation of Tropos relates to its not being founded (regarding its Business Process modeling part) on concepts and theories that allow for a proper consideration of the communicative aspects within the Business Process modeling; this applies also to KobrA, as stated.

And finally, the fourth discovered limitation of Tropos is connected with the lack of clear guidelines on how to reflect the requirements models into UML diagrams. There are many examples considered and the explanations there are rather convincing. However, we still miss sound guidelines on how to identify such models.

In summary, although offering significant value in aligning Business Process modeling and software specification, Tropos is not rigorous in the mapping itself (the mapping towards (UML) models), is limitedly benefiting from the CBD way of thinking, and is inadequately considering essential (in our view) issues such as actor roles and the Business System communicative aspects.

### 3.2.2 Software Specification Tools: Conclusions

UML is the Software Design standard. Hence, Business Process modeling capabilities of RUP-UML or a possibility to use (within UML in a straightforward way) the output of a Business Process modeling tool, would be essential for aligning Business Process modeling and software specification while staying it tune with the current Software Design standards. However, currently RUP-UML does not offer such possibilities since the only existing ideas in this direction (concerning the 'Business use case' concept) still need development: we miss rigorous rules on how to identify such use cases.

The other (consistent with UML) software specification tools, considered, have limitations either in the pre-specification phases or in the specification one. This is an obstacle for soundly aligning Business Process modeling and software specification. We observe that this applies to most current software specification tools. Actually, we are not aware of a popular UML-driven software design tool capable of realizing a sound Business Process modeling prior to the software specification phase.

Thus, in the context of the necessity to adequately align Business Process modeling and software specification, we conclude that the current software specification instrumentarium is insufficiently powerful with respect to this challenge.
3.3 SUMMARY

In the previous sections, we considered modeling tools which relate to either Business Process modeling or software specification. On Figure 3.10, we illustrate their positioning.

![Diagram showing positioning of investigated tools]

Figure 3.10: Positioning the investigated tools

As it is seen from the figure, some of them are semi-formal Business Process modeling tools (DEMO, Norm Analysis, Semantic Analysis, ORM). Others are semi-formal software specification tools (Catalysis, KobrA, UML-RUP). Business-Statistics-related tools (Regression Analysis, for instance [Freund 2003]) are formal Business Process modeling tools. The same is valid for Functional Analysis (and related tools), for example. And finally, Telos (mentioned before) is an example of a formal software specification tool (Telos uses formal mechanisms for specifying software systems based on business requirements). As for Tropos, it is limited to business requirements analysis, addressing from this perspective the software specification task. Unlike Telos, Tropos is semi-formal. Tropos is positioned to the left (in comparison to Catalysis, KobrA, and UML-RUP) because although being a software specification tool, Tropos is capable of analyzing early requirements – this brings it somehow closer to the Business Process modeling area. Petri Net could be used both for Business Process modeling and software specification. Simulation tools could be formal and semi-formal, being used in support of Business Process modeling. Hence, neither of the considered (formal/semi-formal) tools, except for Petri Net, could be used for both Business Process modeling and software specification purposes. However, as seen from our analysis, Petri Net is limited to just one Business System modeling perspective, namely the dynamic perspective.

Thus, our claim is that the software community still misses a well-established modeling tool adequately comprising Business Process modeling and software specification. The conducted analysis has supported this claim, by bringing evidence that the current practices do not include yet an integrated approach towards Business Process modeling and software specification. It has been shown, analyzing different tools, that these two aspects are considered separately.
In our view, this could hardly be overcome by simply extending the scope of any of the considered tools. The reason is that each tool is rooted in theories and concepts which are oriented either towards Business Process modeling or towards software specification. This conceptual limitation could be addressed only if a theoretically consistent background is built on how to align Business Process modeling and software specification. However, such a background does not currently exist [Shishkov and Dietz 2004-1].

We make also the conclusion that the CBD paradigm has not yet ‘penetrated’ the Business Process modeling domain. This is another obstacle in bridging the Business Process modeling and the Software Design ‘worlds’, given the fact that most of the current Software Design methods are embracing the mentioned paradigm.

All this justifies our purpose to propose an approach that allows for adequate component-based business-software alignment. The conceptual framework of the proposed approach will be introduced in the following chapter.
Chapter 4

THE SDBC APPROACH

*We arrive at the truth, not by the reason only, but also by the heart.*
*Blaise Pascal*

In Chapter 1 (the Introduction), we have presented the current research’s background. It includes:
- actual research problem identified;
- two promising research perspectives selected, namely the Component-Based (system) Development (CBD) and the Language-Action Perspective (LAP), which are claimed to be useful in approaching the problem;
- research hypotheses posed on this basis;
- research goals and questions, subsequently formulated.

Then, considering the need for a conceptual elaboration, Chapter 2 has introduced and discussed the concepts/definitions claimed to have fundamental importance for this research. And further on, before introducing its main contributions, a study has been carried out (in Chapter 3), aiming at analyzing existing relevant tools and related knowledge.

Based on all this, the current chapter introduces the foundations of the SDBC (SDBC stands for Software Derived from Business Components) approach [Shishkov and Dietz 2005-1; Shishkov and Dietz 2004-2; Shishkov and Dietz 2004-1]. It allows for software specification based on identified (re-usable) Business CoMponents.

On the basis of the mentioned proposition:
- the following chapter will suggest a selection of modeling tools suitable for making SDBC operational, and
- Chapter 6 will introduce step-by-step methodological guidelines concerning the application of the SDBC approach.

The first section (4.1) of the current chapter will introduce SDBC by presenting the foundational issues behind it. Section 4.2 will reveal on this basis the approach’s framework. Section 4.3 contains concluding remarks.
4.1 ESSENTIAL ISSUES BEHIND SDBC

The previous chapters have lead to three definite conclusions summarized as follows:
1. An adequate analysis/design of a current Business System should necessarily concern not only the production acts (and their flows) but also the corresponding communicative aspects.
2. Further studies are required in the direction of aligning Business Process modeling and software specification.
3. In dealing with the business-software-alignment problem, it is worthwhile considering the CBD paradigm, benefiting in this way from the undisputable advantages of Object-Orientation, among which: re-usability, modifiability, design flexibility [Shishkov and Dietz 2002-2].

The mentioned conclusions are in tune with the research goal (Chapter 1) whose achievement we approach through the SDBC approach. Hence, taking this into account, we formulate the essential desired property of SDBC:

\textit{Considering not only the structural, dynamic, and data perspectives, but also the communication perspective on a Business System, SDBC should allow for its adequate component-based alignment to further software specification.}

Starting from these basic characteristics of what SDBC should be, it is necessary to position the approach within the Software Engineering life cycle [Sommerville et al. 1987] since SDBC aims at bringing improvements related to the creation of software artefacts.

A reasonable starting point in this regard is a consideration of the IEEE Glossary of Software Engineering Terms [IEEE-610]. According to it, \textit{Software Engineering is a systematic approach to the development, operation, maintenance, and retirement of software.} We consider also the \textit{process} definition of Loucopoulos and Karakostas (1995), adapted from a corresponding ISO definition [ISO]: \textit{A process is a unique, finite course of events defined by its purpose or by its effect, achieved under given conditions.} Hence, according to our view:

\textit{Software Engineering is a systematic approach which generally undergoes the phases of development, operation and maintenance, and retirement. Each of these four phases can be viewed as a process.}

The Development Phase is the only phase (from the four ones mentioned above) which directly concerns the creation of a software artefact [Jacobson 1995]. For this reason, we are particularly interested in that phase.

We could define \textit{Software Development as the process which leads to the creation of a software artefact.}

\textit{Specification} and \textit{realization} are two of the essential Software Development steps [Atkinson and Muthig 2002]. Among the other essential steps are coding, testing, and so on. Such a consideration is particularly consistent with the component-based Software Development [Jacobson et al. 1992].

\textit{Specification} addresses the functionalities of the derived software components and their interactions. Refining this, the software \textit{realization} considers deeper the issues related to the realization of the specified functionalities. Therefore, going as deep as the specification is claimed to be enough for SDBC to properly base the software system-to-be on an adequate Business Process modeling background.

All this allows us to further refine our formulation of the desired property of SDBC:

61
**Chapter 4**

**SDBC should be an approach concerning the series of interrelated steps, part of the Software Engineering task, which allow for building a software specification model on the basis of a Business Process one, under the conditions that 1) the Business Process model is soundly elaborated in terms of structural, dynamic, factual, and also communicative issues; 2) the models are aligned in a component-based way.**

The further Software Development steps (including realization, implementation (coding), and testing) are left beyond the scope of SDBC since (as mentioned above) they are not considered to be directly related to the challenge of aligning Business Process modeling and the creation of software; as stated already, this challenge is a fundamental issue for the current research.

Therefore, we could now position the SDBC approach within the Software Engineering task. This is illustrated in Figure 4.1. As seen from the figure, SDBC is concerned with the specification of a software artefact. As stated already, the approach faces the significant challenge of the derivation of a software specification model on the basis of Business Process modeling. In addition to this, there is a demand that the achieved business-software alignment is component-based.

![Figure 4.1: The positioning of SDBC within the Software Engineering task](image)

Thus, being an approach for component-based alignment between business and software specification models, SDBC should allow high-level Software CoMponents (Chapter 2) to be derived based on Business Process modeling. They might be described, for instance, in the UML, and further implemented into Software Components, using conventional Software Development approaches as well as the latest component technologies, for example: JavaBeans [EJB], CORBA [CCM], and .NET [DNET]. The existing knowledge (both theoretical and practical) relevant to these issues, is considered to be sufficient. As mentioned before, the realization, implementation, and testing are left beyond the scope of SDBC.

Taking into consideration the formulated desired property of SDBC as well as the identified promising (relevant) research perspectives and directions, we suggest that SDBC is based on *four essential fundamentals* (as shown on Figure 4.2):

![Figure 4.2: The essential fundamentals of SDBC](image)
1) **Integrated view over Business Process modeling and software specification** — such an integrated view is necessarily consistent with the task of aligning Business Process modeling and software specification.

2) **Business Process modeling embracing the Communication perspective** — in the task of Business Process modeling (to be aligned to software specification), it is considered essential embracing the Communication perspective (not only the structural, dynamic, and data perspectives), in tune with one of the posed research hypotheses (Figure 1.2) and adopted promising research perspectives.

3) **Component-based business-software alignment** — the CBD paradigm is considered being applicable and useful if the goal is to align Business Process modeling and software specification. This is as well in tune with one of the research hypotheses (Figure 1.1) and adopted research perspectives.

4) **Re-use** — it is widely accepted that re-use could make the process of building business/software models more effective [Castano and Antonellis 1993; Castano et al. 1992; De Antonellis 1991; Coad 1992; Johnson and Foote 1998], thus we have defined the goal of allowing for re-use within SDBC.

These four fundamentals are elicited further on in the current section.

### 4.1.1 Integrated View over Business Process Modeling and Software Specification

Current Software Design practices are characterized by a lack of an adequate business-software alignment [Shishkov and Dietz 2005-1]. Small software companies usually rely on an arsenal of best practices / ‘know-how’ [AWARENESS], trying to adapt it to the user’s case. Bigger companies, however, spend more time and energy for getting insight about the target Business System. Anyway, the process of getting such an insight plays just a supportive role for the system specification, without being integrated with it. Therefore, we would introduce a valuable improvement, by integrating (through SDBC) Business Process modeling activities and software specification ones (as shown on Figure 4.3). This distinguishes SDBC from most of the current Software Design approaches (considering business issues from Software Design point of view).

![Diagram](image)

**Figure 4.3: Integrated view over Business Process modeling and software specification**

As seen from the figure, SDBC approaches the software specification task in two stages: firstly, a Business Process model is built considering the corresponding business reality; secondly, a software specification model is built based on the Business Process model. By applying still further decomposition, we derive the detailed process (inspired by studies of Shishkov and Dietz (2004-1), Shishkov and Van Sinderen [AWARENESS] as well as by the ‘classical’ MDA trajectory [MDA 2001]) through which SDBC should progress in our view - Figure 4.4-a.
As shown on the figure, following the SDBC design trajectory, one should firstly consider the initial descriptive information (provided by the future user(s) of the software system-to-be) which is a usual input in any software project, as it is well-known. Then a description of the approached business reality is derived. However, it might be necessary to conduct re-design (imagine that the original business reality consists of a local service provider and users; introducing mobility, we could rely on a number of service providers based in different locations; thus, before specifying software, we would need to describe the 'future' (desired) business reality accordingly). Then, we should delimit a relevant part of the business reality depending on our particular software goal (on what we are going to automate, according to the requirements of the users). Figure 4.4-b summarizes these issues.

Hence, having the description of the delimited part of the original (or eventually re-designed) business reality, we could proceed towards the Business Process modeling task (Figure 4.4-a). As seen from the figure, another related input is to be the domain-imposed requirements characterizing the original Business System:
norms which are to be fulfilled. Further on in the current chapter, we will pay special attention to questions concerning requirements.

Therefore, we build a Business Process model which is to be mapped towards a software specification model. However, as it is depicted on the figure, besides the Business Process modeling input, the SDBC design trajectory envisions two other necessary inputs:
- the user-defined requirements – the requirements which the future user(s) of the software system-to-be have stated concerning its functionality;
- design constraints – the design limitations which should be followed as a result of software/hardware/network (and other) project restrictions.

Thus, 5 basic tasks could be identified, namely description (plus eventually redesign), delimitation, Business Process modeling, software specification as well as requirements elicitation.

The figure shows as well that the requirements elicitation task (although concerned fundamentally with the software specification) spans over both the Business Process modeling activities and software specification ones. The requirements elicitation [Wieringa 1995] will also be discussed (in the context of SDBC) in the current sub-section, as mentioned above.

Concerning the items depicted on Figure 4.4-a: from left to right and from top to bottom they become smaller (in area) and more regular (in shape). This is to indicate that each following state relates to a smaller part of the original business reality (in the delimitation, we exclude issues from the original model, in the Business Process modeling, we further exclude issues from the delimited model, and so on) and is becoming more and more structured.

We will now further elaborate on 4 of the above-mentioned tasks, since they require further elicitation; these are: delimitation, Business Process modeling, software specification, and requirements elicitation.

**Delimitation**

As seen from Figure 4.4-a, before the software specification and even before the Business Process modeling activities take place, it is necessary to conduct sound Business Process study that thoroughly reflects the considered business reality, achieving in this way precise delimitation. We find this necessary because, as it is well-known, an adequate modeling should be conducted based on a proper description and understanding of the addressed reality as well as on a precise focus on the part of the reality to be considered in the modeling process [Shishkov and Dietz 2004-1]. In SDBC, we respond to this through ‘description+filtration’:

- Thorough *description* of the Business System being approached (the business reality under consideration, which might be (eventually) redesigned) needs to be done: the suggested starting point for realizing such a description is the consideration of the original documentation of the studied system; however, it should be taken into account that such information is usually insufficient and often full of errors. Thus, it should be additionally analyzed/refined. The decision how detailed the description should be depends on the selected level of granularity, this level should be adequate to the characteristics of the software system-to-be.

- *Filtration* of only those issues from the description, which are relevant to the software system-to-be is to then take place. They are to be, however, soundly rooted in the broader context of the approached business reality. This link would contribute to building software that is well integrated in the target (business) organization.
Illustrating description+filtration: describing a restaurant means considering (at least) all static and dynamic issues there regardless of their potential relevance to the functionality of a software application(s) to be built. An application might, for instance, handle the (electronically realized) restaurant reservations. Hence, filtering would mean in this case to keep only those description elements which are relevant to the reservation functionality. They are to include, for instance, the issues related to the roles: Reservation manager, Cook, Supplier (the Cook must be aware of the reservations in order to plan the cooking activities in an effective way; the Supplier should also be aware of the reservations in order to guarantee sufficient supplies), and so on.

However, description and filtration are not to be always realized as two separate tasks, it is possible that they overlap. Returning back to the example: it might be obvious from the beginning that describing the Porter (of the restaurant) is of no use since the 'functionality' of the Porter is irrelevant to the restaurant (electronic) reservations; regardless of other circumstances, the Porter must stay by the restaurant's entrance during the opening hours.

It might be concluded that filtration concerns the alignment between Business Process modeling and software specification since it focuses the business study on particular part(s) of the studied business reality, which are to be automated through (software) technology [Shishkov and Dietz 2004-1].

Thus, by applying description+filtration, the modeler can perceive just those elements of the approached business reality, which are relevant to the software-system-to-be. However, such a view, although focused, is insufficient as a basis for deriving a software specification model. Based on the filtered description, a multi-aspect Business Process model (Chapter 1) should be built as a source for the software derivation. In support of this claim, we provide the following arguments:

- A software specification based on a (multi-aspect) Business Process model would be sounder than a specification just based on a description which might be ambiguous.
- Since, as it will be explained further on, the requirements elicitation concerns both the Business Process modeling phase and the software specification one, a Business Process model would bring value because of its structured background.
- The business study input to the software specification would need to be validated. It is easier to validate a model than an unstructured description.
- It is easier to discuss and communicate a model than a description because a model has some structure, some (semi-) formal background.

Hence, a multi-aspect Business Process model relevant to the software system-to-be should be built.

**Business Process modeling**

As it could be seen from Figure 4.4-a, the task following the delimitation is to be the Business Process modeling. It should be complete regarding the four perspectives (as claimed in Chapter 1), namely: structural, dynamic, data, and communication perspectives [Shishkov and Dietz 2005-1], as illustrated on Figure 4.5. This is just an indication that the multi-aspect Business Process modeling is to be integrated with the (further) software specification steps. As for the communication perspective (concerning one of the research hypotheses (Chapter 1)) it will be especially considered further on – being itself one of the essential SDBC fundamentals (Figure 4.2). That is why the upper box on Figure 4.5 is colored in grey (to indicate that).
As for the perspectives: the structural perspective is about the entities and their interrelations; the dynamic perspective is about the flow(s) of events; the data perspective is about the factual issues; the communication perspective is about the communicative acts exchanged during the business operation.

As mentioned already, an issue of crucial importance is how such a (multi-aspect) Business Process model could be adequately mapped towards software specification. Arguments have been presented in favour of the claim that a component-based business/software alignment could be a possible solution. However, this issue is to be considered further on in this chapter as well as in Chapter 6.

**Software specification**

As already stated, SDBC should align (in a component-based way) a (multi-aspect) Business Process model to software specification. Said otherwise, the software model should be methodologically derived from the built Business Process model. The (mentioned) component-based alignment (to be discussed in sub-section 4.1.3) is claimed to be a solution, as mentioned above, making it possible to soundly base a Software Design model on a Business Process model. Another issue is the need for consistency with the current Software Design standards, such as UML (Chapter 3) and XML [XML]. The reason is that any software-related output would be of little use today unless it is relatable and combinable with other relevant existing tools; a compatibility of this type could be realized only on the basis of software standards [Shishkov and Dietz 2004-2]. The software specification's being rooted in these two essential fundamentals is illustrated in Figure 4.6.

Actually, the reflection of a Business Process model in the software specification would be the viable link that should guarantee the proper alignment between the business and software aspects.
Specified in this way, a software model should further undergo realization and integration, in building the software system. However, as mentioned before, this is outside the scope of SDBC.

**Requirements elicitation**

Requirements relate directly to the specification of software [Wieringa 1995]. They are descriptions of how the system-to-be should behave, application domain information, constraints on the system’s operation, or specifications of a system property or attribute [Kotonya and Sommerville 1998]. Thus, a proper consideration of the original business requirements in the specification of a software’s functionality is of significant importance in the process of aligning Business Process modeling and software specification. Our consideration of the Requirements issue is illustrated in Figure 4.7 and is in consistency with the SDBC design trajectory (Figure 4.4-a).

As seen from Figure 4.7, the built Business Process model should concern the discovery of a part of the system requirements, namely those requirements that characterize particularly the Business System under consideration. They are often called ‘**Domain-imposed requirements**’ [NATURE 1995]. The two arrows between the two lower boxes on the figure indicate that not only the (domain-imposed) requirements could affect the initial Business Process model, by causing some updates in it but also that the Business Process model affects the requirements elicitation, by stimulating the discovery (or specification) of additional requirements.

![Figure 4.7: Consideration of the Requirements issue](image)

As also seen from Figure 4.7, besides the domain-imposed requirements one should identify also the so-called ‘**User-defined requirements**’. The latter are determined by the users of the system-to-be and are not directly related to the Business Process model. For this reason, there is a dotted line between the upper two boxes on the figure.

In summary: during the Business Process modeling, the domain-imposed requirements are to be discovered and considered in the mapping towards software specification; next to that, the user-defined requirements are to complement the Business Process model in providing the input for the derivation of the software specification model [Shishkov and Dietz 2005-1]; hence, both types of requirements
should be precisely specified in the process of realizing a business/software alignment.

Another important issue concerning requirements is the verification aspect - showing that the specified software system is correct with respect to its stated requirements. Three major problems arise in connection with this:

a) How to formalize the stated requirements in such a way allowing for their adequate consideration?

b) How to mimic the developed software specification model in such a way that its functionality is demonstrated?

c) How to match up the aspects mentioned in a) and b)?

Some of these issues will be considered in Chapter 6.

4.1.2 Business Process Modeling Embracing the Communicative Aspects

As studied in Chapter 1 and Chapter 2, the essential goal (already motivated) of grasping within SDBC the communicative aspects of a considered Business System could be accomplished by founding the Business System study on LAP - the Language-Action Perspective [Winograd 1988; Habermas 1984; Winograd and Flores 1986; Flores and Ludlow 1986], and in particular on the DEMO interpretation of some essential aspects of this theoretical orientation [Dietz 2003-1]. Valuable in this regard is the relation of DEMO (Chapter 3) also to two other sound and relevant to the mentioned goal theories, namely Organizational Semiotics [Liu 2000] and Philosophical Ontology [Bunge 1979]. Additional inspiration in this direction has brought the study of Dietz, proposing an atom-molecule-fiber layering within the investigation of (business) systems [Dietz 2003-2]. Hence, the LAP Transaction concept, as introduced and considered in Chapter 2, is adapted and adopted within SDBC as an elementary Business Process modeling unit (elementary business engineering building block). This adoption is actually realized through the Business CoMponent concept (also introduced in the mentioned chapter) which is based on the Transaction concept. In this way, SDBC allows for basing a specification of software on a Business Process model that is soundly elaborated in terms of communicative action issues (not only in terms of structural, dynamic, and factual issues what is the case with the current popular Business Process modeling methods). The relation of SDBC (realized through the Business CoMponent concept) to the mentioned communication perspective and in particular, to relevant theories considering it, is illustrated in Figure 4.8.

LAP: Language-Action Perspective
OS: Organizational Semiotics
PO: Philosophical Ontology

Figure 4.8: The importance of the Transaction concept for SDBC
SDBC interprets the Transaction concept as centered around a particular production fact (following the DEMO Production fact definition [Dietz 2003-2]). The reason is that the actual output of any Business System represents a set of production facts related to each other. They actually bring about the useful value of the business operations to the outside world and the issues connected with their creation are to be properly modeled in terms of structure/dynamics/data, as in the popular current Business Process modeling methods.

![Diagram of the SDBC interpretation of the Transaction concept]

**Figure 4.9: The SDBC interpretation of the Transaction concept**

However, the already justified necessity of considering also the corresponding communicative aspects is important. Although they are indirectly related to the production facts, they are to be positioned around them. As already stated, SDBC realizes this through its interpretation of the Transaction concept, as depicted in Figure 4.9; as seen from the figure, the classical LAP-DEMO Transaction concept (introduced and explained in Chapters 2 and 3) has been adopted, with a particular stress on the Transaction’s output – the production fact. The Order phase [Shishkov 2003] is looked upon as an input for the production act, while the Result phase is considered to be the production act’s output. The dashed line shows that a Transaction could be successful (which means that a production fact has been (successfully) created) only if the Initiator (according to DEMO [DEMO]) has accepted the (Production) act of the other party (called Executor). As for the (coordination) communicative acts, grasped by the SDBC Transaction, they are also depicted on the figure. The Initiator expresses a ‘request’ attitude towards a proposition (any Transaction should concern a proposition – for example, a shoe to be repaired by a particular date and at a particular price, and so on). Such a request might trigger either promise or decline - the Executor might either promise to produce the requested product (or service) or express a ‘decline’ attitude towards the proposition. This expressed attitude actually triggers a discussion (negotiation), for example: ‘I cannot repair the shoe today, is tomorrow fine?... and so on’. The discussion might lead to a compromise (this means that the Executor is going to express a ‘promise’ attitude towards an updated version of the proposition) or might lead to the Transaction’s cancellation (this means that no Production fact will be
created). If the Executor has expressed a 'promised' attitude regarding a proposition, (s)he must bring about the realization of the Production act. Then the Result phase follows, which starts with a 'statement' expression from the Executor about the requested proposition that in his/her opinion has been successfully realized. The Initiator could either accept this (expressing an 'accept' attitude) or reject it (expressing a 'decline' attitude). Expressing a 'decline' attitude leads to a discussion which might lead to a compromise (this means that finally the Initiator is going to express an 'accept' towards the realized Production act, resulting from negotiations that have taken place and compromise reached) or might lead to the Transaction's cancellation (this means that no Production fact will be created). Once the realized (Production) act is accepted the corresponding Production fact is considered to have appeared in the (business) reality.

The Transaction concept, interpreted in this way, is the fundamental (within the SDBC approach) Business Process building block (our Business Process definition has been introduced in Chapter 2). The Business CoMponent concept, also introduced in Chapter 2, defines a Business coMponent as a model corresponding to a Business Process. Therefore, via the Business CoMponent concept, SDBC relates the specification of software to a LAP-founded consideration of the communicative aspects of a target Business System.

Based on the LAP-DEMO studies so far and their SDBC interpretation, it is claimed [Shishkov and Dietz 2004-1] that by applying the Transaction concept, SDBC achieves:

- a realistic position of the issues belonging to a Business System;
- a granularity level (considered) which is the right one as long as the atomic business (process) issues are of interest;
- a sound theoretical justification in the (theories of) Language-Action Perspective, Organizational Semiotics, and Philosophical Ontology.

The Transaction concept directly relates to the Business CoMponent one which is fundamental for the SDBC approach. The Business CoMponent concept (Chapter 2) is of essential importance for the following sub-section.

### 4.1.3 Component-Based Business-Software Alignment

The perspective of realizing the alignment between the two significant SDBC tasks (Business Process modeling and software specification) on the basis of components is a crucial issue in the SDBC approach. The proposed component-based alignment has been justified in the previous chapters by the undisputable and well proven (in practice) advantages of the CBD/OO paradigms – it allows for representing a (software) system in terms of objects/components. As mentioned in the Introduction, it is claimed and justified (by researchers, like Jacobson, for example) that the CBD way of thinking could be (successfully) applied in modeling Business Systems. Thus, identifying Business CoMponents and reflecting them in (sets of) Software CoMponents would be well founded theoretically and there would be good design flexibility and traceability [Shishkov 2002-1]. Next to that, coMponents/components could be re-used. As it is well-known, re-use is an essential advantage for any system design method. Re-use will be discussed in the following sub-section, as a foundational issue within the SDBC approach. The component-based alignment between Business Process modeling and software specification is illustrated in Figure 4.10.

71
As depicted on the figure, the target business reality is to be reflected in a set of identified Business CoMponents (these are Business Process models according to Chapter 2). Based on them, a (component-based) software model is to be specified. The Business and Software CoMponents are not to be necessarily mapped one to one. The bottom line in building up the Business Process model (through the identification of Business CoMponents) should be a purely business-oriented study that has nothing to do with the specification of software and related issues. On the other hand, the software specification (and integration), though based on the Business CoMponents, is to be realized from the perspective of the functionality of the software system-to-be. Therefore, it is possible that more than one Software CoMponents are derived based on a Business CoMponent (these issues will be considered in detail in the following sub-section).

Hence, embracing the principles of Component-Based (system) Development brings all the advantages associated with this type of system development to both the Business Process modeling phase and the software specification one. The component-based perspective makes it doable to easily trace a relation between a designed (or even implemented) Software CoMponent and its originating Business Process model. Adding corrections and/or modifications further on would be easy as well. Even entire models/coMponents could be inherited and built in a new context, for example. Therefore, the mapping itself (between Business Process modeling and software specification) could be significantly facilitated. Next to that, re-use would be possible and easily realizable at different levels. A (specified) Software CoMponent could be used for the design of different systems; an implemented Software Component could be used for the development of of different software artefacts as this is successfully done currently. Also the Business CoMponents could be re-used – if some business requirements change, it would be possible to replace a Business CoMponent with another one without affecting the entire model of the Business System. As mentioned already, the re-use aspect which is of great importance for SDBC, will be focused in the following sub-section.

### 4.1.4 Re-use Issues

As mentioned before, re-use options are essential for SDBC. The approach benefits from them in both the Business Process modeling phase and the software specification one; SDBC allows for re-use of Business Processes – if they are abstractly described and also for re-use of coMponents (Business and Software CoMponents). This is depicted in Figure 4.11.
Hence, as seen from the figure, three re-use levels are to be distinguished: at the lowest level: re-use of Software Componenents; the re-use of Business Componenents; and at the highest level: the re-use of Business Processes. These issues are discussed below.

**Re-using Software Componenents** is an option within the SDBC approach. It is taken into account that the re-use of implemented parts (called 'Software Components' according to Chapter 2) of ICT applications is one of the main motivations for component-based software development since, as stated in Chapter 1, re-using a prefabricated piece of (implemented) software in another application(s) saves the associated development costs and also the effort involved for ensuring the component's quality and integrity. However, the SDBC approach only partially supports that since, as it has already been explained, the approach goes as far as the specification of software. It has been explained as well (Chapter 2) that our 'Software Component' definition stays within the 'specification area'; it might be useful reminding that by 'Software Component' we mean a specification model of a part of the software system-to-be. Through such a support for the re-use at the specification level, it is claimed that SDBC supports the component-based Software Development since the approach provides a methodological mechanism for deriving Software Componenents which are further reflectable into Software Components which would handle the actual functionality of the software system-to-be.

Speaking about the derivation of Software Componenents, and considering the background of SBCS, according to which the (specified) software model should be methodologically based on a Business Process model, it is clear that within the approach, the Software Componenents are to be derived on the basis of Business Componenents. Thus, we should comment on the 'Business Component' concept, the 'Software Component' concept, and the relation between them.

The mentioned concepts have already been thoroughly introduced and discussed in Chapter 2. For this reason, we will not discuss them in much detail. We made it clear that by 'coMponent' we mean a *model of a part of a system*. Hence, we view a 'Business Component' as a *model of a part of a Business System* (associated with a particular Business Process); and we view a 'Software Component' as a *(software specification)* model of a part of the software system-to-be. As for the business/software alignment, it should lead to the *reflection of a Business Component in a resulting software specification model*. Actually, in our view, a particular software system-to-be corresponds to a Business Component (refer again to Chapter 2). Hence, SDBC would allow for reflecting a Business Component in a software specification model that corresponds to the software system-to-be. Based on the model, SDBC should allow for the identification of Software Componenents. Such componenents could be identified from the model via decomposition. This is illustrated on Figure 4.12.
As seen from the figure, a Business Component is to be methodologically reflected in the specification of software. As also seen, such a 'Business Process input' alone is insufficient for specifying a piece of software. One is to consider as well what do the (future) users of the system-to-be require (said otherwise, considering the user-defined requirements) and some technical (and technological) issues, leading to design restrictions (since software systems are about the technological solutions of some problems in Business Systems) – this has already been mentioned and reflected in Figure 4.4-a. Based on all this input, a Business Component could find its reflection in a software specification model of the system-to-be. The model could be presented, for instance, in the use case notations. However, for the purpose of re-use, we might find it useful to identify (by decomposing the model) some Software Components. Hence, we arrive at the identification of a Software Component(s). As shown on the figure, there is also another possibility, especially when we do not have the usual situations of a number of Software Components corresponding to one Business Component: the situation might be (because of the granularity of a Business Component, for example) that a Business Component is reflected in a software specification model which is not wise to undergo decomposition (because it is re-usable as it is, for example). In such cases we directly arrive at the identification of a Software Component, on the basis of the Business Component. Figure 4.12 (its right part) illustrates particularly how in the first situation (situation ‘a’) we reflect a Business Component in a number of Software Components, and in the second situation (situation ‘b’) we reflect a Business Component in just one Software Component.

Identified in this way, the Software Components should be described (using the UML Use case model, for example), further specified (specifying in detail some
essential use cases from the model, for example), elaborated (reflected in other UML aspect models, like the UML State diagram, for example), integrated, realized, and implemented.

As already stated, SDBC concerns the identification, description, specification, elaboration, and integration. However, the approach does not consider realization and implementation issues.

As far as these issues are concerned, Software Componennts specified within SDBC, could be implemented using any of the currently popular and proven distributed computing environments like CORBA [CCM], EJB [EJB], and .NET [DNET], which have already been mentioned. This is claimed to be in tune with the latest software re-use practices.

As far as the re-usability of Software Components is concerned, the existing knowledge [Szyperski 1998, 1999; Atkinson and Muthig 2002] is considered to be sufficient; it is reflected in developing SDBC. Actually, the component-based software development differs from traditional approaches by splitting the development process into two distinct activities [Atkinson et al. 2001]:

- Development for reuse – creating high-quality, specialized components which concentrate on doing a specific job well; they are to be of use in multiple cases.
- Development with reuse (called also 'integration') – creating new artefacts (or possible larger components) by assembling prefabricated components.

All this (existing) knowledge (summarized above) is claimed to be applicable for both components and components, and is thus adopted within the SDBC approach.

Re-using Business coMponents is of significant importance for the approach being proposed. Below we will present the relevant innovative ideas within SDBC.

If we consider the building up of a system (in general) out of a number of building blocks and want to re-use some of them, we have two solution directions [Shishkov and Dietz 2004-2] – we could have either a core unit that we should build further on, in order to specify a building block, or we could have a 'multiple function' unit that we should adjust in order to create a particular building block. An analogy for the first case is a truck (lorry) platform – it is a standard construction which is upgradeable in different ways depending on what is to be transported (flowers, animals, cars, and so on). An analogy for the second case is a universal plug adaptor – it has in it a number of functionalities that might be adjusted in one way or another depending on the particular purpose of use: in Europe, USA, or Japan, for example.

![Diagram of re-usable building blocks](image)

Figure 4.13: Re-usable building blocks

Following the 'classical' OO terminology [Booch 1994; Rumbaugh 1991], it is suggested that the first of the mentioned types of re-usable building blocks is called 'general building block' and the second type – 'generic building block'. This is illustrated on Figure 4.13.
These basic principles could be applied to the Business Com ponent concept within the SDGC framework, bringing about possibilities of re-using Business Com ponents. If general or generic Business Com ponents are identified, they could be re-used in the specification of different software artefacts; this could be realized either by extending a general Business Com ponent or by parameterizing a generic Business Com ponent, as depicted in Figure 4.14.

Figure 4.14: Extending a general Business Com ponent (left) or parameterizing a generic one (right)

General Business Com ponents are models which reflect core issues and can be extended in a number of directions. For example, a general brokerage model could be further developed – in one way for building an e-trade system and in another, for building a hotel reservation system. Hence, the particular extension of a general Business Com ponent is motivated by the purpose of use. On the contrary, a generic Business Com ponent should contain in itself more than one optional functionalities. Through parameterization, such a com ponent can be adjusted depending on the desired purpose of use.

Two illustrative examples are considered below. They are built using the graphical notations of DEMO [DEMO].

Figure 4.15 depicts a general (payment) Business Com ponent. It could be extended by the designer depending on what particular payment system is to be modeled, for example e-trade payment unit, reservation payment unit, and so on.

Figure 4.15: Example of a general Business Com ponent

The model grasps the general situation in which one party provides a service to the other party and requests payment in return. Therefore, in the general case, we are to consider three role-types – the customer from one side and the service provider and payment controller – from the other side (Figure 4.15). On the figure, the external data bank PB01 concerns the data about realized services. Based on
considering this data, the payment controller (which is (in this case) modeled as periodically activateable) will charge the customer accordingly. Hence, the payment controller should be aware of the services realized (this is indicated by the dashed line between it and the service provider). As for the Transactions, they are specified below as follows (based on the above information):

<table>
<thead>
<tr>
<th>transaction type</th>
<th>result fact type</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 service provision</td>
<td>F1 service &lt;S&gt; is delivered</td>
</tr>
<tr>
<td>T2 payment</td>
<td>F2 the fee for period &lt;P&gt; by Customer is paid</td>
</tr>
<tr>
<td>T3 payment controller’s activation (periodically)</td>
<td></td>
</tr>
</tbody>
</table>

Thus, the designer would extend such a model by adding elaboration regarding the service delivery mechanism and Provider-Customer relations to be modeled. This is not just an instantiation (filling in particular values for the customer and the service provider) because the designer might essentially update the model, by adding new role-types and/or Transaction-types, for example.

Figure 4.16 shows a generic model. Since the purpose is just illustrative, the model is incomplete, depicting only some of the issues to be modeled. A BAN system is modeled. BAN stands for Body Area Network. According to AWARENESS [AWARENESS] a BAN is a device network which is worn on the body and which moves around with the person (that is, the wearer is the unit of roaming). A BAN incorporates a set of devices which perform some specific functions and which also perform communication, perhaps via a central controlling device. Devices may be simple devices such as sensors or actuators, or more complex multimedia devices such as cameras, microphones, audio headsets or media players such as MP3 players. The central controlling device may perform computation, coordination and communication functions.

Such a BAN is owned by a health organization and operates with the goal of facilitating patients. For the purpose of this example, we consider just the cases in which a BAN supports two types of patients:

- pregnant women (the ‘Gynecology’ case);
- epileptic patients (the ‘Epilepsy’ case).

In the Gynecology case, the BAN performs tele-monitoring. This includes delivering vital signs (such as blood pressure, for instance) to a caregiver (in all BAN cases, the caregiver and the patient are supposed to be in different locations) who in return determines the diagnose/situation. It is interpreted by the BAN which (based on that) provides health instructions to the particular woman (for example, ‘stop moving’, ‘ask for emergency help’, and so on).

In the Epilepsy case, the BAN performs tele-treatment. This includes delivering vital signs to a caregiver who in return prescribes treatment (for example, ‘inject medicine X’). The prescription is interpreted within the BAN which (based on that) provides treatment instructions to an actuator (a tele-treatment device (within the BAN) which is attached to the patient’s body and can execute injections, for example).

Hence, we consider (in this example) the modeling of a BAN Gynecology system (GBAN) and of a BAN Epilepsy system (EBAN). By studying the two situations, it has been found out that some issues are the same no matter if we specify a GBAN or an EBAN.

The things staying the same are the following:

- The BAN (its distribution unit) should deliver health information to a caregiver;
- In order for the BAN’s distribution unit to deliver the mentioned information, it must have received messages (sent from a measurement unit inside the
BAN) containing sampled data on conducted measurements of the patient’s vital signs.

- In order for the measurement unit to sample and send the mentioned messages, it must have the vital signs delivered from the patient.

However, besides these, there are some other issues which are different for the GBAN and EBAN cases.

- In the GBAN case:
  - The BAN’s distribution unit should deliver health instructions to the patient;
  - In order for the distribution unit to deliver the mentioned instructions, it must have received a diagnosis/statement from the caregiver.

- In the EBAN case:
  - The treatment executor (within the BAN) should perform treatment to the patient;
  - In order for the treatment executor to perform treatment, it must have received a treatment specification from the BAN’s distribution unit;
  - In order for the distribution unit to deliver treatment specification, it must have received a treatment prescription from the caregiver.

Therefore, taking into account the existing commonality between the GBAN and EBAN cases, we propose using a re-usable Business CoMponent. One possibility is to capture the common issues in a general Business CoMponent and allow for extending this CoMponent either towards GBAN or towards EBAN. However, since we know as well both the GBAN-specific issues and the EBAN-specific ones, it would be wiser to identify a generic Business CoMponent which would be (thus) straightforwardly adjustable by the designer for either of the two (GBAN/EBAN) cases.

Hence, we build a CoMponent, where we reflect the core issues (valid for both GBAN and EBAN), the GBAN-specific issues, and the EBAN-specific issues. The Transactions’ specification is as follows:

<table>
<thead>
<tr>
<th>transaction type</th>
<th>result fact type</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 h. inf. delivery</td>
<td>F1 h. inf. regarding patient &lt;P&gt; is delivered</td>
</tr>
<tr>
<td>T2 meas.inf. delivery</td>
<td>F2 data sampled and sent regarding patient &lt;P&gt;</td>
</tr>
<tr>
<td>T3 signals collection</td>
<td>F3 vital signs are collected from patient &lt;P&gt;</td>
</tr>
<tr>
<td>T4 instr. delivery</td>
<td>F4 instructions to patient &lt;P&gt; are delivered</td>
</tr>
<tr>
<td>T5 diagnosing</td>
<td>F5 diagnosis regarding patient &lt;P&gt; is provided</td>
</tr>
<tr>
<td>T6 treatment</td>
<td>F6 treatment to patient &lt;P&gt; is realized</td>
</tr>
<tr>
<td>T7 tr. spec. generation</td>
<td>F7 tr. specific. about patient &lt;P&gt; is generated</td>
</tr>
<tr>
<td>T8 tr. prescr. delivery</td>
<td>F8 tr. prescr. about patient &lt;P&gt; is delivered</td>
</tr>
</tbody>
</table>

For the purpose of this example, we have done the business description to be directly usable for identification of Transactions. However, in many cases, it is not so straightforward to adequately derive the Transactions. Anyway, we concentrate now just on illustrating a generic Business CoMponent. How we arrive at a list of Transactions starting from a unstructured business description, is considered in Chapter 6.

The identified Transactions are reflected in the DEMO Coordination Structure Diagram represented on Figure 4.16. As seen from that figure, we have put the Transactions in three groups:

1. Transactions 1, 2, and 3 reflect those issues (as it is according to the above description) which are commons for both cases (they are the core issues) – in the upper side of the figure.
2. Transactions 4 and 5 (dashed background) are the GBAM-specific ones.
3. Transactions 6, 7, and 8 (dotted background) are the EBAN-specific ones.

Therefore, we allow the designer with two options, either to take the core Transactions together with the GBAN-specific ones, adjusting in this way the generic Business CoMponent into a GBAN coMponent (Option 1), or to take the core Transactions together with the EBAN-specific ones, adjusting in this way the generic Business CoMponent into an EBAN coMponent (Option 2).

So, we have demonstrated (with these two examples) the benefits of applying re-use. In both cases (the ‘payment’ case and the ‘BAN’ one) there is a commonality concerning a number of situations to be modeled. For this reason, re-use is motivated, allowing the designer not to start from the scratch every time but re-use instead already created modeling outputs. However, unlike the first example in which we captured a core payment functionality and reflected it in a general Business CoMponent to be extended for whatever payment models might need to be specified, the second example allowed for capturing not only a core piece of functionality but also two optional ones – hence we built a generic Business CoMponent. The difference is that it could be reflected much easier in an actual
(GBAN or EBAN) model, by just adjusting it (removing the irrelevant Transactions and role-types).

In summary: within SDBC, it is possible to derive a Business CoMponent in three ways: either in the trivial way (by building a model corresponding to a Business Process), or by extending a general Business CoMponent, or by parameterizing (adjusting) a generic Business CoMponent (Figure 4.17).

![Diagram](image)

**Figure 4.17: Deriving a Business CoMponent**

**Re-using a Business Process** within SDBC includes describing a Business Process (defined in Chapter 2) at a general level, making such a description sufficiently abstract so that re-use is possible. This is illustrated on Table 4.1. The left column depicts a general Business Process concerning service arrangement. Hence, only some general Transactions are considered: 'registration', 'payment', 'reduction approval', and so on.

<table>
<thead>
<tr>
<th>general bp</th>
<th>special bp #1</th>
<th>special bp #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrangement of a service</td>
<td>hotel reservation arrmgmnt.</td>
<td>auto insurance arrmgmnt.</td>
</tr>
<tr>
<td>Registration</td>
<td>no registration</td>
<td>registr. in ins. company</td>
</tr>
<tr>
<td>Payment</td>
<td>payment of a deposit</td>
<td>insurance payment</td>
</tr>
<tr>
<td></td>
<td>pmnt. adminstr. costs</td>
<td>pmnt. adminstr. costs</td>
</tr>
<tr>
<td>reduction approval</td>
<td>early booking rdctn appr.</td>
<td>No-claim rdctn. approval</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Table 4.1: Re-using a general Business Process**

This general Business Process could be reflected in different special Business Processes, by adding some particular 'content' to the general process description. For
example (as represented in the second column), we could instantiate this general description for the purpose of hotel reservation services, by elaborating the Transactions (from 'payment' to 'payment of a deposit' and 'payment of administrative costs'; from 'reduction approval' to 'early bookings reduction approval', and so on). It is also possible that Transactions are included in the special Business Process, which have not got their origin from the general Business Process. Analogously, the left column presents an instantiation of the general Business Process for the purpose of an auto insurance service.

Consequently, Business Processes could be re-used during the early business analysis, and this facilitates the modeling work. However, re-using Business Processes does not have a significant role within SDBC especially as long as the business/software alignment is concerned.

4.2 SDBC - OUTLINE

Based on the essential SDBC fundamentals which have been introduced, this section will briefly outline the approach. Two graphical techniques have been developed for that purpose, namely the Activity Model and the Input/Output Model. The development of such techniques was considered necessary because neither of the popular ones (Activity Diagram, Flow charts, Petri Net, IDEFo and so on [Hommes 2004]) proved to be sufficiently effective for thoroughly representing the steps of SDBC, by providing information on both the dynamics of the activities to be realized and the inputs and outputs of each of them. It is particularly useful not only that the Activity and the Input/Output models provide views in these two essential directions (mentioned) but also that these two graphical techniques are completely consistent with each other.

Hence, the dynamic and 'input-output' aspects (which are essential for describing a modeling process [Shishkov and Dietz 2004-1]) are soundly matched between the two models. Next to that, the Activity Model itself is sophisticated in terms of dynamics (it supports parallel processes, two types of synchronization, and so on, to be seen from the legend further on). Tools like Activity Diagram [OMG 2000] or Petri Net [Petri Nets] are sound and much effective just as long as dynamic aspects are concerned. Tools like IDEFo offer a broader perspective [Hommes 2004]. However, neither of the popular tools is both powerful in terms of dynamics and capable of adequately matching this to input/output information. This has motivated (as stated above) our introducing the Activity Model and the Input/Output Model.

The Activity Model (Figure 4.18) represents the dynamics of the activities to be realized in applying SDBC; the Input/Output Model (Figure 4.19) represents the inputs and outputs of each activity. The legend regarding the graphical representation of these tools is as follows:
We will first consider the SDBC Activity Model, depicted on Figure 4.18. There are nine activities on the figure, and also four minor activities (they are not assigned a number; their names are backgrounded in grey).

There are three decision points and a point to which a sufficient number of iterations have to be made before proceeding further. There are two OR synchronization bars: the first one is associated with the IN points 'A' and 'B' (the AB bar); the second one is associated with the IN points 'E', 'F', and 'G' (the EFG bar). There is an AND synchronization bar; it is associated with the IN points 'C' and 'D' (the CD bar). There is a trigger to the application of SDBC, pointing to Activity 1 ('information structuring'). The last activity from the model is Activity 9 ('integration'). Activity 1 and Activity 9 are thus assigned 'start' and 'end' labels, respectively.
The SDBC Approach

Figure 4.18: SDBC – Activity Model

The trigger is pointing to Activity 1. It is about the information structuring, concerning a focused structured description of the target business reality; this includes thus a delimitation step (Figure 4.4). Then we arrive at the first decision point ("conduct Business Process generalization?"). There a decision is to be made on whether the mentioned structured business reality description should be used for the specification (modeling) of a particular Business Process (e.g. hotel reservation match-making), as reflected in Activity 2 ("identification of a Business Process"), or whether the description is to be used for achieving a generalized view (e.g. match-making), as reflected in Activity 3 ("generalization of a Business Process") and illustrated in Table 4.1. This decision should be based on certain criteria discovered in the process of studying the particular domain. For example, it might be known that an issue is unique for a company and thus, there is no sense to develop a generalized model of it. As seen from Figure 4.18, such a Business Process generalization (Activity 3) could be realized not only based on a structured description of the studied Business System but also based on the specification of a
particular Business Process (this should be done if a generalization of such a specification will be also needed further by the modeler). That is why both before and after Activity 2, it is allowed for reaching the ‘AB’ synchronization bar which leads to Activity 3.

As also seen from Figure 4.18, a model of a particular Business Process (realized within Activity 2) might be used as well for building a generic Business CoMponent (Activity 4), as it is according to the second decision point (‘model a generic Business CoMponent?’), in particular if the process flows towards the ‘CD’ synchronization bar. Otherwise, the process would flow towards the minor activity ‘MODELING’, from where we arrive at Activity 6 (‘constructing a Business CoMponent’), through the EFG synchronization bar. This reflects the situation in which no re-use is realized – we just specify a Business Process (Definition 2.6) and reflect it into a Business CoMponent (Definition 2.11). The re-use facilities of SDBC relate to Activities 3, 4, and 5.

As for Activity 3, after it there follows the third decision point (‘model a general Business CoMponent’). There a decision is to be made on whether a general Business CoMponent is going to be modeled; a general model of a Business Process is sufficient for building a general Business CoMponent [Shishkov and Dietz 2004-1]. If Yes, Activity 4 (‘modeling a general Business CoMponent’) is reached, leading afterwards to the minor activity ‘EXTENSION’, from where we arrive at Activity 6 (‘constructing a Business CoMponent’), through the EFG synchronization bar. Otherwise the ‘CD’ synchronization bar is reached. It leads to Activity 5 (‘modeling of a generic Business CoMponent’). As seen from the figure, for modeling such a component, the required input is a specification of at least two (seen from the “2” at IN point ‘D’) models of particular Business Processes AND a general Business Process specification (model). The reason is that the generic model would require not only a general specification which captures ‘core issues’ (derived from a generalized Business Process model) but also at least two particular Business Process specifications to be related to (at least two) selection options (options to be selected by parameterizing the model); actually, the rationale for using generic modeling patterns (that capture, as discussed already, several possible design outputs based on grasped core issues) is that the modeler would be able to easily adjust the generic pattern, arriving at either of the optional design outputs offered by the pattern. After Activity 5, the process flows towards the minor activity ‘PARAMETERIZATION’, from where we arrive at Activity 6 (‘constructing a Business CoMponent’), through the EFG synchronization bar.

Thus, the EFG synchronization bar reflects the three ways of deriving (within SDBC) a Business CoMponent: either without realizing re-use (by reflecting a Business Process model in a Business CoMponent), or by extending a general Business CoMponent, or by parameterizing a generic Business CoMponent. This has already been considered in Sub-section 4.1.4. So, this representation (Figure 4.18) is consistent with what is presented in Figure 4.17.

A constructed Business CoMponent is then to be reflected in a software specification model; hence, we arrive at Activity 7 (‘deriving a software specification model’). A sound mapping is to be accomplished allowing for a precise reflection between the two. Both the Business CoMponent and the resulting software specification model should undergo validation concerning at least structural and dynamic issues [Shishkov et al. 2003-2]. This is indicated by the label ‘Validation’, positioned along the line between Activity 6 and Activity 7.

As for the software specification model, as mentioned before, depending on the granularity of the originating Business CoMponent, the model could or could not refer
to a particular Software CoMponent (Figure 4.12). The question of software granularity is to be addressed particularly from the perspective of the software system-to-be. Usually, as discussed in sub-section 4.1.4, a derived software specification model is to be reflected in more than one Software CoMponents. So, progressing from Activity 7 to Activity 8 ("elaboration") comes through the minor activity 'DECOMPOSITION' (indication of the need to decompose the software specification model into more than one Software CoMponents). However, in the cases in which no decomposition would be necessary, the software specification model is considered itself being a Software CoMponent.

Once identified, a Software CoMponent needs to be specified in more detail – further elicitation should be provided concerning the CoMponent's entities and interactions [Cockburn 2000; Shishkov and Dietz 2003-3; Insfran et al. 2002]. So, once identified and specified, a Software CoMponent should undergo elaboration (Activity 8). This means adding more relevant aspect models so that all necessary design perspectives are 'covered'. More explanations on this follows in Chapter 6.

After a sufficient (see below) number of Software CoMponents have been identified, specified, and elaborated, they should be integrated (Activity 9) in the process of specifying the functionality of the software system-to-be. Hence, there is a more special relation between Activity 8 and Activity 9; an indication for this is the symbol positioned on the line between these activities, showing that many Software CoMponents would be necessary that would represent together a sufficient input for specifying a complete model of the software system-to-be. However, it is often not easy to provide guidelines on how to decide what particular Software CoMponents represent a sufficient input for specifying the software system-to-be; this decision is often subjective and/or intuitive; anyway, we adopt in SDBC the relevant general guidelines of Atkinson [Atkinson and Muthig 2002], related to the component-based Product-line Engineering [Atkinson et al. 2001].

So, after considering the (SDBC) Activity model, we proceed to the (SDBC) Input/Output model. It is depicted in Figure 4.19. As seen from the figure, the starting input for applying SDBC is any (informal, unstructured) description of the business system to be considered (Input 1.1), including domain-imposed requirements representing norms [Stamper 1996] (they could be expressed using Norm Analysis [Liu 2000]). The description might be textual or it might be a graphical model, a conversation or any other form. The first activity's output (Output 1.1) should be a structured description of the studied system. This description should thoroughly reflect the considered business reality; next to that, the description must be precisely delimited, as mentioned above. As seen from the figure, such a structured and delimited description might be stored in a bank (D bank) from where to be usable also in other relevant modeling tasks.

Such a description could be used as an input for either Activity 2 (Input 2.1) or Activity 3 (Input 3.1) (either for identifying a Business Process or for building a generalized Business Process model). Building a generalized Business Process model could be done as well based on an identified Business Process (Input 3.2). An indication for this is the line between Activity 2 and Activity 3.
Figure 4.19: SDBC – Input/Output Model

A generalized Business Process model could be stored in a bank (P bank) for multiple uses. It could also be used as an input for constructing (Activity 4) a general Business CoMponent (Input 4.1). As seen from the figure, general Business CoMponents could also be taken from an external bank (C1 bank) (Input 4.2). A constructed general Business CoMponent could be either stored in a bank – C1 bank
(for use in other project(s)) or used as an input for the construction (Activity 6) of a Business CoMponent (Input 6.2). As seen from Figure 4.18, this comes through extending the general Business CoMponent.

Regarding the modeling of a generic Business CoMponent, it should be based on a generalized Business Process model AND at least two (Figure 4.18) models of particular Business Processes; this concerns Input 5.1, Figure 4.19. Generic Business CoMponents could also be taken from an external bank (C2 bank). As seen from Figure 4.19, a constructed generic Business CoMponent could be either stored in a bank (C2 bank) (for use in other project(s)) or used as an input for the construction (Activity 6) of a Business CoMponent (Input 6.3). As seen from Figure 4.18, this comes through parameterizing the general Business CoMponent. And finally, as seen from Figure 4.19, the third possible input (Input 6.1) for the construction of a Business CoMponent is a Business Process model (Output 2.1).

Deriving a software specification model (from which Software CoMponents could be identified, by applying decomposition, as already mentioned) is based either on a Business CoMponent constructed in the above proposed way (Input 7.1) or on import of Software CoMponents from an external bank (Input 7.2).

Each of the derived Software CoMponents should be elaborated (Activity 8; Input 8.1) in terms of structural, dynamic, and data aspects (in order to bring sufficient elicitation for the further software design activities, as already mentioned; this will be considered in detail in Chapter 6, as mentioned above) and stored in a bank (S bank). From there, Software CoMponents will be taken (Input 9.1) and integrated for the purpose of specifying the software system-to-be.

A specification model of a software system represents the final output (Output 9.1) of the SDBC approach. Hence, the end point is reached and this is indicated by labelling Activity 9 with 'end', as stated already.

4.3 CONCLUDING REMARKS

In this chapter, we have introduced the foundations of the SDBC approach. We did this, by deriving (based on the problem statement and research hypotheses (Chapter 1)) the significant desired properties concerning SDBC, and reflecting them in the determination of four essential SDBC foundations (Figure 4.2). We have 'zoomed in' in each of these four directions, identifying particular problems there and proposing ways of resolving them. Based on this, we have determined the approach outline, presenting it in Section 4.2. There we have elaborated on the essential (proposed) modeling activities to be carried out in accomplishing the SDBC goals. These activities have been considered at the conceptual level, not at the application level (actually, in Chapter 6 we are going to derive (based on this conceptual background) step-by-step methodological guidelines concerning the application of SDBS – these guidelines will include concrete modeling steps to be realized using particular modeling tools, in applying the approach).

Figure 4.20 summarizes the essential issues behind SDBC:

- The input for applying SDBC consists of descriptive information about the business reality + domain-imposed requirements (Figure 4.19) as well as (user) requirements towards to software system-to-be (Figure 4.12).
- The output of SDBC represents the specification model of the software system-to-be, which is consistent with the current software design standards (Figure 4.6).
• In realizing its goal, SDBC steps on the background of four essential fundamentals (Figure 4.2), namely:
  - integrated view over Business Process modeling and software specification;
  - four Business Process modeling aspects (perspectives), including the ‘classically’ considered structural, dynamic, and data perspectives [RMODP Part 1] as well as the communication perspective (justified in Chapter 1) which has particularly been addressed in Sub-section 4.1.2;
  - component-based business/software alignment;
  - re-use of general and generic modeling patterns in the modeling process.
• Next to that, SDBC is supported by a particular modeling process whose concepts have been elaborated (Figure 4.18; Figure 4.19).

![Diagram of SDBC - Summary](image)

**Figure 4.20: SDBC - Summary**

Considering the SDBC foundations, presented in the current chapter, Chapter 5 will propose a selection of modeling tools which are suitable for making SDBC operational. And further on, Chapter 6 will define methodological guidelines for applying SDBC – a step-by-step overview on what and how is to be done.
Chapter 5

MAKING SDBC OPERATIONAL

Science is organized knowledge.
Immanuel Kant

Having introduced (in the previous chapter) the foundations of the SDBC approach, we will now focus on making it operational. This means proposing a coherent set of modeling tools through which the SDBC conceptual framework could actually be applied. Hence, the current chapter identifies appropriate tools through which the framework could be applied, explains why these tools are considered adequate for that purpose, and approaches the problem on how to properly combine them.

Thus, the current chapter, together with Chapter 4, represent the necessary input for deriving the step-by-step SDBC application methodological guidelines (to be introduced in the following chapter). Said otherwise, in order to specify guidelines on how to apply SDBC, we need the SDBC conceptual framework as well as a selection of modeling tools which are adequately relatable to the major tasks within the framework, and which are also useable together in concert.

The current chapter will address only the tools having major significance for SDBC. Some tools with minor significance for the approach will be considered directly in Chapter 6.

The first section of this chapter addresses general issues, mainly the tool selecting criteria followed. Section 5.2 considers some particular tools’ suitabilities concerning the application of SDBC, and Section 5.3 contains concluding remarks.

5.1 SELECTING APPROPRIATE TOOLS: GENERAL ISSUES

The modeling tools through which a particular approach is to be applied, should be precisely selected, analysed, and considered. Regarding the particular (proposed) application of SDBC through a number of tools, a tools selection has been carried out. The selection has been based on several demands:

- The tools’ possessing sound theoretical roots has been considered to be a significant issue concerning the selection. Such roots would make sure that the particular role (in the SDBC application context) of a tool has been clearly envisioned. Next to that, a conceptual background would be a guarantee that the desired modeling outputs (related to a tool) are adequate and doable.
Chapter 5

- The selection of tools has been considered in the context of particular inter-tool mapping potentials. Said otherwise, one tool alone would be insufficiently capable of handling the application of SDBC (if this was the case, SDBC itself would not be necessary, obviously); thus a number of tools are to be jointly applied, which means that the mapping among them would be as important as the tools themselves. Even if a tool is capable of providing a useful modeling output, the overall usefulness would be low unless this output is adequately mapped towards further modeling activities.
- It has been demanded that each tool’s (modeling) output is relevant to a particular SDBC desired output (Figure 4.19). Otherwise, it would be difficult to consistently match the SDBC conceptual framework (Chapter 4) to particular tools.
- It has been demanded also that each tool’s output is adequately validateable. In other words, mechanisms should exist, allowing for validating (at least in terms of structure and dynamics, as it is according to the current practices [Barjis et al. 2002]) the output achieved through a modeling tool.
- Another demand is that the application of the SDBC approach should be UML-driven. We will elaborate below on the choice of UML as the SDBC software design tool.

Current software systems are built by a multitude of modelers and developers distributed across different locations and possessing different types of knowledge and expertise. Therefore, it is crucial that the software artefacts are being designed using common, unified, standard modeling tools and environments that are known by as broad circles as possible [Shishkov and Dietz 2002-4]. The Unified Modeling Language – UML is becoming de facto the standard language for modeling software systems, widely recognized by both researchers and practitioners [Cockburn 2001; Atkinson and Kuhre 2002; Shishkov and Dietz 2002-3]. For this reason, it has been suggested basing SDBC (as an approach concerning the specification of software) on UML. Moreover, the completeness of UML allows for a thorough consideration of the software specification task.

In summary, the above stated considerations have affected our identifying the set of modeling tools to be used in order to apply the SDBC approach in a proper way.

5.2 FOCUSSING THE SELECTION

In the previous section, we have clarified the criteria followed in selecting modeling tools. As mentioned before, this selection directly concerns the SDBC conceptual framework (Chapter 4). In the current section, we will address the major tools selected. They, of course, should relate to:
- Business CoMponents’ specification, elaboration, and validation;
- Software CoMponents’ derivation.

In the context of the conceptual SDBC framework and based on information about existing business/software modeling tools (Chapter 3), as well as following the above-mentioned criteria, we are proposing the following selection of tools:

1. The structural and communication aspects of a (identified) Business CoMponent are specified, elaborated, and validated using DEMO.
2. The dynamic aspects of a Business CoMponent are specified and elaborated using Petri Net.

3. The structural/dynamic aspects of a Software CoMponent are specified and elaborated using UML.

4. The dynamic aspects of a Business/Software CoMponent are validated using discrete event simulation, and in particular – ARENA.

4. The data aspects of a Business CoMponent are specified and elaborated using ORM.

5. The mapping towards a (UML-based) software specification model is facilitated using use cases.

This is illustrated in Figure 5.1, where we have used the following abbreviations: 
\textit{bk} - Business CoMponent; \textit{ssm} - software specification model; \textit{sk} - Software CoMponent; \textit{sc} - Software Component.

As seen from the figure, the initial SDBC modeling steps should result in the identification of Business CoMponent(s) (bk). We would need to soundly model them in terms of structure, dynamics, data, and communication aspects (Chapter 4). As mentioned before, we claim that DEMO could be useful for grasping the structural and communicative issues. The reasons are not only in its full consistency with a well-established communication theory, namely the Language-Action Perspective (Chapter 3), but also in its capability of grasping structural and communication aspects, representing them in adequate models supported by sound graphical notations. As for the bk dynamic elaboration, we have selected Petri Net not only because (as it will be briefly discussed) this is the most popular and widely used dynamic modeling tool but also because of it adequate relatability with DEMO. Ideas of combined DEMO – Petri Net application have been suggested [DEMO] and further explored by Shishkov and Barjis (2002). As for the bk data elaboration, ORM has been suggested as a sound factual tool possessing relevant theoretical background and combinable with DEMO as studied by Dietz and Halpin (2004). Regarding the bk validation, DEMO possesses structural and communication validation mechanisms in itself, as it is well-known from numerous DEMO-related materials and examples [DEMO]. As for the bk validation in terms of dynamics, we have suggested using discrete event simulation and in particular, the ARENA tool [ARENA]. This choice relates not only to the straightforward derivability of an ARENA model from a Petri Net dynamic bk model but also to the great validation potential of simulation models in general and ARENA models, in particular [Shishkov et al. 2003-2; Barjis and Shishkov 2001-1]. These potentials could be seen from the useful visualization and animation mechanisms, applied to complex dynamic structures requiring adequate elicitation.

As seen from the figure: software specification model(s) and Software CoMponents should be derived from a Business CoMponent.

Regarding the mapping towards a UML-driven software specification model(s), the necessary 'link' are to be use cases [Shishkov and Dietz 2004-3]. Use cases and UML, and their relevance, have been already thoroughly considered within the thesis and we are therefore not going to add more information on that in the current section.
Summarizing: in the above paragraphs, we have formulated our choices for the major modeling tools claimed to be suitable for the application of the SDBC approach. As already stated, some tools with minor significance will be directly introduced in Chapter 6, as part of the step-by-step methodological SDBC application guidelines. It is to be noted that the identification of Business CoMponents is not viewed as being based on a particular tool; this identification will be considered in Chapter 6.

Further in this section, we will motivate our selection proposition.

### 5.2.1 DEMO – Suitabilities for SDBC

We will consider below the strengths of the DEMO methodology (based on the analysis of DEMO (Chapter 3)), especially in the context of its suitability for being applied within the SDBC approach. Firstly, the essential strengths of DEMO will be briefly discussed. Following this, it will be studied what are the particular strengths of DEMO in the context of SDBC.

**Three fundamental strengths of DEMO**

Considering in general the DEMO conceptual background that has already been outlined (Chapter 3), three fundamental strengths can be identified (Figure 5.2); these strengths are also addressed in [Dietz 2003-2].
Figure 5.2: Three fundamental strengths of DEMO

*Essence* concerns the ability of DEMO to identify the real business issues, correctly distinguishing them from the informational issues. For example, realizing a hotel booking is essential, but calculating the accommodation costs is informational.

*Atomicy* concerns the ability of DEMO to capture things that are units from the business point of view; actually DEMO allows modelers to consider the right granularity level of detail. For example, although to realize a hotel booking is related to the performance of many non-essential actions (sending reservation information via fax or e-mail, calculating a deposit that is to be paid, and so on), DEMO captures the hotel booking as a modeling unit since it is atomic from the business point of view.

*Completeness* concerns the ability of DEMO to reflect completely and exactly the business reality — no business things are overlooked and the DEMO models do not contain irrelevant things.

These three fundamental strengths, resulting from DEMO, are of crucial importance for realizing a consistent Business Process modeling. Actually, very few Business Process modeling approaches have these capabilities [Dietz 2003-2]. As already stated, DEMO is a methodology which aims at modeling the essential Business Processes, abstracting from their realization, and this is a basic reason for considering DEMO within the SDBC framework.

**The role of DEMO in the SDBC approach**

On the basis of considering:
- the particular demands within the SDBC approach, determined in the previous chapter;
- the DEMO methodology that has already been discussed (Chapter 3);
- the fundamental strengths of DEMO (outlined above),
the particular role of DEMO (illustrated on Figure 5.3) within the framework of the SDBC approach is specified.
As seen from the figure, DEMO plays role in two crucial directions, namely concerning the structural and communicative Business Process modeling aspects. DEMO also plays a supportive role (in general) in the initial Business CoMponent specification.

DEMO possesses definite strengths concerning the specification of a Business CoMponent, namely essence, atomicity and completeness; these strengths guarantee that a DEMO-driven Business CoMponent adequately and completely reflects the business reality. Instead of providing 'extensions' of instruments for modeling information processes, DEMO considers Business Processes as an independent field of study and thus provides an instrumentarium that is applicable especially in the context of Business Process modeling. This is of great value for realizing a consistent, business-focused model, guaranteeing that the further software specification activities will not miss some essential issues from the target business reality. Next to that, the sound graphical notations of DEMO [Dietz 1999; Shishkov 2003] allow modelers to represent and visualize their modeling activities in good notations.

As for the structural Business Process modeling perspective, the DEMO Coordination Structure Model is adequate [Shishkov and Dietz 2002-3] for reflecting actions and corresponding actor-roles. The model is rigorously constructible based on identified Transactions. It is usable (as it will be discussed further in the thesis) as a source for derivation of use cases. The model is relatable to other relevant aspect models, for example based on Petri Net or ORM.

As for the communication Business Process modeling perspective, DEMO offers possibilities for sound and thorough elicitation. This distinguishes DEMO from most of the widely used currently Business Process modeling tools. The capabilities of DEMO concerning this aspect are originated in the DEMO Transaction concept. It has been introduced in Chapter 3 and interpreted in Chapter 4. The Transaction instrumentarium and also its consistency with the DEMO Coordination Structure Model make DEMO uniquely capable of grasping the structural aspects of a given business reality while at the same time be precise regarding the communicative issues concerning it.

Applying Organizational Semiotics, and in particular Norm Analysis [Liu 2000; Stamper 1997; Shishkov et al. 2003-3] in support of transaction elicitation is claimed to have significant value and will be illustrated in Chapter 7.

DEMO, thus, is to play central role in the application of the SDBC approach. SDBC could benefit from DEMO as far as Business CoMponents and their specification
(and elaboration) are concerned. The sound theoretical foundation of DEMO (Chapter 3) brings about validation values to the models built using DEMO.

Regarding the mapping towards software specification, and in particular the problem of deriving use cases, as long as the UML-based software design is concerned, a definite strength of DEMO is the possibility use cases to be consistently derived based on a DEMO model. This has been studied [Shishkov and Dietz 2003-1] and will be elaborated in the following chapter.

In summary, allowing for creation of consistent Business Process models that are relatable to prior Business Process identification activities and reflectable in the derivation of use cases, DEMO represents a useful Business Process modeling tool to be applied within the framework of the SDBC approach.

5.2.2 Petri Net – Suitabilities for SDBC

Based on the analysis of Petri Net (Chapter 3), the consideration of possibilities for combined application of DEMO and Petri Net, and addressing the SDBC demands concerning the modeling of dynamic Business CoMponent aspects (Chapter 4), we propose that Petri Net (PN) is used on the basis of DEMO as a tool to model the overall dynamics (behavior) of a Business CoMponent. Our motivation on this choice is presented below.

According to Peterson (1981), PNs are appropriate to model the dynamics of a business reality, while allowing for a sound analysis/design as part of it. As studied in [Jensen and Rozenberg 1991; Aalst and Best 2003; Shishkov and Barjis 2002], PN is applicable in a number of business domains which makes it widely usable for Business Process modeling in general. This results mainly from the graphical facilities of PN [Shishkov and Barjis 2002]. Petri Nets have rich graphical notions and a well-defined semantics allowing formal analysis and building models which are easy to read and understand.

PN models are not only easily understandable but also powerful in terms of workflow representation potentials [Aalst and Best 2003]. PN could be especially useful if the modeling needs to capture concurrency, parallelism, and synchronization. As concerns complex processes, PNs use a number of sub-models which are combined into a higher hierarchy making the overall model more compact and manageable.

As for the combinability with DEMO, studies have been conducted, supporting the claim that such a combination is both adequate and useful. According to [DEMO] PN models are consistently relatable to and derivable from DEMO Business Process models.

As for the validation of PN models, it has been studied that such validation is straightforwardly realizable. According to [Barjis and Shishkov 2001-1; Barjis and Ilkov 2000], PN-based Business Process models are soundly validatable using discrete event simulation. Simulating PN-based models would not only soundly validate the dynamics of a Business CoMponent, but also complement the PN model by adding visualization and animation elicitation.

In summary, PN's comprehensive graphical notations, powerful formal specification tool, and executable tool for specification, analysis and design of systems make the tool adequate as far as the SDBC demands are concerned. The (studied) combinability of PN with DEMO (viewed as a source for building a PN model) and simulation tools (viewed as a validation facility), are additionally appreciated.
Chapter 5

We claim, for this reason, that PN is capable of adequately complementing DEMO (in terms of dynamics) in the process of Business CoMponents’ specification and elaboration.

5.2.3 ARENA – Suitabilities for SDBC

A dynamic (PN-driven) view on a Business CoMponent, needs to be adequately validated [Shishkov et al. 2003-2]. Once there is an identified Business CoMponent, it is of significant importance to trace precisely all essential dynamic issues related to the CoMponent model in order to make sure that the performance of the software system-to-be will be properly designed. Otherwise, there is a risk the software system to function inadequately.

A way to reliably and illustratably trace the interactions within a Business CoMponent is via discrete event simulation [Barjis and Shishkov 2001-1]. Analyzing systems through simulation has proved its methodological advantages [Barjis et al. 2002]. According to Tumay (1996) a Business Process simulation model can realistically capture the resource constraints, the decision rules and the stochastic behaviour of real-world processes. Furthermore, such a model can capture the behaviour of both human and technical resources in the process [Hulupic and Robinson 1998]. It can mimic the operation of the processes by stepping through the events in compressed time while displaying an animated picture of the workflow. The model output data, gathered through the simulation can be used to evaluate the performance of the processes [Ilikov 2004] and validate dynamic Business Process models [Shishkov et al. 2003-2].

On this basis, several strengths of simulation could be summarized, in concert with the studies of Giaglis and Paul (1996) as well as Law and Kelton (2000):

- A process can be conveniently represented in a simulation model as a time-ordered set of interrelated events that modify the state of an entity, which closely corresponds to the concept of a business workflow [Davenport 1993; Aalst et al. 2000].
- The capabilities to model the evolution of a process over time and take into account its stochastic characteristics are among the biggest advantages of simulation compared to other approaches which use static representation and deterministic models [Tumay 1995].
- The animation of model operation together with the interactive depiction of key model performance indicators is a powerful tool for gaining insight into the operation of the simulated processes, identifying bottlenecks and problems, and generating ideas how to solve them.
- A simulation model can easily be changed to represent different ways to organize a process, thus providing a low-cost way of estimating the effect of proposed changes to the process and for evaluating alternative process scenarios.

Thus, a way to benefit from simulation is to use it as a validation tool applied to created dynamic Business Process models, for instance - Petri Net models. In support of this is the claim [Pritsker 1992] that simulation could be used as a design assessor, evaluating proposed solutions on the basis of simulated performance measures, as well as modeling and synthesizing new alternative solutions.

For this reason, we have suggested simulation to be used as a dynamic validation tool in support of the application of the SDBC approach. Besides the above (general) arguments, we offer (below) several more specific ones:

- By validating dynamic Business Process models through simulation, modelers could have the chance to grasp the peculiarities of the system as a whole, such as: the variability and the randomness of Business Processes, as well as the
(often complex) relationships which exist among the different parts of a process [Ilkov 2004].

- Entity flow models including decision points, parallel flows, and the dynamic behaviour of a Business Process are straightforwardly reflectable in simulation models [Bhaskar et al. 1994] and therefore simulation via them is doable.

- Because of their powerful animation facilities, simulation models are easily understandable not only for the system modelers but also for potential users, giving insight not only in the performance of a system but also in the possible consequences of a proposed change in the way processes are organized [Choi and Chan 1997]. This might help modelers and potential users recognize the benefits of this change.

For all these reasons, we claim that it is feasible to expect that simulation models can be helpful for grasping and validating the dynamics of a Business CoMponent being modeled.

We consider in particular the ARENA [ARENA] simulation tool as one of the appropriate tools that can be used for validating the dynamic aspects of Business Process models. An advantageous feature of ARENA is its animation facilities. They are rich and well-formulated so that the resulting simulation models are interesting, attractive, and easy for training people as well as demonstration purposes [Barjis and Ilkov 2000].

Being a general simulation tool (as it has already been explained (Chapter 3), ARENA could bring value to the SDBC approach through its business edition which addresses Business Process simulation. The appropriateness, adequacy and suitability of applying this particular edition of ARENA for the purpose of validating the dynamics of Business CoMponents has been explored and demonstrated [Shishkov et al. 2003-2].

Thus, based on this (mentioned) adequacy and also on the sound PN-ARENA mapping allowance [Barjis and Shishkov 2001-1], we claim that ARENA could bring value to the application of the SDBC approach, concerning particularly the task of validating the dynamic view over Business CoMponents.

### 5.2.4 ORM – Suitabilities for SDBC

The ORM tool has been introduced (and also discussed), in Chapter 3. Based on the information there, we conclude that ORM is capable of soundly modeling the data aspects of a Business CoMponent. Next to that, the wide popularity and applicability of ORM as well as its being used in many cases, make it a good candidate-tool concerning the data Business CoMponents elaboration, within the SDBC.

What brings further arguments in favour of ORM (in this particular respect) is its consistent alignability with DEMO. Dietz & Halpin (2004) have proposed a mechanism for applying DEMO and ORM in concert.

Therefore, once we have specified a Business CoMponent and elaborated it in terms of structure and communication issues, it could be straightforward to add factual elicitation via ORM.

For this reason, we propose using ORM within the application of the SDBC approach for the purpose of data Business CoMponents elicitation.

### 5.2.5 Use Cases – Suitabilities for SDBC

In the previous section, we have presented arguments in favour of the claim that the current reality makes it sensible applying (within SDBC) UML-driven software specification in order to make the approach widely usable.
Use cases have been presented and studied in Chapter 3, as modeling constructs that should link the application domain (the business world) to the software domain, in a UML-driven Software Design. Hence, the SDBC-driven Business Process modeling output (presented in terms of Business CoMponent) could be consistently reflected in UML (only) via the derivation of use cases. Use cases have their place in UML, namely the UML Use case diagram, and this is a guarantee that based on a use case model, a further UML modeling and design could be realized adequately.

For this reason, there is no doubt that a SDBC Business CoMponent, specified using DEMO, Petri Net, and ORM should come through use case notations in order to find its adequate UML reflection.

In the current section, we are not going to consider use cases in more detail. They have been thoroughly introduced and analysed in Chapter 3, including in terms of their advantages. What has not been addressed in the mentioned chapter, however, is the Business Process modeling – use case link. This link relates to a current ‘bottleneck’ in the UML theory and practice, and is of significant importance for adequately aligning (also within SDBC) Business Process modeling and software specification [Shishkov and Dietz 2003-1; Shishkov and Dietz 2004-1].

An essential issue, therefore, is how to derive use cases based on Business CoMponents. Such a derivation has been addressed, studied, and demonstrated by Shishkov & Dietz (2004-3). For this reason, we claim that a mechanism exists through which to build a use case model, basing it consistently on a particular Business CoMponent which is specified in DEMO notations.

5.3 CONCLUDING REMARKS

In the previous two sections we have identified a useful set of modeling tools through which the basic tasks, concerning the application of the SDBC approach, could be realized.

As studied, DEMO could significantly contribute in this direction, especially with regard to the specification and elaboration (in particular structural and communicative) of a Business CoMponent, based on identified (business) Transactions (extended with Semiotic Norms). As studied as well, extending DEMO with Petri Net and ORM could bring elicitation in two other aspects, namely dynamic aspect and data aspect. All this brings about a possibility to soundly elaborate a Business CoMponent in all 4 perspectives (aspects), as according to Figure 1.2. This is a crucial desired property of SDBC, reflected in its conceptual framework.

According to this framework, such a 4-aspect Business Process model would need to be adequately reflected in the derivation of a UML-driven software specification model. As studied, this could be significantly facilitated by the DEMO – use case transformation mechanism [Shishkov and Dietz 2004-3, 2003-1].

The following chapter will propose, on the basis of the SDBC conceptual framework (introduced in Chapter 4) and the current analysis, step-by-step guidelines on the application of SDBC, exploring in detail how the proposed tools could be used and combined together in order to reach the SDBC design goals. Chapter 6 will elaborate also on the identification of business Transactions (which has not been covered by the current analysis mainly because no particular tool has proved to be solely sufficient for realizing that).
Chapter 6

APPLYING SDBC

The art of being wise is the art of knowing what to overlook.

William James

Chapter 4 has introduced the conceptual framework of the SDBC approach. Chapter 5 has motivated the selection of the major modeling tools through which to make the approach operational. On this basis, the current chapter will propose step-by-step methodological guidelines for the application of SDBC, elaborating as well on some modeling tools (not considered in Chapter 5) with minor importance concerning the application.

Logically, the application guidelines relate to the SDBC Input/Output Model (Figure 4.19) from where we derive four interrelated phases through which the application process should progress:

• Structuring the initial case information (in consistence with Activity 1 - Figure 4.19), following the goal of bringing order in the initially available (usually textual) case input that concerns the addressed business reality as well as the demands towards the software system-to-be.

• Identifying Business CoMponents (in consistence with Activities 2 to 6), coming through consideration of corresponding Business Processes.

• Deriving, specifying, and elaborating software specification model(s) (in consistence with Activities 7 and 8), on the basis of the identified Business CoMponents.

• Validating the built business/software models.

These four phases are reflected in Sections 6.1 to 6.4 of the current chapter; Section 6.5 will present concluding remarks.

6.1 INFORMATION ANALYSIS AND STRUCTURING

As mentioned already, the guidelines addressed in this section concern the goal of analyzing and structuring the initial case information in order to resolve vagueness, incompleteness, and errors.
We suggest achieving this in several steps:

- Identification of actor-roles as essential compositional elements concerning our adopted LAP-driven consideration of business transactions/processes/components.
- Structural elaboration concerning the inter-role relations.
- Deriving a constructional view over the considered business reality – bringing together the compositional and structural aspects.

The above suggestion is driven by the claim (Chapter 2) that it is essential to properly consider both compositional and structural system aspects as well as to adequately bring them together.

Further in the current section, we will follow the above-mentioned three steps.

6.1.1 Identifying Actor-Roles

We propose approaching the explored domain through a classical generalization hierarchy [Booch 1994; Stamper 1973], as a way to guarantee that all actor-roles are rigorously and completely identified. It is known from literature and numerous examples how to reflect any information concerning a (business) reality into a generalization hierarchy. Next to that, any actor-role should definitely correspond to an entity from such a generalization hierarchy, because an actor-role itself reflects the competence of such an entity.

Thus, we start by building a generalization hierarchy for the explored domain, positioning the essential entities under consideration in a business-object model. For example, concerning a fast-food restaurant, we could draw the following generalization hierarchy where we forbid irregularities (for example it is impossible that a restaurant is at the same time traditional and fast-food); the bold words indicate that the particular type is going to be further sub-typed:

```
PERSON
* natural person
* legal person
** company
** non-profit organization
** ...
*** manufacturing company
*** distribution company
*** financial company
*** restaurant
*** ...
**** traditional restaurant
**** fast-food restaurant
**** cantina
**** ...
```

We claim that this is not only a good foundation for discovering actor-roles (role-types, in particular) but also an appropriate initial step in conducting information analysis and structuring. Positioning the entities under consideration in such a hierarchical model reflects the natural structure of things in the considered (business) reality, as it is well-known from the OO studies [Booch 1994]. Hence, such a (constructed) generalization hierarchy puts the entire modeling process on solid backgrounds and guarantees a well-established link between the business modeling and the addressed business reality. This should concern also the actor-roles discovery (we are not going to discuss the helpfulness of considering actor-roles rather than entities; this has been done in Chapter 2). So, the identification of roles (actor-roles) is to start with a consideration of the roles which are typical for each of the (discovered) entities. We propose here a study in two directions, to be illustrated
below. If we have discovered, for example, the type MANAGER which is sub-type of
EMPLOYEE, in the context of a consideration of a company named 'ABC', we are to
realize the following:

* We should study first the particular word that labels the considered entity –
the word 'manager' (in our example). Often the word would give us a direct clue
about the role. For example, if we consider the role-type SECRETARY (again sub-type
of EMPLOYEE), we directly relate the word 'secretary' to the fundamental role which
any instance of SECRETARY should fulfill, namely secretarial activities, so we
formulate a role-type SECRETARY. However, this is not always the case. It might be
that based on a material (from which we derive information about a case) we
consider the type BUSINESS ASSISTANT; and when we analyse the corresponding
responsibilities, we discover that actually an instance of BUSINESS ASSISTANT
performs secretarial functions and we should therefore do a reflection towards the
role type SECRETARY. Thus, it is always helpful considering the words that label the
entities from our generalization hierarchy; however, in addition to this, a deeper
information analysis is to take place.

* We should study as well other eventual responsibilities associated with the
considered entity. In our example, analyzing the responsibilities' specification
concerning MANAGER, we might discover, for example, that they include driving the
company car between the offices of company ABC. Hence, if Mr. I. Ivanov is an
instance of MANAGER, then according to the mentioned specification, we would
expect to see Mr. I. Ivanov acting either as manager (performing business
administration and decision-making) or as chauffeur (driving the car of the company).
Therefore, in this situation we derive an additional role-type (in addition to
MANAGER), namely CHAUFFER.

Chapter 7 will illustrate in detail the role identification.

After having considered the role identification within SDBC, we will proceed in
the following sub-section, as stated before, with structural elaborations concerning
the inter-role relations.

6.1.2 Elaborating on the Inter-Role Relations

The proposed starting point in a study concerning the inter-role actions (relations) is
to make sure whether or not a relation exists between Role A and Role B (from now
on in this chapter, we will use 'relation' for short, meaning 'inter-role relation'). This
logically relates to information gathering (it might include analysis of materials about
a case, interviews with employees of a studied company, and so on) that aims at
clarifying between which roles there is a relation. The question to answer is clear –
'Is there a relation between Role A and Role B (Yes or No)'. Usually, answering such
a question should not cause confusion. It is quite clear, for example that there is a
relation between the role-types STUDENT and PROFESSOR (in the context of a
consideration of a university) or that there isn't a relation between the role-types
SUPPLIER and CUSTOMER (in the context of a consideration of a grocery store).
There are situations as well in which the answer is not so obvious. However, the
answer to such a question should definitely be clear if the relevant information is
considered (including through interviews) because there are only two possibilities –
either there is a relation or not, and if there is no information in the (studied)
materials about such a relation, then the answer is 'NO'.

It is necessary to further describe each of the identified relations. The reason is
that a consideration of just the role-types involved is insufficiently informative
concerning the relation. For instance, if we have a relation between the role-types
BANK and (bank) CLIENT, this relation might be either about a credit given by BANK
to CLIENT, or about a money transfer done by CLIENT via BANK, or anything else. Thus, it is needed to know not only the role-types involved in a relation but also what is the relation about. We propose addressing this issue in two steps:

**Step one.** We take as input the discovered role-types and also the information about relations among them. Within this step, we consider the role-types one by one. So, we take a role-type and aim at elaborating on it. In doing this, we first describe its usual output. For example, the role-type PUBLISHER might be associated with the output 'book'. In this way, a number of candidate associations could be formed. We take next a particular one (it is to be described by a pair of nouns, for example: 'PUBLISHER – book', since both a role and an output are describable by a noun) and try to elaborate it further, by describing how do the role-type and its output relate. Logically, this is to be done using a verb. In the above example: How do 'PUBLISHER' and 'book' relate? The answer is: 'PUBLISHER publishes a book'. Thus, in elaborating a role, we form a binary relationship (a relationship concerning two entities, which is usually described by - noun-verb-noun: the nouns corresponding to the entities and the verb – describing the relation among them). Hence, in our example we would have:

\[
\text{PUBLISHER} \quad \text{-} \quad \text{publish} \quad \text{-} \quad \text{book.}
\]

**Step two.** We match the list of inter-role relations and the identified binary relationships concerning each of the role-types, and we remove in this way all irrelevant binary relations. For example, if we have the following two binary relations about the role type BANK:

\[
\text{BANK} \quad \text{-} \quad \text{give} \quad \text{-} \quad \text{credit}
\]

\[
\text{BANK} \quad \text{-} \quad \text{possess} \quad \text{-} \quad \text{shares},
\]

and if BANK is related to the role-type CLIENT, then obviously we remove the latter of the two (above presented) binary relationships. Thus, in this way, we form more complex relations that consist of two role-type names and information about their interrelation. In this example, it would be:

\[
\text{BANK} \quad \text{-} \quad \text{give} \quad \text{-} \quad \text{credit} \quad \text{-} \quad \text{CLIENT.}
\]

Then all (resulting) relations are to be considered as a whole, being checked against redundancy, because if we form such relations from each of two interrelated role-types, we would arrive at 'mirror' expressions which reflect one and the same information. In our example, this means that, considering all role-types, we should have (among all) these two complex relations identified:

\[
\text{BANK} \quad \text{-} \quad \text{give} \quad \text{-} \quad \text{credit} \quad \text{-} \quad \text{CLIENT}
\]

\[
\text{CLIENT} \quad \text{-} \quad \text{receive} \quad \text{-} \quad \text{credit} \quad \text{-} \quad \text{BANK},
\]

reflecting one and the same thing, namely the BANK’s giving credit to CLIENT. In such cases, one of the expressions (randomly chosen) is to be removed.

In summary, we have identified role-types and inter-role relations sufficiently elaborated and adequately specified, following rigorous rules and clear steps.

Because of our focus on a target organization, we would need to position this modeling output, so that it has the perspective on the (target) organization.

For this purpose, we introduce the RR chart; 'RR' stands for 'Roles & Relations'. The chart basically reflects the mentioned output, adding more precision on our viewing the target organization (to be supported by the software system-to-be).

We construct the RR chart by taking each pair of role-types (related among each other) and putting them in boxes between which we put the corresponding description. Then names of the role-types are put in capital letters. The names of those of the role-types relating to the realization of a particular activity (for instance, the activity: 'sell financial product') are underlined. Next to that, the name of the role-type corresponding to the target organization (according to the considered case) should disappear. On its place, the particular instance name is to appear. It should
not be with capital letters because it is the name of an instance. If we consider
the bank 'Bank Abc', then instead of 'BANK' we should therefore put 'Bank Abc'. We
should also give a unique number of each row of the RR chart, which attached to the
letter 'R' (from 'relation') would form the identification string about each inter-role
relation. A partial RR chart about a case related to the bank Bank Abc and other
related role types is shown on Figure 6.1.

![RR chart example](image)

Figure 6.1: Example of a RR chart

For a detailed illustration of the building of a RR chart, readers could consider
the following chapter reflecting a real-life case study.

We propose then extending the RR chart with *Semiotic Norms* (Liu, 2000),
providing in this way a well-defined standardized specification of each of the inter-
role relations [Filipe 2002]. Organizational Semiotics has been considered in Chapter
3, including Norm Analysis, as a theoretical foundation for rigorously specifying rules,
applicable for a wide range of modeling tasks concerning both Business Process
modeling and software specification. Thus, Norm Analysis is adequate to be
considered in applying SDBC where the Business Process modeling output should be
reflected in software specification models.

So, following the OS theory (Chapter 3), we attach one Semiotic Norm per each
of the relations (this is illustrated thoroughly in the following chapter). For a brief
illustration, we will show how we define a norm concerning the relation R1 (Figure
6.1), using 'R1' also within the norm's identification string, adding 'N' before it (from
'norm'). Hence the norm NR1 would be as follows:

NR1
*Whenever* Bank Abc has offered a financial product to CLIENT
*If* CLIENT decides to purchase this product
*Then* Bank Abc
*Is obliged to* deliver the product following the offered (product) specification.

In this way, we have inter-role relations identified which are adequately
specified. Such a specification could be reflected straightforwardly in the specification
model of the software system-to-be.

And finally, we would need to position each relation in terms of the target
organization's structure. Said otherwise, we need to know which unit of the target
organization is actually concerned with a particular relation. The reason is that in
designing software for the purpose of supporting an organization, one must be
precise about the internal functioning of the organization (and the software's
operation in this context). This internal functioning includes the collaboration among
different organizational units in bringing about the overall organizational functionality.

In realizing this, we discover the major units within the target organization and
attach all relations concerning the organization to the corresponding unit. The
decision to which unit to attach a particular relation is made based on an analysis of
the case information (this might be complemented by interviews). Illustrating this,
we have presented a positioning of two relations of the above example (Figure 6.2):
Thus, we have a mechanism for identification of role-types and inter-role relations as well as guidelines for the adequate elaboration of these relations, including their:
- realistic description
- normative specification
- organizational positioning.

On the basis of this, we proceed towards deriving a constructional view over the considered business reality, bringing together the compositional and structural aspects, as it has been already mentioned at the beginning of the current section.

6.1.3 Summarizing the Achieved Output

Taking into account the intention to use the modeling outputs achieved so far for deriving Business Process patterns which are to be used for a DEMO-driven identification of Business Components (as it is according to Chapters 4 and 5), we propose the following two steps to be considered:
1. summarizing the achieved (modeling) output;
2. deriving (from it) standardized (Business Process) patterns.

The first of these two steps is to be considered within the current sub-section. The second step will be considered in Section 6.2.

Since the intention is to summarize the achieved modeling outputs, bringing together the compositional and structural aspects, we would need to focus on the different elements of the mentioned outputs, bringing them together. This would make it clear how the transition towards Business Process patterns is achieved ('Business Process patterns' are to be defined further on in this chapter).

The SCI ('SCI' coming from 'Structuring the Customer Information') chart is suggested particularly for the purpose of summarizing the (mentioned) achieved modeling output. It is proposed building a SCI chart, based on the mentioned output. The model is to be drawn as follows:

- The target organization is represented in a rectangle with rounded corners indicating that the organization is the target one.
- In the upper left corner of the rectangle is written the name of the organization.
- Within the rectangle, each unit (internal for the organization), reflected in and viewed as a sub-role, is represented in its own rectangle.
- All relevant role-types characterizing organizations from the environment of the target organization are represented in rectangles outside the rounded-cornered rectangle.
- Each of these rectangles is connected (with a line) to those of the rectangles representing target organization's sub-roles to which it has a relation; each line is numbered using Natural numbers.
• The rectangles of those sub-roles which relate to each other, are also connected; it is to be noted that the relations among target organization’s sub-roles have not been addressed in the modeling steps considered so far, the SCI chart is where this issue appears for the first time, within SDBC. The same numbering (as the one specified above) is applied for these relations as well as to relations between external role-types.
• Each entity on the chart is briefly described (with one sentence). The description is in a rectangle attached to the entity’s one.

Just a partial SCI chart for the above discussed example is presented in Figure 6.3. As seen from the figure:
• A target organization is considered whose core business is (according to the example/case information) ‘delivery of financial products’. The name of the target organization is ‘Bank Abc’.
• For the purpose of the study, several sub-roles are considered (concerning Bank Abc): Sales Department (SD), Financial Department (FD), and so on (Figure 6.2).
• The relevant role-types corresponding to the environment of Bank Abc, are CLIENT, INSURER, and so on (as it can be seen from Figure 6.2).
• Based on the relations’ analysis (that has been conducted), the identified relations are: SD – Client (1), FD – Insurer (2), and so on.
• In addition, an inter-sub-role relation has been identified:
  – SD – FD (3): Because it might be that SD need an approval from FD before delivering a financial product to a client.

As stated above, the SCI chart is incomplete. The considered example is just illustrative and its goal is to facilitate the explanation of the proposed (modeling) steps. A real-life case will be presented in Chapter 7.

![Figure 6.3: SDBC – SCI Model](image)

Hence, the SCI chart summarizes results concerning different modeling tasks. However, some other aspects (such as the normative one) are not reflected in the chart. Therefore, this is to be taken into account in the further usage of the SCI output; it is to be complemented by such issues.
In summary, SCI provides a modeling output useful for a further (DEMO-driven) Business CoMponents’ identification within SDBC. It has become clear that in building a SCI chart, a modeler follows rigorous guidelines guaranteeing preciseness, and also that the SCI output integrates compositional and structural aspects, viewing both roles and inter-role relations.

Thus, the SCI output which could be adequately elaborated and complemented in organizational and normative directions, is considered to be a sufficient basis (possessing sufficient descriptive value) for the identification of Business CoMponents, guaranteeing that nothing essential has been overlooked from the original business reality.

The following section considers particularly the identification of Business CoMponents, based on the SCI modelling output.

### 6.2 IDENTIFICATION OF BUSINESS COMPONENTS

As mentioned at the beginning of the current chapter, the guidelines addressed in this section relate to Activities 2 to 6 (Figure 4.19), following the goal of identifying Business CoMponents, using the SCI-driven modelling output considered in the previous section. According to what has been proposed in Sub-section 6.1.3, we are to reflect the mentioned output in Business Process patterns which are to realize a bridge between SCI and DEMO, in reflecting the mentioned modeling output in an adequate identification of Business CoMponents.

For this reason, in this section, we will address Business Process patterns: their definition and (SCI-driven) derivation (Sub-section 6.2.1), and then – their mapping towards (DEMO-driven) Business CoMponents (Sub-section 6.2.2).

#### 6.2.1 Business Process Patterns

After summarizing (via SCI) in a comprehensive way the modeling output achieved so far, it is now possible to consider the derivation of Business Process patterns.

A *Business Process pattern* is defined as a 5-tuple \(<I,A1,V,N,A2>\) as follows:

- \(I\) is a Natural number – the number of a relation (as according to a SCI model).
- \(A1\) is the character string characterizing one of the two roles concerning a relation, in particular – the role associated with the responsibility (concerning it).
- \(V\) is a character string representing the corresponding verb (as from a RR model (Figure 6.1)).
- \(N\) is a character string representing the corresponding noun (as from a RR model (Figure 6.1)).
- \(A2\) is the character string characterizing one of the two roles concerning a relation, in particular – the role not associated with the responsibility (concerning it).

It is suggested using 'BPP1' in identifying Business Process patterns ('BPP' comes from Business Process Pattern; \(i\) is a unique (Natural) number given to each Business Process pattern). For instance, the first of the identified Business Process patterns would be identified as BPP1.

Hence, we construct our Business Process patterns by considering: first, a particular relation (and its number) from a SCI model; and second, the relevant roles (A1 and A2) seen from the SCI model (including a consideration which of them holds the responsibility – this is seen from the RR model), and finally the verb+noun characterizing the relation (again seen from the RR model).
Returning to the considered example, identification of several Business Process patterns will be realized.

1. SD sell (banking) product Client
2. Insurer take (over) (insurance) risk FD
3. ...

Hence, we have achieved a standardized modeling output soundly rooted in the series of appropriate modeling steps (as considered within this chapter so far). The SCI-BPP output is much relevant to the DEMO-transactions-identification tasks since it gives adequate information about the considered role-types and their interrelation. Therefore, such an output is focused and easily usable for a further identification of transactions which are to adequately found the specification of (DEMO-driven) Business CoMponents.

This will be seen further in this section.

6.2.2 Mapping Towards Business CoMponents

As mentioned before, the SCI-BPP modeling output represents the starting point for the identification of Business CoMponents. Essential for this is the discovery of DEMO Transactions.

We present below our suggested mapping guidelines on how the mentioned output could facilitate the identification of DEMO Transactions. We will just explain what and why should be done, without illustrating it. Chapter 7 will illustrate this.

It has to be noted that we will discover DEMO Transactions in the context of building a DEMO Coordination Structure Model [DEMO]. It is to be noted as well that in parallel with the SDBC-driven research on DEMO Transactions identification, an alternative mechanism for such an identification has been evolving [Dietz 2004]; it would be in the scope of further research to conduct a comparative analysis of both mechanisms. Anyway, our suggested mechanism clearly positions the Transaction in terms of target, system, environment, and so on. The mentioned context includes as well an adequate consideration of the corresponding role-types which should be properly reflected in a Transaction’s specification.

Our suggested mapping guidelines are as follows:

Firstly, we position the system under study, reflecting this in building the system’s boundary with the notations of DEMO; this information, we take straightforwardly from the SCI chart.

Secondly, considering a particular Business Process pattern, we take the information of the role-types to be addressed; we position them either inside or outside the system’s boundary, guided by the SCI chart.

Thirdly, we take (from the Business Process pattern under consideration) information about the essence of the relation, for example ‘sell product’. Logically, this would be the basic (core) Transaction to be considered (we label it therefore ‘basic Transaction’)

Fourthly, considering the need for the identification of the starting Transaction, as discussed in the previous chapters, we turn our attention to the cause of the basic Transaction. We might discover, for instance, that SD could sell a financial product only after a confirmation from the accountants that this is feasible. Hence, this determines the starting Transaction (Chapter 2).

Fifthly, we analyze the basic Transaction and the starting one, aiming at discovering whether anything else happens in between. If so, the identification of other Transaction(s) and/or role-types might take place, following the principles of DEMO.
Sixthly, having the complete set of Transactions (relevant to a particular Business Process pattern) discovered, we are able to define a Business Process (the structure of interrelated Transactions executed to fulfill a starting Transaction, according to Chapter 2).

Seventhly, we reflect the Business Process in a DEMO Coordination Structure Model, reaching the structural specification of our target Business CoMponent.

The following chapter fully illustrates this evolution coming through SCI, BPP, Transactions, Business Processes, reaching finally a DEMO Coordination Structure Model.

We would need to further elaborate the identified Business CoMponent (to be viewed in four perspectives (as already stated), namely structural, dynamic, communicative and factual). We realize this as follows:

Firstly, we construct the communicative-aspect-model, by deriving a DEMO Process Step model [DEMO], based on the DEMO Coordination Structure Model (this transformation is well studied and demonstrated, thus we will not explain about it).

Secondly, we use the derived model as a source for building a dynamic model (Petri Net or Activity Diagram) – this transformation has been studied [Shishkov et al. 2003-2] and is demonstrated in Section 6.4.

Thirdly, we derive an ORM data model (demonstrated in Chapter 7), using the well-studied DEMO-ORM integrated application [Dietz and Halpin 2004].

Therefore, the SDBC Business CoMponent identification is to progress through a number of steps, as illustrated in Figure 6.4 which offers also information on relevant modeling tools for each of the steps (at the left side of the figure).

As seen from the figure, there is a strictly specified set of steps and corresponding tools. As explained in the paragraphs above, the transformations are straightforward and clearly defined. This will be illustrated in the following chapter which is going to demonstrate the SDBC approach and its application.

![Figure 6.4: SDBC – from SCI to Business CoMponents](image)

As seen from the figure, a Business Process pattern (BPP), derived straightforwardly from a SCI model, facilitates the discovery of DEMO Transactions to be reflected in a Business CoMponent, particularly a structural aspect view over it, modeled using the DEMO Coordination Structure Diagram. The communication
aspect view is achieved further by a mapping towards the DEMO Process Step Diagram to be reflected then in a Petri Net model providing the dynamic aspect view over the Business CoMponent. Finally, the data aspect view is achieved using ORM.

Summarizing: in SDBC we use the SCI-BPP modeling output as a foundation for the identification of Business CoMponents to be adequately elaborated in terms of structural, dynamic, communicative, and factual issues.

The following section is going to consider the mapping of such a Business Process modeling output towards (UML-driven) software specification.

6.3 DERIVING A SOFTWARE SPECIFICATION MODEL

In this section, we will consider the reflection of (identified) Business CoMponents in a software specification model. This concerns (as already mentioned) Activities 7 and 8 (Figure 4.19). The model is to be then adequately specified and elaborated.

Taking into account the goal (as motivated in the previous chapters) of arriving at a UML-driven software design model and taking also into account that (as already explained in Chapter 3) use cases are modeling constructs which serve to link the application domain (the business world) to the software domain, in a UML-driven Software Design, we consider it necessary to firstly reflect a Business CoMponent in a use case model and then adequately conduct software specification (and also elaboration) on the basis of such a model. Thus, we will address in this section firstly the use case derivation (based on a Business CoMponent) and secondly, the further specification and elaboration (as mentioned above).

6.3.1 Deriving Use Cases from Business CoMponents

In this sub-section, we will consider the reflection of a (DEMO-driven) Business CoMponent in a use case model. Hence, a crucial question in this regard is whether it is possible achieving an adequate DEMO – use case mapping (if not, some other modeling facility should be introduced).

What has been found out by Shishkov and Dietz (2004-3, 2003-1, 2003-2) is that DEMO possesses potentials adequate to its proper mapping towards use cases. Further on in this sub-section, we will (firstly) motivate the mentioned suitability of DEMO and (secondly) introduce (considering the mentioned motivation) particular set of guidelines for deriving a use case model based on a DEMO-driven Business CoMponent.

Motivation

We will motivate below the adequacy and suitability of DEMO as a use case derivation source.

Studies directed towards derivation of use cases from Business Processes have been reported in [Shishkov and Dietz, 2003-1; Shishkov et al. 2003-3; Shishkov and Barjis, 2002], and also in [Shishkov and Dietz 2004-3] where the use case derivation has been approached through three fundamental business study perspectives, namely: Language/Action Perspective and the DEMO theory in particular, Organizational Semiotics and Norm Analysis in particular, and Workflow Modeling perspective and Petri Net in particular. Relevant information has been reflected also in Chapter 3 of the current thesis.
Some strengths of DEMO have been identified (based on the mentioned studies), which are among the reasons for choosing DEMO as the SDBC Business Process modeling environment. Based on this (among other things), the SDBC use case derivation procedure has been suggested (it will be further addressed within the current sub-section). As it will be seen, DEMO plays an essential role in the procedure which could also be enriched with issues related to Norm Analysis and other Business Process modeling tools. This has been motivated by the purpose of achieving synergy in deriving use cases from Business Processes - relevant strengths have been considered and combined from different Business Process modeling tools.

As it has been studied by Shishkov and Dietz (2004-3), a use case model can be consistently derived based on a DEMO Business Process model since, applying DEMO allows for producing a sound Business Process model for the Software Design, a clear model that captures the features which remain unchanged from realization. Such a model could be a proper basis for improving the delimitation, identification, and the specification of the modeled software system [Dietz 1994]. Hence, DEMO possesses (and this has also been studied in Chapter 3) completeness and capability of capturing the essence of Business Processes. Therefore, if the specified software model stems from a DEMO Business Process model, the software designer would have the right (re)design freedom [Dietz 1999]. All this makes a DEMO model being a sound basis for further software specification activities. For this reason, it is considered worthwhile exploring the use case derivation centered around DEMO business process modeling. In this regard, it has been studied that DEMO Transactions are straightforwardly relatable to the essential pieces of functionality concerning the software system-to-be. Thus, deriving use cases based on DEMO is well founded theoretically. Next to that, the actors associated with the identified use cases would be a reflection of the DEMO actor roles because they concern the same Transactions. All this makes the reflection of a DEMO Business Process model in a use case model complete, consistent, and well founded theoretically.

In support of claiming all this, we have studied a case example (the HRB Case) which is presented below. It is to illustrate a use case derivation supported by DEMO. A system, representing a Hotel Reservation Broker (HRB) is modeled for the case's purpose.

HRB matches the data about clients' required accommodation and hotel offers. Both the hotels and clients need to register in order to use the service for a selected period of time. The subscription fees for hotels are fixed depending on the chosen period and the hotel size; the fees are fixed for clients also, depending on the chosen period. Besides these subscription fees, both clients and hotels pay fixed fees when a match-making is realized. Further on, we refer to these fees as: a "reservation fee" (paid by a client) and a "hotel fee" (paid by a hotel). HRB accepts accommodation requirements from clients (e.g. check in/out dates, place, type of accommodation, price, etc.) and accommodation information from hotels (e.g. number and type of rooms/beds available, etc.). Once HRB has received requirements from a client, if requested, it performs match-making on a real time basis. HRB provides the client with a list of available accommodation (all of them meeting the client's requirements) to select from. Once the client has accepted one of the offers, he pays the reservation fee. He has to pay also the cost of the selected accommodation. Then, HRB is obliged to guarantee the accommodation. HRB should contact the selected hotel and realize the actual booking of a particular room/bed. Then, the hotel must pay to HRB the hotel fee, and will be paid (by HRB) the cost of the reserved accommodation. Once this is done, the reservation is actually completed. The service is considered finished.

110
On the basis of the above presented case information, we will proceed towards a use case derivation facilitated by DEMO. We will skip all (previously considered) SDBC modeling steps concerning the derivation of DEMO-driven Business Componenets; we directly jump to identified DEMO Transaction types (listed in Table 6.1, together with their corresponding resulting fact types).

As seen from the table, the focus is only on Transactions on the essential level, in order to keep the business model abstract enough so that it remains unchanged during (eventual) re-design of its realization. This is actually in concert with the principles of DEMO [Dietz 1999].

Based on the Transactions and result facts, the system(s) to be investigated should be selected, relevant DEMO actor(s) – identified, and their positioning (as customer and producer) – determined. Once this is done, all interaction relationships are determined. All this is depicted in Figure 6.5, representing the Coordination Structure Model – CSM (it is incomplete, since the goal is just illustrative).

<table>
<thead>
<tr>
<th>transaction type</th>
<th>result fact type</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 match-making</td>
<td>F1 match &lt;M&gt; is made</td>
</tr>
<tr>
<td>T2 subscription</td>
<td>F2 subscription &lt;S&gt; is arranged</td>
</tr>
<tr>
<td>T3 subscr. payment</td>
<td>F3 the fee for period &lt;P&gt; by &lt;CI,Hotel&gt; is paid</td>
</tr>
<tr>
<td>T4 reserv. payment</td>
<td>F4 the fee for reserv. &lt;R&gt; by &lt;CI,Hotel&gt; is paid</td>
</tr>
<tr>
<td>T5 accom. payment</td>
<td>F5 the cost for accom. &lt;A&gt; by &lt;O&gt; is paid</td>
</tr>
<tr>
<td>T6 accom. compens.</td>
<td>F6 &lt;Hotel&gt; is compensated for ac. &lt;A&gt;</td>
</tr>
<tr>
<td>T7 refund</td>
<td>F7 refund &lt;RE&gt; for violation &lt;V&gt; is arranged</td>
</tr>
<tr>
<td>T8 reservation</td>
<td>F8 reservation &lt;R&gt; by &lt;Hotel&gt; is arranged</td>
</tr>
</tbody>
</table>

Table 6.1: HRB - business Transactions list

The system under study (HRB) is considered as well as the Client and Hotel (as actor roles). Regarding the system under study, it is represented on the figure in more detail: (actor) roles A3 and A4 (they are represented in white boxes because they are elementary actors – involved in just one transaction each) are depicted as well as A5 and A6, whereas Client and Hotel are outside the system. C3 represents a so-called conversation for initiation. It models the periodic activation of A5 to issue payment requests. The external bank EB1 contains client/hotel data; A3 is allowed to inspect the contents of EB1. In other words, A3 is allowed to know the information provided by hotels. The reason for this allowance is that A3 needs to know the provided information. How A3 gets access and also how hotels add/remove data is not shown. These matters are considered to belong to the information and documental perspective and thus are not represented in the CSM.

So, based on the above Business Process modeling output, we proceed towards use case derivation. It is done intuitively (in a sense that no established and proven guidelines were followed – such guidelines actually do not exist; the guidelines (to be presented further in this section) are to be the SDBC contribution to the problem of use case derivation), as in all preliminary materials considered [Shishkov and Dietz, 2003-1; Shishkov et al. 2003-3; Shishkov and Barjis, 2002].

As mentioned already, DEMO Transactions are straightforwardly relatable to the essential pieces of functionality concerning the software system-to-be and thus allow us to derive just those of the use cases (concerning the system) which reflect essential behavior. The use cases concerning ‘informational’ issues (e.g. informational update) are hence not straightforwardly identifiable using DEMO [Shishkov and Dietz, 2003-1]. We will use the terms essential use cases and informational use cases, to distinguish between these two major types of use cases [Shishkov and Dietz 2004-3].
So, the DEMO Transactions are straightforwardly mappable into essential use cases – the derived use case model is depicted on Figure 6.6. The use cases are numbered in order to trace directly from which DEMO transactions they are derived. This is illustrated below:

<table>
<thead>
<tr>
<th>Source Transaction(s)</th>
<th>Derived use case(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0; 1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4.1; 4.2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

As seen from Figure 6.6 and also from the matching between source Transactions and derived use cases, it is possible that more than one use cases are derived from just one DEMO Transaction. Transaction 4, for example, has the same essence no matter who the role-type is. As for the use case model, it should clearly distinguish the cases in which CLIENT pays reservation fee from the cases in which the HOTEL does this. The reason is that the realization issues are different in these
two situations. Thus, two use cases are needed. As for Transaction 1, it is reflected in
the use case 'Perform Match-making', but in order for this functionality to be realized,
it is necessary the data accuracy to be checked first. However, this is not an essential
business Transaction, it is just information checking. For this reason, the use case
'Check Data Accuracy', being related to the use case 'Perform Match-making', is also
derived from Transaction 1.

As seen from the use case diagram (Figure 6.6), the numbered use cases are
essential use cases. The other use cases are informational use cases – such are the
use cases 'Check Data Accuracy' and 'Add data in DBC' (the abbreviation 'DBC' will be
explained below). As studied in [Shishkov and Dietz 2004-3], the informational use
cases are derived based on an analysis gravitating around the DEMO and also
Ontology models [Liu 2000]. As for the actors, it is clearly seen that reflecting the
DEMO actor roles into use case actors is adequate and also puts the identification of
use case actors on sound business modeling background.

\[ \text{Figure 6.6: Use case diagram of HRB} \]

The diagram on Figure 6.6 shows use cases and actors typical for such a HRB.
Since the purpose is just illustrative, only some of the use cases and actors typical for
such a system are considered. Regarding the diagram, the abbreviation 'DB' stands
for the database used by HRB. For convenience, DB is virtually divided into DBC and
DBH (containing data of offered and searched accommodation, respectively). The
diagram shows two actors: Client and Hotel. Concerning Client (Hotel) – he takes the
decision, has the responsibility, has the goal to add request (offer) in DBC (DBH),
and/or remove it from DBC (DBH), and have his data matched up with relevant data
from DBC (DBC). There are sixteen use cases: ‘Add Data in DBC’, ‘Check User’s Inf.’,
and so on. There are three <<include>> relationships (‘Perform Match-making’
requires ‘Check Data Accuracy’; ‘Add Data in DBC’ and ‘Add Data in DBH’ require
‘Check User’s Inf.’) and two <<extends>> relationships (in some cases, before
adding their data to DBC/DBH, the system might request from Client/Hotel additional
information, so the basic use cases are ‘Add Data in DBC’ and ‘Add Data in DBH’, and
they are extended with ‘Request Additional Inf.’).

Hence, on the basis of the conducted study (considered in the paragraphs
above) as well as of the comparative analysis concerning use case derivation based
on different Business Process modeling tools [Shishkov and Dietz 2004-3], and also
based on the example which has been presented in this section, strengths of DEMO
have been identified, regarding the derivation of use cases from Business Process.
These strengths will be briefly outlined below.

A definite (related) strength of DEMO is its allowing for the creation of models
which are essential, fully abstracted from all realization issues. Next to that, DEMO
can provide multi aspect (and further extensible) modeling output (as already
discussed). Hence, using DEMO (plus other relevant tools) one could build a sound
and complete (essential) Business Process model. Therefore, by basing the use case
derivation on such a model, designers could be sure that the software specification
activities are put on sound theoretical foundation.

Another (relevant) strength of DEMO is the possibility (as already illustrated) to
straightforwardly derive not only use cases from DEMO Transactions (in particular
those use cases reflecting essential behavior – the essential use cases) but also
actors (from the DEMO actor-roles). As for the rest of the use cases, DEMO indirectly
facilitates also their discovery (as it has been shown through the example).

Taking into account the significant importance of the used graphical notations
for the ease and understandability of the derivation process, it should be stated that
(as shown from the example) the graphical notations of DEMO (in particular, the
DEMO Coordination Structure Model) are suitable with respect to the use case
derivation. The DEMO actor roles and Transactions are depicted graphically in a very
convenient way for their reflection in actors and use cases, in a use case model. All
this adds value to the derivation mechanism, making it easily understandable and
illustratable.

Thus, use case derivation on the basis of DEMO is well founded theoretically and
easily realizable mainly because (as stated above) DEMO Transactions and actor roles
can be used as a consistent basis for deriving a use case model: in particular, use
cases and actors, respectively.

We have brought sufficient evidence in support of the claim that the DEMO – use
case reflection is adequate and useful. Next to that, we have achieved some practical
knowledge on how such a reflection could be realized (based on an analysis
[Shishkov and Dietz 2004-3] as well as on a study of the case example, presented).
All this is a foundation for our proposing use case derivation guidelines within SDBC.
This issue will be addressed below.

114
Use case derivation guidelines

As mentioned before, we consider the above discussed (and presented) studies, in proposing the SDBC use case derivation guidelines. We consider also a fundamental software design principle [Jacobson et al. 1995], namely that in building a software system, one should be concerned with issues related to not only the Business System (to be supported by the software system-to-be) but also to the particular requirements (defined by the future user(s) of this system) towards the software [Wieringa 1995]. We consider as well a fundamental DEMO principle, namely the principle of clear separation between essential business issues (characterizing a business reality) and realization issues (concerning the particular way in which a software system, for example, realizes some functionality within the context of the mentioned business reality). This principle has been discussed both in Chapter 3 and above in the current chapter.

Therefore, we propose the following way of distinguishing among the functionality pieces concerning a software system, introducing three groups of functionality pieces:

* essential functionality pieces: reflecting essential functionality patterns from the (considered) business reality;
* Informational functionality pieces: reflecting informational aspects concerning a possible realization of a particular action(s);
* user-defined functionality pieces: reflecting the requirements which the future system users have requested.

Hence, some overlaps are possible among the above three groups, especially between the information and user-defined functionality pieces. However, this does not lessen the usefulness of grouping software functionality pieces in this way because the informational functionality pieces would appear as a natural extension to the essential ones (being reflected towards realization), while the user-defined functionality pieces are not necessarily straightforwardly related to the original business reality, they might appear just as a requirement from the user(s) [Loucopoulos and Karakostas 1995].

Based on the studies reflected in the current chapter (so far), we claim that DEMO transactions are an adequate source for identifying the essential pieces of functionality, characterizing the software system-to-be. As for the informational functionality pieces, they should be identifiable from the dynamic reflection of a Business CoMponent (carried out in SDBC through Petri Net (Figure 6.4)) because in realizing such a dynamic reflection (based on the DEMO Process Step Model, in the SDBC application) it is necessary to specify completely the events flows to take place, and this for sure includes consideration of informational issues (not just essential). As for the user-defined functionality pieces, we would not propose (within SDBC) particular rules for their discovery, leaving this dependent on the particular software design decisions, positioning this issues as belonging to the Requirements Engineering domain where one can find promising relevant results [Wieringa 1995; Loucopoulos and Karakostas 1995].

On the basis of the above considerations and arguments (and using the abbreviation 'UDR' standing for 'User-Defined Requirements'), and preserving the 'initial' terms, namely essential/informational use cases, we define three types of use cases:

1. Essential use cases (they are about essential functionality pieces)
2. Informational use cases (they are about informational functionality pieces)
3. UDR use cases (they are about user-defined functionality pieces).
Hence, we should derive the essential use cases on the basis of DEMO transactions, the informational use cases—using the (Petri Net) dynamic aspect model of a Business CoMponent, and the UDR use cases—following particular design decisions and Requirements Engineering practices [Wieringa 1995]. This is depicted in Figure 6.7.

![Diagram of use case derivation guidelines](image)

**Figure 6.7: The SDBC Use case derivation guidelines**

As seen from the figure, in order to properly specify the UML use case model, it is necessary to place it adequately on a corresponding Business CoMponent, and also to soundly consider the user-defined requirements. In particular, the *essential use cases* are to be derived from corresponding DEMO Transactions (considering a structural view over a Business CoMponent) while the *informational use cases*—from corresponding Petri Net patterns (considering a dynamic view over a Business CoMponent).

The introduced use case derivation guidelines have been theoretically justified through the studies reported in the current chapter, through an analysis [Shishkov and Dietz 2004-3] as well as through the content of Chapter 3 (considering use cases and other relevant modeling tools) and Chapter 5 (analyzing tools and their suitability). The guidelines will be illustrated in the following chapter.

We now continue with questions related to the specification and elaboration of a (derived) use case model.

### 6.3.2 Further Specification and Elaboration

In this sub-section, we will consider, as mentioned above, the specification and elaboration regarding a derived use case model. By *specification* we mean extending the descriptive information concerning a particular use case. By *elaboration* we mean extending the UML Use case model with other (relevant) UML models including the UML Activity diagram and the UML Class diagram. The specification and elaboration issues will be subsequently addressed further on in this sub-section.

**Specification**

Once a UML use case model has been derived on the basis of Business Process models, a fundamental business-software alignment has been achieved, providing elicitation about the reflection of the major business and requirements—related issues in the model of the software system-to-be. However, the format of the use
case model allows for just a general elicitation. It alone would be too vague as an input for the further software design steps. This motivates the need to add further elicitation to at least the most important use cases from the use case model. Thus, a use case specification is necessary in order to adequately proceed with the further Software Design steps.

Being inspired by the (relevant) original ideas of Cockburn (2000), Shishkov and Dietz have analysed the use case specification potentials (2001) and proposed a use case specification template (2002-4; 2003-3), which overlaps with some recent research activities [Insfran et al. 2002].

Thus, we propose, based on all this, a use case specification template (to be applied to those use cases from a use case model, which have major significance. In the template, it is included:

1. The name of the use case; Source: the use case model.
2. Information about the primary actor; Source: the use case model.
3. Information about the other stakeholders involved, which is necessary because the software system-to-be would in any case need to function in accordance with the interests of not only the primary actor but also the other stakeholders involved; Source: a Business CoMponent, in particular, the (dynamic) Petri Net business process model (because it should contain the information of all relevant events taking place and therefore, of the involved stakeholders).
4. Information about the goal associated with the use case, mainly because such information would be necessary in precision the software functionality (including some deliverables); Source: a Business CoMponent, in particular, the (structural) DEMO business process model (the Coordination Structure Diagram) which reveals information of the transactions (including the basic Transaction), from them one could derive the goal.
5. Information about the precondition(s), trigger(s), and so on, which is necessary concerning the software functionality; Source: additional study and analysis of the case information.
6. Information whether the use case is a 'long running' one or restricted to the performance of a particular activity; Source: DEMO Coordination Structure Diagram.
7. Information of the events flow concerning the use case, revealed through the (widely used) format 'main success scenario + extensions'; Source: the (above mentioned) Petri Net model (however, the information from the Petri Net model would need to be further detailed).

Thus proposed, the SDBC use case specification template is well-founded in previous studies. Regarding its illustration, Chapter 7 will provide an exhaustive relevant illustrative information.

Elaboration

Once a UML use case model has been derived and (adequately) specified, it is necessary (as mentioned before) to complement the model with other UML models as it is according to the RUP-UML perspective [OMG 2000; Kruchten 2000] which is followed as a 'de facto' standard (according to what has been discussed already).

We propose in SDBC following the mentioned perspective in elaborating the (derived) UML Use case model in terms of structure and dynamics. The other two perspectives considered within our business modeling activities, namely the communicative and data perspectives, are also to be taken into account. As for the first of them, it is not going to be directly reflected in the software design model
since it has properly been reflected in the derivation of use cases, which affects the software model. As for the data perspective, we propose keeping in use ORM, making accordingly the 'business-software updates' which are necessary.

As for the structural and dynamic perspectives, they are to be considered as follows:

- Deriving a UML Class diagram should provide the structural view concerning the software system-to-be.
- Deriving a UML Activity diagram should provide the dynamic view concerning the system.

The RUP-UML 'environment' provides well-established mechanisms for building Class diagram and Activity diagram base on a UML Use case model. For this reason, we are not going to discuss the transformations from use cases to such (UML) diagrams.

The UML State diagram could also complement an Activity diagram and is also derivable in a well-studied way from a UML Use case model.

In Chapter 3, we have briefly introduced and discussed all above mentioned UML diagrams.

In Chapter 7, we will illustrate the derivation of Class/Activity diagrams on the basis of a UML use case model.

6.4 BUSINESS/SOFTWARE MODELS' VALIDATION

The identified Business/Software CoMponents have to be appropriately validated, as a guarantee that the (delivered) modeling output is adequate. What is usually validated, is the models' structure and dynamics, as it is well known [Barjis et al. 2002].

As for the structural aspect, we claim that the modeling steps followed do possess also valudational value. The rigorous rules in deriving DEMO Transactions from a SCI chart and the DEMO modeling itself represent a reliable modeling mechanism which is theoretically founded (the principles of Component-Based Development and the Language-Action Perspective are followed). Next to that, the derivation of UML models (based on DEMO) is being done following rigorous rules and relying on the proven values of UML. We consider all this as having a validation value that is adequate to our addressing the structural validation aspect.

Thus, further on in this section, we will consider the dynamic validation aspect. Tracing adequately the dynamic issues in this concern is of importance.

Once there is an identified Business/Software CoMponent, it is of significant importance to trace precisely all essential dynamic issues related to the coMponent model, in order to make sure that the performance of the software system-to-be will be properly designed. Otherwise, there is a risk the software system to function inadequately.

A way to reliably and illustratably trace the interactions within a Business/Software CoMponent is via discrete event simulation [Shishkov et al. 2003-2]. As discussed in previous chapters, analyzing systems through simulation has proved its methodological advantages [Law and Kelton 2000]. Next to that, the possibility to represent and visualize the system under study in sound graphical notations is of particular importance since modelers would have the chance to grasp the peculiarities of the system as a whole. Another advantageous feature regarding simulation models is that they are easily understandable not only for the system
modelers but also for potential users [Barjis and Shishkov 2001-1]. For all these reasons, we claim that it is feasible to expect that simulation models can be helpful for grasping the dynamics of a component being modeled.

We consider in particular the ARENA simulation tool [ARENA] as one of the appropriate tools that can be used for the goal which has been stated above; ARENA has already been introduced (and also analysed in Chapter 3), and as concluded: an advantageous feature of ARENA is its animation facilities which are rich and well formulated enough to make simulation models interesting, attractive and easy for training people as well demonstration purposes [Barjis and Ilkov 2000].

The usage of the ARENA discrete event simulation tool for validating Business/Software Components in terms of dynamics has been studied by Barjis, Shishkov, and Dietz (2002). Based on these results as well as on a relevant methodology, proposed by Shishkov and Dietz (2002-3), Shishkov, Barjis, and Dietz have introduced a methodology for validating the dynamics of identified components, using ARENA [Shishkov et al. 2003-2], based on which a set of steps is proposed, through which the mentioned (ARENA-based) validation could be realized (it must be noted that the pre-simulation issues are also to be considered, being an essential part of the simulation task).

The mentioned set of steps is as follows:

- **Describing the considered business/software reality and adequately reflecting it into a structural model (concerning the target system and its interaction with its environment), as done according to the SDBC application guidelines.**

Such a Business/Software Component is to be viewed as a self-contained part of a larger system. Applying DEMO/UML, respectively, is considered to be most appropriate, mainly because of the capabilities of DEMO/UML to grasp the essential business/software issues.

- **Reflecting the (built) structural model into an appropriate process model that elaborates on the dynamics considered.**

This is necessary as a consistent basis for building an appropriate pre-simulation model. As far as Business Components are considered, for instance, the DEMO Process Model [DEMO] is claimed to be proper for this purpose not only because it is straightforwardly derivable from the DEMO Coordination Structure Model (which is applied within the SDBC application, as it has already been shown) but also because of its capability to consistently elaborate on the transactions considered.

- **Building the pre-simulation model.**

Building an appropriate model, straightforwardly mappable into a simulation model is crucial for the soundness of the simulation model; the UML Activity diagram can be properly used for this purpose, particularly as far as ARENA discrete event simulation is concerned [Barjis and Shishkov 2001-1].

- **Deriving the simulation model.**

This is the last of the proposed steps, delivering the actual validation output.

We will illustrate the suggested steps through a case example which has been studied in [Shishkov et al. 2003-2], namely the *Ticket machine case.*

A situation is reflected within the case, in which a person intending to travel from one place to another, uses a ticket machine; he could buy either a train ticket or a ticket for the public transport (by default – train ticket).

Buying a train ticket, one must select the desired destination first. Then there follow the selections of the ticket type (one-way ticket or round ticket) and the price type (normal price or a price with reduction). These could be done in any order – the person could select first the ticket type and then the price type or first the price type
and then the ticket type. Finally, (s)he should pay (inserting coins/banknotes or paying by a card). Then the machine issues a train ticket.

![Diagram of Ticket Machine](image)

*Figure 6.8: Ticket Machine – partial DEMO Coordination Structure Diagram*

Buying a ticket for the public transport, one must select the ticket type first (ticket for one trip, ticket for two trips in one hour, ticket for one day, ticket for one week, and so on). Then he should pay (inserting coins/banknotes or paying by a card); and the machine issues a ticket for the public transport.

An identified (DEMO-based) Business Component reflecting the case situation (described above) is depicted on Figure 6.8 (we have depicted a partial DEMO model, just for illustrative purpose):

The corresponding transaction/fact types are as follows:

<table>
<thead>
<tr>
<th>Transaction type</th>
<th>resulting fact type</th>
</tr>
</thead>
<tbody>
<tr>
<td>T01 ticket issuing</td>
<td>F01 ticket has been issued</td>
</tr>
<tr>
<td>T02 selection</td>
<td>F02 selection has been made</td>
</tr>
<tr>
<td>T03 payment</td>
<td>F03 the ticket price has been paid</td>
</tr>
</tbody>
</table>

Concerning the model, and in particular the identification of the essential Transaction types – three Transactions are identified, as above listed, together with corresponding resulting fact types.

The diagram on Figure 6.8 shows the actor-roles (represented by boxes: grey box marked by S02 – external actor; white box marked A01 – internal elementary actor), transaction types (diamonds in disks), and the relationships between them (the small black boxes denoting which actor is-role is the executor of a transaction type). The grey-lined boxes represent the boundary of the considered system (Ticket machine in this case). Outside the boundary is actor S02 as well as an external data bank DB01 containing in this case destination data needed by actor A01, to whom is the dashed line. The DEMO notations have already been considered in Chapter 3.
Based on the DEMO Coordination Structure model, it is straightforward to further elaborate on the transaction issues, as explained already. This means building a DEMO Process Model [Dietz 1999]; DEMO has been already introduced in Chapter 3. Here we are not going to deepen in explanations in this direction since these issues are thoroughly covered by the DEMO methodology [DEMO]. The built DEMO Process Model is depicted in Figure 6.9.

The DEMO Process Model represents a sound basis for the construction of Activity diagram, since the model reflects the dynamics concerning the essential business issues. We further elaborate on the dynamics of the business issues, by applying Activity diagram. Moreover, as already stated, it can be used as a basis for consistently deriving an ARENA simulation model. Activity diagram, corresponding to the ticket machine case is represented in Figure 6.10.

![Ticket Machine - partial DEMO Process Model](image)

**Figure 6.9: Ticket Machine – partial DEMO Process Model**

It is seen from the figure that Intend to Travel is the first activity, a starting bundle for the ones to follow. There are two triggers coming out of Intend to Travel.

The first trigger leads to another activity (Select Destination), related to the person’s preferring train transport to public transport, which is followed by a synchronization bar. There are two outgoing triggers attached to the bar – they go to the following activities: Select Ticket (one-way ticket or round ticket) and Select Price (normal price or price with reduction). It is worth noting that these two activities correspond to parallel processes – their order is irrelevant. When dealing with parallel behavior, it is necessary to synchronize. For this reason the triggers from these activities go together to a synchronization bar. Then the trigger coming out of the bar goes to the activity Payment, after which Machine issues a train ticket. And finally, the activity Travel is realized. From there the trigger goes to the end point.
The second trigger leads to a decision activity. This is because the first decision of the person should be about train travelling – this decision determines the triggers, coming out of Intend to travel; if the destination is not reachable by train, the person goes to a second decision, related to public transport (this second decision is marked with the Decision diamond). If the destination is not reachable by public transport either, a trigger coming out of the decision activity, goes directly to the end point, indicating that the desired consequences had not taken effect. Otherwise, three activities are realized: Select type of ticket (the person should choose the type of public transport ticket to be used), Payment, Machine issues a ticket for the public transport, before the activity Travel takes effect. From there the trigger goes to the end point. The diagram represented in Figure 6.10 is helpful also to identify processes, decision points, and synchronization points as studied in [Shishkov and Dietz 2001; Shishkov et al. 2003-2].

Further on, the Activity diagram is used as a basis for the development of an ARENA simulation model (the mapping has been in details studied and explained by Barjis and Shishkov (2001-1)). It is presented on Figure 6.11. The (ARENA) model is useful for describing and visualizing processes as well as for identification of processes, decision/synchronization points – all this directed towards the (dynamic) validation task. In this particular example, there are: one ‘choose’ element, one ‘branch’ element, one ‘batch’ (synchronization) element and processes as ‘Select destination’, ‘Payment’ and so on.

As it is known from previous studies [Shishkov et al. 2003-2; Barjis and Shishkov 2001-1] such a simulation model can be a helpful mean for analysing a Business/Software CoMponent, adequately validating it.
Summarizing the content of this section: we have proposed and illustrated a set of steps directed towards adequately validating a component in terms of dynamics, by simulating the component's dynamics. As shown already, the suggested steps (presented also at the 2003 Edition of the Summer Computer Simulation Conference) allow for the development of a sound simulation model consistently rooted in prior modeling activities. It has been also suggested what particular modeling tools can be used to realize the introduced steps.

We have demonstrated the dynamics validation of a Business Component. If a SDBC Software Component is to be validated, the task is even simpler because we could directly use the UML (Activity Diagram) model of the component, reflecting it in ARENA. Figure 6.12 summarizes the validation scheme followed:

```
Business Component
  ↓
DEMO
  ↓
Software Component
  ↓
Activity diagram
  ↓
ARENA
```

**Figure 6.12: The dynamics validation of components in SDBC**

As seen from the figure, we consider in SDBC a dynamic validation concerning both Business Components and Software Components. Regarding the structural validation, as already stated, we consider the modeling tools (through which SDBC is applied) as having reliable validation facilities in themselves.
6.5 CONCLUDING REMARKS

We did show in the current chapter how the SDBC approach, whose conceptual framework has been introduced in Chapter 4, could be practically applied, following four phases, namely: Information structuring, Identification of Business Process models (Business CoMponents), Derivation of software specification models (further reflected in Software CoMponents), and Validation of the business/software models. Based on a proposition about suitable modeling tools for the application of SDBC (Chapter 5), the current chapter has elaborated each of the proposed steps with information on how a particular step could be realized through a particular (modeling) tool. All this is outlined in Figure 6.13:

![Diagram](image)

Figure 6.13: Applying SDBC (steps and tools) - outline

Chapter 7 will further elaborate on the SDBC application issues, by conducting a real-life case study. There, all steps considered within the current chapter, are going do be adequately illustrated.
Chapter 7

CASE STUDY

For the things we have to learn
before we can do them,
we learn by doing them.
Aristotle

On the basis of the proposed research framework (Chapter 1), the introduced essential concepts (Chapter 2) as well as the analysis of existing tools and their limitations (Chapter 3), chapters four to six have introduced the SDBC approach, including: its conceptual framework (Chapter 4), a selection of modeling tools appropriate to making the approach operational (Chapter 5) as well as step-by-step guidelines for its application (Chapter 6).

In this chapter, the application of SDBC will be demonstrated by means of a test case study carried out at a large insurance company in The Netherlands. Since it prefers not to be mentioned by name, the company is referred to as 'Icomp' (a name given by us, standing for Insurance company). The goal of the case study is not only to provide practical evidence about the strengths of SDBC but also to validate the essential ideas and concepts suggested within the current research.

This chapter is structured as follows: Section 7.1 presents the case study background, bringing elicitation on the case's focus, problem, and goals as well as on the selection of the target organization (where the case study has been carried out). Section 7.2 outlines the collected information, to be used as an input for the application of SDBC. This information has been kindly delivered to us by representatives of the target organization. The particular application of the SDBC approach is reflected in Section 7.3. Section 7.4 contains concluding remarks.

7.1 BACKGROUND

The preparation of the case study has been driven by its goal and also by the consideration of several other relevant aspects. Among them are the selection of a target organization, the case study focus, and the problem definition. These aspects will be briefly discussed below:
Goal of the case study
From the perspective of the needs of the current research, the main goal of the case study has been defined as follows: validation of the conceptual framework of the SDBC approach and also of its application guidelines, by applying them in a real-life situation. Taking into consideration the essential elements presented as part of these guidelines, the case study should focus on the following aspects:

- **Business Processes analysis and modeling**, comprising the consideration of the initial (case) information, the structuring of its elements, the identification of relevant Business Process modeling units and their adequate reflection in Business CoMponents (soundly elicited in terms of structure, dynamics, data, and communicative issues).

- **Derivation of a software specification model**, comprising the reflection of a Business CoMponent(s) into a corresponding software specification model, to be further decomposed into Software CoMponents.

Selection of an organization to be explored
In order to (adequately) validate SDBC, a suitable organization had to be found, an organization—willing to participate in the case study, by providing relevant information. The choice of organization has been based on the following criteria:

- **Size of the organization.** The size of an (explored) organization has an influence on the complexity of the Business Processes within the organization. This concerns the sophistication of the support provided by corresponding information systems to these Business Processes. For instance, small organizations working in an ad-hoc manner are rarely facilitated by sophisticated Information Systems and technologies. Thus, focusing (in this research) on the support to Business Processes, provided by ICT applications which are comparatively (more than average) complex, we have had the requirement for a large organization (consisting of more than 2000 employees).

- **The business domain.** As it is well-known, organizations belonging to some business domains are more dependable on a proper application support than organizations belonging to other business domains. We have targeted business domains related to the financial sector because financial companies are currently among those which greatly depend on Information Systems and technologies.

  Hence, as stated before, the case study considered in the current chapter, has been carried out in a financial (in particular insurance) company, namely the company Icomp, situated in The Netherlands. Icomp delivers financial products (and financial services) to end customers.

Focus of the case study
Considering the company Icomp, the case study has focused in particular on a part of the company’s business, namely: the distribution of financial products. This choice has been made not only because such a focus has a direct relation to the core of the business of Icomp (this will be seen from the information provided in the following section) but also because the mentioned part of the business of the company strongly requires appropriate Business Process modeling and is dependent on support by Information Systems and technology. Therefore, relevant improvements in these directions could be much useful for Icomp.
Problem definition
Considering the available actual information about Icomp (this information is reflected in the following section), we have defined two problems to be addressed in the current case study:

- The environment of Icomp demands a sounder and more flexible way in which the company specifies and modifies its financial-products-related Business Processes, grasping adequately all essential aspects.
- A better clarity would be appreciated about the impact of eventual reorganization within the company's financial-products-related business activities. This is driven by the necessity of introducing relevant technology in support of the mentioned activities.

Goals in context
Considering the main goal of this case study, which has already been formulated, and in relation to the defined problems (addressed above), we have made the following elaboration of the general case study goals, in the light of particular benefits that the case study could bring to Icomp:

- Provide insight into the way in which the financial-products-related Business Processes of the company could be modeled so that there is a possibility for flexible modifiability (facilitated by re-use options, for example), soundness, and completeness (regarding the essential aspects of a business reality).
  - This might include the modeling of the essential issues characterizing the company and its environment.
- Provide insight into the way in which a software specification model could be (soundly) derived on the basis of a Business Process model.
  - This might include a proposition concerning the introduction of an ICT application and a demonstration how its specification could be realized.

The following section will provide information on Icomp.

7.2 ICOMP
As stated before, this case study has been carried out at the company Icomp. The current section will briefly introduce it, considering one particular view on Icomp, namely the 'financial products distribution' part of its business (this view has already been mentioned in the previous section). Further on, we will mean by Icomp just those things concerning the company, which are associated with this particular view.

This perspective on Icomp has been taken (as explained) because of its direct relation to the core of the company's business, namely distribution of financial products to end customers through brokers. As the studied information shows, particularly this essential part of the business of Icomp would (eventually) need an application support.

Distributing financial products through brokers means that there are a number of (insurance) financial companies, a number of brokers, and a number of end customers, concerning this distribution mechanism. Broker \( j \) distributes products of a number of companies (including Icomp, if it has an agreement with Icomp) to a number of end customers. End customer \( k \) might be advised by a number of brokers about the products of a number of financial companies. Hence, Icomp uses a number of brokers through which it distributes its (financial) products to a number of end customers.
customers. Thus, we could relate Icomp basically to two actor-role types, namely 'BROKER' and 'END CUSTOMER', as shown on Figure 7.1. BROKER could be fulfilled by any of the intermediary (brokerage) companies registered with Icomp. END CUSTOMER could be fulfilled by any human or organization interested in the financial products distributed by Icomp.

![Diagram](image)

Figure 7.1: Brokers, facilitating the relations of Icomp with end customers

Thus, on the figure, the line between Icomp and END CUSTOMER is dashed, indicating that the relation to end customers is indirect; it comes through brokers.

The brokers collaborating with Icomp, distribute its financial products on the basis of an agreement specifying which products the particular broker could sell to end customers and what commission the broker would get from Icomp. The following information elaborates further on the Icomp-broker relation:

- An agreement can be started/changed/ended between Icomp and a broker.
- A broker might receive support from Icomp. For example, if a broker has been successful in selling products (of Icomp) to representatives of a particular customer segment, it might be useful that Icomp provides to the broker a specialized training concerning this particular segment.
- The commission paid by Icomp to a broker is as follows:
  - For each new agreement, a broker gets 'starting commission'.
  - For each month in which an end customer keeps his/her insurance (particularly advised by a broker), the broker gets 'monthly commission'.
- A broker must pay a premium to Icomp for an agreement initiated.

With respect to the financial products distribution, Icomp has relations not only with intermediary (brokerage) companies but also with re-insurance companies, product development companies, investigation companies, and other (less important) ones.

It is possible that in some cases a re-insurance company takes over insurance risk from Icomp.

In complicated situations, Icomp relies on investigation companies for the provision of expert support, for instance: realization of an expertise.

For keeping its product portfolio actual, Icomp receives support from product development companies delivering new financial (including insurance) products.

As for Icomp itself, it is essential to consider its being divided into five departments: Account Management, Acceptance, Claims, Finance, and Marketing.

The Account Management department manages the Icomp-broker relations. It proposes agreement(s) to a broker and once an agreement is signed, the department controls its execution by making sure that the broker's results are in accordance to what is in the agreement.

The Acceptance department handles requests of end customers for financial product(s), for example, a request for a property insurance.

The Claims department deals with claims of end customers and the (eventual) investigation of these claims by experts.
The *Financial department* deals with payments, including the premium payments received by Icomp from end customers, the payments of Icomp to end customers for claims, the commission payments that brokers receive from Icomp, and the payments of Icomp to product development companies.

The *Marketing department* is responsible for the product strategy of Icomp, dealing with product development, and also with advertising and public relations.

The following section focuses on the application of the SDBC approach.

### 7.3 APPLYING SDBC

Based on the case study background on one hand and on the information about Icomp, on the other hand (addressed subsequently in the previous sections), this section is to elaborate on how SDBC could be applied within the context of the Icomp case, following the guidelines introduced (and also motivated) in Chapter 6.

The section is divided into three sub-sections:

- Sub-section 7.3.1 will focus on the considered (within the case study) information and the identification (based on this information) of several relevant Business CoMponents.
- Sub-section 7.3.2 is to consider the specification and elaboration of a particular Business CoMponent.
- Sub-section 7.3.3 will demonstrate the derivation of a software specification model, based on the specified Business CoMponent (Sub-section 7.3.2).

Since the steps-to-follow in applying the SDBC approach have been introduced, explained, and discussed in the previous chapter, we will now just follow those of them relevant to the tasks within the case, without explaining in much detail those steps.

#### 7.3.1 From the Case Information to Business CoMponents

As mentioned before, in this sub-section, we will show how Business CoMponents could be identified based on the case information (Section 7.2). We provide below a *roadmap* (fully consistent with the SDBC application guidelines (Chapter 6)) which gives in advance information about the modeling activities (steps) to take place within the current sub-section, in order to achieve what we have already defined as a goal:

- **Step 1**: Building of a generalization hierarchy for the explored domain.
- **Step 2**: Identification of relevant actor-roles.
- **Step 3**: Identification of the corresponding inter-role actions (relations).
- **Step 4**: Elaboration of these relations with Semiotic Norms.
- **Step 5**: Decomposition of Icomp and a related positioning of the relations.
- **Step 6**: Construction of a SCI chart.
- **Step 7**: Derivation of Business CoMponents.

These seven steps will be followed within the current sub-section.

**Building of a generalization hierarchy for the explored domain**

Structuring and positioning semantically the case information is in line with Activity 1 (from the SDBC Input/Output Model (Chapter 4)) and the guidelines presented in Chapter 6.
As a starting point from the case information (Section 7.2) we select the entities (natural/legal persons) who collaborate with the target company (Icomp). They are (in alphabetical order): intermediary companies, investigation companies, product development companies and re-insurance companies. Investigation companies are actually a sub-type of consultancy companies (according to the interviewed specialists from Icomp). The rest of the mentioned ones are sub-types of financial companies. Being an insurance company itself, Icomp is a financial company too.

This information is sufficient for identifying a generalization hierarchy (organizational business objects model) for the explored domain. The hierarchy is charted in accordance with the guidelines proposed in the previous chapter. The reason for doing this, as mentioned in the chapter, is to bring order in the original input information. The organizational business object model regarding Icomp is shown in Figure 7.2 a:

![Organizational Business Object Model](image)

**Figure 7.2 a: The organizational business object model concerning the case study**

Hence, as seen from the figure, Icomp collaborates with:
- three types of financial companies, namely – 1) re-insurance compagnie(s), 2) intermediary compagnie(s), and 3) product development compagnie(s);
- investigation compagnies which are type of consultancy compagnies.

The position of Icomp within this model is also clear – Icomp is an insurance company.

Besides these types of companies, Icomp collaborates also, of course, with its customers. According to the considered case information, a customer of Icomp might be any person, legal or natural. This is illustrated on Figure 7.2 b where 'LP' stands for legal person and 'NP' stands for natural person.

![Customer Business Object Model](image)

**Figure 7.2 b: The business object model regarding the customers of Icomp**
In the rest of this chapter, any customer, no matter to which of the two customer sub-types (Figure 7.2 b) belongs would be called 'Customer'. The reason is that Icomp does not distinguish between its customers in any way. It is to be stated also that besides the term 'Customer', the materials concerning Icomp, including the case briefing, contain also the terms 'Client' and 'End customer'. Actually, these two are synonyms of 'Customer'. Therefore, they will be left out for the rest of this chapter and only the term 'Customer' will be used.

Identification of relevant actor-roles

Following the roadmap, the next step is to produce an actor role model based on the business object models. As studied and motivated within the current thesis, if the actor-role concept is applied, then it would be easier to model complex systems – we have mentioned as well examples of role substitution, which are in support of this view: for instance, if a manager sends a fax, then (s)he plays the role 'Secretary'. Hence, in such a case, if we do not model an actor-role, we should either model the individual natural/legal persons (which is too much complicated in such situations), or oversimplify these issues, which might lead to limitations in the business model being created. Further on, the term 'role' will be used, meaning 'actor-role'. Some relevant concepts and definitions are to be found in Chapters 2 and 3.

We proceed, thus, towards a role identification which is in tune with the SDBC application guidelines (Chapter 6). It starts with an initial consideration of the roles which are typical for each of the identified company types (Fig. 7.2 a). In particular, the starting point is to find (formulate) a suitable word for each of these roles. This is done by studying the case information (customers should be considered, too). The next step is to find out whether there is information of a type of company which in some situations takes roles different from its typical role (in the Icomp case we do not have such examples).

However, formulating a word does not give full information about the meaning of the role. Therefore, the word should be extended with some elaboration. By role elaboration is meant describing what characterizes a particular role. We do this as follows:

- The typical role type for a reinsurance company is formulated as REINSURER: one fulfils REINSURER if taking over a risk from an existing insurance. A re-insurance company is not expected to take other roles.
- The typical role type for an intermediary company is formulated as BROKER: one fulfils BROKER if matching customers to relevant financial companies, in particular – insurance companies (in this case), by 1) giving to customers financial consultations about these companies; 2) directing customers to particular companie(s) if there is a match between customer requirements and company product(s) – this direction is realized by advising for a product of a particular company. An intermediary company is not expected to take other roles.
- The typical role type for a product development company is formulated as SUPPLIER: one fulfils SUPPLIER if delivering financial products to insurance companies. A product development company is not expected to take other roles.
- The typical role type for an investigation company is formulated as EXPERT: one fulfils EXPERT if realizing expertises (analyses and investigations) for insurance companies. An investigation company is not expected to take other roles.
- In is necessary also to formulate the role type CUSTOMER: one fulfils CUSTOMER if purchasing financial (including insurance) products and providing specialized information upon request.
- Icomp, as the target company in this study, fulfills the role type INSURER; one fulfills INSURER if selling financial (including insurance) products.

We stress on the fact that we have identified role types (rather than particular roles). This means that, speaking of REINSURER, for example, we are not interested in any particular instance(s) related to this role type. It could be fulfilled by many reinsurance companies.

These (identified) role types are expected to relate somehow to initiators/executors of particular transactions. This, as a part of the modeling output reflected in the current section, would facilitate the identification of business components specified in the notations of DEMO.

Figure 7.3 shows the identified role types and also their elaborations. They are depicted in rectangles outlined by double line. Attached to them are rectangles outlined with single line. The role elaborations are depicted in them.

![Diagram of role types](image)

**Figure 7.3: Basic role types within the Icomp case**

As seen from the figure, there are six role types: INSURER corresponds to insurance companies (such is the company under study (Icomp)), CUSTOMER corresponds to the customers of insurance companies, and the other roles are straightforwardly derived from the hierarchy model represented on Figure 7.2 a: REINSURER (Figure 7.3) is the role type typical for a Reinsurance company (Figure 7.2 a), BROKER (Figure 7.3) is the role type typical for an Intermediary company (Figure 7.2 a), and so on. As for the role elaborations which are also depicted on Figure 7.3, they have been formulated based on the case study information and interviews with employees of Icomp.

**Identification of inter-role relations**

Based on the identified major role types, the actions (relations) among them are studied. We will call these relations 'inter-role relations' from now on, or 'relations' for short. Studying the relations would be useful with regard to a consideration of the structure and dynamics of the explored business system. As a first step in identifying the existence of relations, the interviewed Icomp employees were asked to answer whether or not a relation exists between each two of the role types. Table 7.1 contains the collected data. As seen from the table, only the grey rows correspond to an existing relation. For example, from the third row (from top to bottom) it is seen that there exists a relation between INSURER and EXPERT.
<table>
<thead>
<tr>
<th>INSURER</th>
<th>REINSURER</th>
<th>Yes, a relation exists</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSURER</td>
<td>BROKER</td>
<td>Yes, a relation exists</td>
</tr>
<tr>
<td>INSURER</td>
<td>EXPERT</td>
<td>Yes, a relation exists</td>
</tr>
<tr>
<td>INSURER</td>
<td>SUPPLIER</td>
<td>Yes, a relation exists</td>
</tr>
<tr>
<td>INSURER</td>
<td>CUSTOMER</td>
<td>Yes, a relation exists</td>
</tr>
<tr>
<td>REINSURER</td>
<td>BROKER</td>
<td>No, a relation does not exist</td>
</tr>
<tr>
<td>REINSURER</td>
<td>EXPERT</td>
<td>No, a relation does not exist</td>
</tr>
<tr>
<td>REINSURER</td>
<td>SUPPLIER</td>
<td>No, a relation does not exist</td>
</tr>
<tr>
<td>REINSURER</td>
<td>CUSTOMER</td>
<td>No, a relation does not exist</td>
</tr>
<tr>
<td>BROKER</td>
<td>EXPERT</td>
<td>No, a relation does not exist</td>
</tr>
<tr>
<td>BROKER</td>
<td>SUPPLIER</td>
<td>No, a relation does not exist</td>
</tr>
<tr>
<td>BROKER</td>
<td>CUSTOMER</td>
<td>Yes, a relation exists</td>
</tr>
<tr>
<td>EXPERT</td>
<td>SUPPLIER</td>
<td>No, a relation does not exist</td>
</tr>
<tr>
<td>EXPERT</td>
<td>CUSTOMER</td>
<td>Yes, a relation exists</td>
</tr>
<tr>
<td>SUPPLIER</td>
<td>CUSTOMER</td>
<td>No, a relation does not exist</td>
</tr>
</tbody>
</table>

Table 7.1: Identified inter-role relations

The next step, according to the SDBC guidelines (Chapter 6), is to describe briefly each identified relation. In achieving this, we will first consider in more detail the particular role types and second – address the aspects that concern their relations.

As stated in the previous chapter, the first sub-task could be realized through binary relationships. As already explained: a binary relationship does concern two entities and is usually described by: noun-verb-noun; the nouns correspond to the entities and the verb describes the relation among them. If we take, for example, the role type COMPOSER, related to the expression: ‘writing songs’, then we could form a binary relationship. It would be between COMPOSER and song (between the role type and something related to its output). The verb ‘write’ describes how these two relate.

Thus, looking at Figure 7.3, we could analogously form binary relationships since, as it could be seen, a role type name plus a role elaboration could be considered as a noun-verb-noun expression. Thus we have:

- **INSURER** - sell - (insurance) products
- **REINSURER** - take over - (insurance) risk
- **BROKER** - give - (financial) consultation
- **BROKER** - advise for - (insurance) products
- **EXPERT** - realize - expertise
- **SUPPLIER** - deliver - (financial) products
- **CUSTOMER** - purchase - (insurance) products
- **CUSTOMER** - provide - (specialized) information

Further on, according to the guidelines, we extend (based on Table 7.1) each of the above eight expressions with one more noun corresponding to a role type which relates to the role type represented within the particular expression. This is done as follows: For a particular role type, we can see the ‘candidate’ matches from the table. Thus, we have to choose any of them. The criterion is how it matches the context of the expression. For example, starting from INSURER, we see from Table 7.1 that it relates to REINSURER, BROKER, EXPERT, SUPPLIER, and CUSTOMER. Therefore, we ask the question: To whom does INSURER sell insurance products? The answer (according to the case information) is: ‘to CUSTOMER’. Therefore, we extend the first expression with CUSTOMER:
If we go further, we see from Table 7.1 that REINSURER relates only to INSURER; we ask the question: From whom does Reinsurer take over risk? The answer is: ‘from INSURER’. Therefore, we extend the second expression with INSURER:

**REINSURER** - take over - **(ins.) risk** - **INSURER**

We continue analogously:

**BROKER** - give - fin. consultation - **CUSTOMER**
**BROKER** - advise for - financial products - **INSURER**
**EXPERT** - realize - expertise - **INSURER**
**SUPPLIER** - deliver - financial products - **INSURER**
**CUSTOMER** - purchase - insurance products - **INSURER**
**CUSTOMER** - provide - spec. information - **EXPERT**

We now need to consider (as explained in Chapter 6) the (above) expressions and check them against redundancy - there is a risk to describe twice one and the same thing, like in the following two expressions:

**INSURER** - **sell** - insurance products - **CUSTOMER**
**CUSTOMER** - purchase - insurance products - **INSURER**

Considering the case information, we have concluded that the information in the above two expressions is about one and the same thing, namely the INSURER’s selling of insurance products to CUSTOMER. Therefore, we randomly choose one of the above two expressions and leave out the other one. Let’s select the first one.

Further on, following the SDBC application guidelines, we will use the above expressions as an input for building the Icomp **RR chart**. Chapter 6 has introduced it in general, as a chart facilitating the description of relations (RR stands for Roles & Relations). In order to build the chart, as described in the chapter, we need to consider the above expressions, putting the role type names in boxes. The names of those of the role types which relate to the realization of a particular activity (for example, the activity: ‘sell insurance products’) are underlined. Next to that, the name of the role type corresponding to the target (within the case study) organization should disappear. On its place we put the particular name of the organization (Icomp). Hence, we should replace the type role (INSURER) with the instance role (‘Icomp’). This is because we are not interested in any company which could fulfill INSURER but in this role as performed in particular by the company Icomp.

Between each two boxes, concerning role types and characterizing a particular relation, we should put together all the text (from the corresponding expressions above) which is between the names of the role types. For example, we take the text: ‘realize expertise’ from the line:

**EXPERT** - realize - expertise - **INSURER**

The RR chart is depicted on Figure 7.4. As seen from the figure, each line contains two role type names (the name of the target company is in some places instead a role type name) and in between is the description of the relation. All these are derived straightforwardly from the previously constructed expressions. As it could be seen, we have also given a unique code to each relation (R1 to R7). Onwards, we will refer to each of the modeled relations using these codes.
Norm elaboration

According to Chapter 6, an important step to take place is adding further precision to the description of the relations, by applying Organizational Semiotics, and in particular - Norm Analysis. The strengths of this Semiotic tool for modeling (complex) relations, are known. The choice of Norm Analysis has also been discussed and motivated (Chapter 5 and Chapter 6). The construction of norms has been introduced in Chapter 3. Hence, no explanations will be made here about how the norms are constructed on the basis of the information on a relation. A norm will be attached to each relation (Figure 7.4), being given a name containing ‘N’ (from the word ‘norm’) and the code of the relation. The seven constructed norms are:

NR1
Whenever BROKER has advised CUSTOMER in favour of a Icomp’s product and CUSTOMER fits within Icomp’s policy
If CUSTOMER decides to purchase this product
Then Icomp
Is obliged to insure CUSTOMER according to the concrete product details and based on a payment from CUSTOMER, made accordingly.

NR2
Whenever there is a long run relation between Icomp and REINSURER
If an insurance to be realized by Icomp would include a unacceptably high risk for Icomp and the insured objects fit within REINSURER’s policy
Then (if asked) REINSURER
Is obliged to take over risk(s) from Icomp regarding the particular insurance.

NR3
Whenever CUSTOMER has a request for consultation to BROKER
If an insurance company having got an agreement with BROKER has an appropriate product with regard to the CUSTOMER’s particular request
Then BROKER
Is obliged to consult CUSTOMER about this product.
NR4
**Whenever** there is an agreement between Icomp and BROKER
**If** a product of Icomp is a best match with regard to a CUSTOMER's request
**Then** BROKER
**Is obliged to** do advice for CUSTOMER in favour of Icomp's product(s).

NR5
**Whenever** there is a non-standard situation regarding a stated claim
**If** Icomp asks EXPERT for an expert evaluation (expertise)
**Then** EXPERT
**Is obliged to** realize an expertise with regard to the stated claim.

NR6
**Whenever** there is an agreement between Icomp and SUPPLIER about delivery of insurance products
**If** CUSTOMER wants to have a product whose production and delivery falls in the mentioned agreement as a responsibility of SUPPLIER, and Icomp has ordered this financial (in particular - insurance) product to be developed
**Then** SUPPLIER
**Is obliged to** deliver the financial product.

NR7
**Whenever** EXPERT is involved in an expert evaluation (expertise)
**If** EXPERT asks CUSTOMER for specialized information
**Then** CUSTOMER
**Is obliged to** cooperate by providing the required information.

**Positioning of the relations**

We have realized so far an identification and a thorough elaboration of the essential relations concerning the Icomp case.

Our position towards Icomp as the company under study requires adding more precision about the way Icomp handles the mentioned relations. Said otherwise, it is of interest to know which of the departments (organizational units) within the company are involved in which of the relations. Such information would be of significant importance for specifying an ICT application which, for example, might operate across some (or all) of the mentioned departments.

Therefore (and according to Chapter 6), the next step should be to position the relations in terms of the Icomp organizational units. These units are defined based on the information about Icomp (Section 7.2).

We consider Figure 7.4 and leave out of consideration the relations R3 and R7 because they do not relate directly to Icomp. We take then the remaining relations (which concern Icomp) and conduct interviews in order to clarify for each particular relation the involved Icomp departments. Of course, it appears that often a relation concerns more than one department. For example, the relation between Icomp and BROKER comes firstly through the Account management department (considering the agreement and also the Icomp – BROKER collaboration in general) and secondly, through the Financial Department (as long as payments are concerned).

Figure 7.5 contains the results. The names of the departments (of Icomp) are put in rectangles. Each relation (in circle) having connection with a department is linked to it.
We have purposefully simplified slightly the way we look at the organizational structure of Icomp because this would make our further modeling activities easier to understand. However, the modeling complexity would be sufficient for adequately testing the strengths of the SDBC approach.

![Diagram](image)

*Figure 7.5: Relations and organizational units concerning the Icomp case*

**Construction of a SCI chart**

Based on the modeling results which have been achieved so far, we will (according to Chapter 6) apply the SCI chart in order to summarize the initial case information. The chart has been thoroughly introduced in the mentioned chapter (SCI stands for Structuring Customers’ Information) and no explanations will be presented now either on the chart or on the way it is constructed. Anyway, in principle, the modeling outputs depicted in Figures 7.3, 7.4, and 7.5 should be a sufficient basis for constructing the chart. The SCI chart is presented in Figure 7.6 where the following abbreviations are used:

- **am** stands for ‘Account management department’;
- **md** stands for ‘Marketing department’;
- **fd** stands for ‘Financial department’;
- **ad** stands for ‘Acceptance department’;
- **cd** stands for ‘Claims department’.

On the figure, the target organization (Icomp) is represented within the rectangle with rounded corners. Inside are depicted the five departments -> source: Figure 7.5; within an attached rectangle is an elaboration concerning Icomp. In rectangles around the rounded cornered rectangle, are depicted the five considered role types plus their elaborations -> source: Figure 7.3. On the basis of Figures 7.4 and 7.5, the role types are linked (where appropriate) to corresponding departments within Icomp. Also, where appropriate, the roles are linked to each other. Each line is given a number.

In this way, through the SCI chart, we have achieved a compact, complete, and focused view on the target organization (and relevant information).
Figure 7.6: Icomp – SCI chart

Derivation of Business CoMponents

According to the guidelines for application of SDBC (Chapter 6), a SCI chart could facilitate the identification of Business CoMponents, particularly Business CoMponents specified in the notations of DEMO.

According to the guidelines, based on the SCI chart, we could consider the lines. Each line originates one or more Business Process patterns, as specified in the previous chapter. In this patterns we consider the organizational units within Icomp as roles. Hence, the set of Business Process patterns, derived from the Icomp SCI chart is:

1  am  start  AGREEMENT  BROKER
1  am  end  AGREEMENT  BROKER
1  am  manage  AGREEMENT  BROKER
2  fd  pay  COMMISSION  CUSTOMER
3  BROKER  advise  (for a) PRODUCT  CUSTOMER

The core of the business of Icomp is selling of insurance products to humans and organizations.
This output represents the starting point for the identification of Business Componenets. Essential for this is the discovery of DEMO Transactions. It is claimed (and motivated) that the above output could facilitate the mentioned discovery. Next to that, this output's being focused adds value to the overall consistency of the set of Transactions and Business CoMonents (being identified). This means that the resulting DEMO models would adequately reflect the considered business reality.

We will take, for illustrative purpose, several of the above Business Process patterns. Through identification of Transactions, we will reflect these patterns in DEMO Coordination Structure Models, identifying in this way Business CoMonents. We will consider, in particular, the patterns which are represented in bold on the above list, starting with the following one:

1. am start AGREEMENT BROKER

Firstly, we are clear what the system under study is. It is Icomp (the Account management department is one of its departments) - this is reflected in one of the roles in the above expression. Secondly, we are clear which the roles under consideration are; in this case they are 'am' and 'BROKER'. Hence, we could model this, using DEMO notations, as follows (Figure 7.7):

![Diagram](Image)

Figure 7.7: Representing a pattern in DEMO notations
Chapter 7

What we know also from the pattern is the essence of the inter-role relation: 'start agreement'. However, reflecting it directly in one Transaction would not provide a complete view, as explained in the previous chapter. We need though to analyze this information and identify the starting Transaction. To achieve this, one would (usually) answer the helpful question: 'what is the cause'. We have done this, discovering that a broker could have an agreement started only based on an application (submitted). Therefore, the starting Transaction would be:

\[ T2 \text{ application} \quad F2 \text{ application } <A> \text{ has been submitted} \]

We ask then what happens next. It is that \( am \) receives an application from a broker and, before being able to start an agreement with the broker, \( am \) needs an approval by a controller within Icomp (we have not considered it so far because of it not having a significant importance). Thus, we identify an additional role-type, namely CONTROLLER. As for the corresponding Transaction, it would be:

\[ T3 \text{ approval} \quad F3 \text{ approval concerning application } <A> \text{ has been done} \]

Based on this approval is the starting of an agreement, by \( am \):

\[ T1 \text{ agreement} \quad F1 \text{ agreement based on application } <A> \text{ has started.} \]

Hence, taking the information from Figure 7.7 and also the identified (above) Transactions and a role-type, we are able to build the relevant Business CoMponent. This is depicted in Figure 7.8.

![Figure 7.8: An identified Business CoMponent – structural view](image)

Considering the above output and following the DEMO mapping mechanisms, we construct (Figure 7.9) as well a model that elaborates on the communicative aspects concerning the Transactions.

We will be more detailed about the elaboration of identified Business CoMponents in the following sub-section. In this sub-section we just consider the identification of Business CoMponents.

We continue with the rest two Business Process patterns to be considered and reflected in Business CoMponents:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>fd</td>
<td>pay</td>
<td>PREMIUM</td>
<td>SUPPLIER</td>
</tr>
<tr>
<td>8</td>
<td>fd</td>
<td>pay</td>
<td>CLAIM</td>
<td>CUSTOMER</td>
</tr>
</tbody>
</table>
Proceeding analogously, we will identify Business CoMponents based on the patterns.

Since both patterns concern payment, we propose (based on the SDBC application guidelines) using a re-usable Business CoMponent, in particular a general Business CoMponent. It is to be extended afterwards. We are not going to explain these issues since they have been considered both conceptually (Chapter 4) and within the mentioned guidelines (Chapter 6).

A general payment Business CoMponent specified in the notations of DEMO is depicted in Figure 7.10.

As seen from the figure, in the general case, we have an organization providing a service to a customer and claiming therefore payment in return. Usually, the entity
delivering the service is not the entity handling the payment: there are two internal role-types depicted on the figure, therefore, namely SELLER and PAYMENT CONTROLLER. SELLER delivers a service to the customer ('BUYER') and informs about this PAYMENT CONTROLLER who as a result of self activation (on a periodic basis) would handle the payment.

Taking the first of the two considered patterns, we extend straightforwardly the model shown in Figure 7.10. BUYER in this case would be Icomp (its Marketing department (md) buys a financial product and its Financial department (fd) has to pay to a corresponding supplier). This is represented on Figure 7.11:

![Figure 7.11: A possible extension of the general payment Business CoMponent](image)

As for the second pattern, we again reflect straightforwardly the information: this time the payment should be directed to a customer. However, before the payment could be initiated (as studied from the case information) it is necessary that an expert (external to Icomp) investigates the case. Considering this accordingly, we derive a Business CoMponent, as represented in Figure 7.12.

![Figure 7.12: Another possible extension of the general payment Business CoMponent](image)

So, we have demonstrated Business CoMponents identification. In the following sub-section, we will consider the elaboration of a particular (identified) Business CoMponent.
7.3.2 Elaborating a Business Component

In the previous sub-section, we have demonstrated the identification of Business Components, using SDBC. As mentioned at the beginning of the current section: in this sub-section, we will demonstrate the specification and elaboration of an identified Business Component; in the following sub-section, we are going to demonstrate the derivation of a software specification model, based on the Business Component.

As for the particular component to be considered, it will not be one of the components identified on the basis of the SCI chart (Sub-section 7.3.1). It will be, instead, a Business Component resulting from a business improvement proposal concerning Icomp (the conceptual framework of SDBC allows for business re-design, as a possible design step, whenever this is considered necessary, as mentioned in Chapter 4).

Our reason for introducing a business improvement proposal is that such an improvement is expected to create an adequate foundation for realizing a useful software support to Icomp while simply automating any currently existing business processes within the company would bring less value to it.

Therefore, we will address below:
- the problem concerning the need for improvement;
- a relevant business improvement proposition;
- a resulting Business Component, to be adequately specified and elaborated.

The Business Component will be elaborated in terms of structure, dynamics, data and communicative issues (Figure 7.13), as according to the SDBC approach.

![Diagram of Business Component](image)

*Figure 7.13: Elaborating a Business Component*

**Problem statement**

Regarding some relatively simple cases in which an advice is straightforwardly deliverable (based on relevant information and rules), using human brokers is too expensive. It would be more appropriate if human brokers are used just in cases in which their particular expertise is to be applied.

**The Financial Mediator – a proposition**

We have made, reflecting the above problem, a business improvement proposition according to which a new business unit is to be introduced, namely a Financial Mediator (FM).

The FM facilitates Dutch insurance companies. In order to use FM, a company should subscribe (for free). FM brings about the following useful deliverables:

- advice (to customers or insurance brokers) on what of the offered (by the registered companies) products best satisfies a particular customer demand;
• delivery (to customers) of products of insurance companies.

Any customer could request (for free) FM to do for him/her either advice or delivery of a product. The customer should firstly specify his/her request (choosing from a list): (s)he should make clear whether the request is about a health insurance, auto insurance, and so on, specify the particular demand (for example: to insure a car against theft with the highest possible coverage (that includes car accessories, tires, and so on)), and so on. Based on this, a request processing unit within FM generates a standardized specification regarding the customer’s request, which is delivered to a match-making unit within FM. This unit is to further realize a match allowing the FM to do the advice. The match is driven by the particular criterion chosen by the customer (for example: a preference for the cheaper or the most reliable product available). In order to deliver such a criterion-driven match, the match-making unit uses a data bank of relevant rules and procedures. Besides the output given by the request processing unit, the match-making unit needs as well an output from a data search and processing unit. It searches through the information that concerns registered companies, applying procedures to it. This allows for a precise identification of candidate-matches, relevant to the particular customer’s request. Thus, given this output plus the (mentioned) standardized specification of the customer’s request, the match-making unit would be able to realize a match, applying the mentioned rules and procedures.

As for the subscription of (insurance) companies, any (Dutch) company could subscribe for free. This is facilitated by a subscription processing unit within FM. This unit could realize a subscription only after another unit within FM (a company profile builder) creates a profile of the particular company, making its data available through a data-bank (to be usable also by the data search and processing unit). Usually, FM creates ‘standard profiles’; however, several special companies could have ‘golden profiles’ (with more benefits).

Allowance: a customer’s using FM (either for advice, or contract, or product delivery) is to be limited to no more than five times per month. As for (insurance) companies’ allowance, a company is allowed to subscribe to FM only if it is licensed according to the Dutch financial laws.

As for a product delivery: once a customer has chosen a product, (s)he might request that FM facilitates the actual product delivery. The customer requests an offer (FM is to be authorized to generate offers, based on information from the particular company, kept in its profile). Once FM (its offer generating unit) has produced an offer, it should have it first approved by the respective (insurance) company, before delivering it to the customer. From the moment of the delivery, the particular insurance (or other financial product) is in effect – between the customer and the corresponding (insurance) company.

A company should pay a commission to FM for each realized (through FM) insurance (or other product).

**Financial Mediator (FM) — the Business CoMponent**

On the basis of the above proposition, we identify a relevant Business CoMponent, namely the FM Business CoMponent.

Since we have already demonstrated the SDBC Business CoMponent identification mechanism, we will not demonstrate the identification itself again (it has been done analogously, as in the previous sub-section). The current sub-section aims instead (as stated already) at demonstrating the specification and elaboration of a Business CoMponent.
Hence, we go directly to the **identified Transactions** (the Transactions listed below). The first six of them relate to the FM's delivering advices. The rest (backgrounded grey) relate to the FM's contracting concerning financial (insurance) products.

| T1  | Deliver advice     | F1  | advice <A> is delivered |
| T2  | Perform match-making | F2  | match of advice A is made |
| T3  | Generate customer's information specification | F3  | customer's information concerning advice A is processed |
| T4  | Generate candidate-matches | F4  | data search and processing is done concerning advice A |
| T5  | Realize subscription | F5  | Subscription <S> is realized |
| T6  | Create profile     | F6  | profile is created concerning subscription S |
| T7  | Offer contract     | F7  | contract <C> is offered |
| T8  | Approve contract   | F8  | contract C is approved |
| T9  | Deliver financial product | F9  | the product specified in contract <C> is delivered |
| T10 | Submit agreement   | F10 | agreement concerning contract C is submitted |
| T11 | Accept contract    | F11 | contract C is accepted |
| T12 | Activate payment collections | F12 | payment activation <A> is realized |
| T13 | Realize payment    | F13 | Commission for product(s) specified in contract <C> is paid |

As seen from the above list, FM could deliver an advice. This requires that a match-making is performed, based on a standardized specification of the customer's information and on generated candidate-matches. As for the consideration of (insurance) companies, FM could offer them subscription. It is completed only after a particular company profile has been created by FM.

It is seen also that once a customer has chosen a particular financial (insurance) product, it could be facilitated by FM for the product's delivery. FM offers a contract based on which the customer would acquire the product. The contract, however, would need to be approved by the particular company, before being offered to the customer. After it has been offered, the customer should accept it and from this moment on, (s)he has rights and obligations concerning the product. For each product delivery, a payment of commission should take place, from the particular financial (insurance) company to FM. A payment controller is activated periodically in collecting all payments due for the particular period.

Further on, we will reflect these Transactions in DEMO models, offering elaboration in two aspects: structural and communicative. We will further elaborate the DEMO models with Semiotic Norms, following the steps presented in Chapter 6.

Afterwards we will derive, based on these models, Petri Net and ORM models, offering elicitation in terms of dynamics and data, respectively. We will attach to the Petri Net models some further refined norms.

All this is in concert with the SDBC application guidelines. They have been presented in detail in the previous chapter.

Hence we will address further the structural, communicative, dynamic, and data aspect of the considered business reality. This four-aspect business view (Figure 7.13) is in tune with the SDBC foundations (Chapter 4).
Financial Mediator — structural and communicative aspects

For clarity’s sake, in modeling the above Transactions, we will firstly consider those of them which concern the FM’s offering advice services, and secondly — those concerning the FM’s offering contracting services (backgrounded in grey color).

As for the first of the (mentioned) Transaction groups, we have reflected it in the DEMO Coordination Structure model, represented on Figure 7.14. The model concerns the structural Business CoMponent aspect.

As mentioned before, the identification of such a model has been demonstrated, including the identification of roles and Transactions. Hence, we take them directly in building the model, without explaining how we have identified them.

The functionality of FM concerns customers and insurance companies (for short — ‘IC’). Hence, we have two major role-types: CUSTOMER and INSURANCE COMPANY. As seen from the figure, in the DEMO model, they are reflected as the roles ‘Customer’ (S02) and ‘Insurance Company’ (S03); these roles are external with respect to FM. The transactions T01 and T05 concern the FM-Customer and FM-IC relations, respectively.

Next to that, a number of actions take place within FM where we have identified six roles (internal as concerns FM). They are: A01 (Advisor), A02 (Match-maker), A03 (Request Processing Unit), A04 (Data Search & Processing Unit), A05 (Subscription Processing Unit), and A06 (Company Profile Builder). Transactions T01, T02, T03, and T04 as well as roles A01, A02, A03, and A04 concern directly the advice delivery: they are about the mere (FM’s) delivery of an advice to a customer. Transactions T05 and T06 as well as roles A05 and A06 concern indirectly the advice delivery: they are about the collection and use of information (concerning Insurance companies) needed for performing an advice.

As seen from Figure 7.14, A01 is to deliver advice (T01). However, this could be done only based on a realized match-making — a matching between what the particular customer requests and what is offered by the Insurance companies (registered with FM). A02 is to realize such a match-making (T02). What A02 needs in order to realize the match are two things: 1) a complete specification regarding the request of the (particular) customer, a specification presented in standardized notations (the reason is that if such a specification is not standardized it would be hardly matchable with information concerning insurance companies); 2) list of candidate-matches. A03 must generate the mentioned standardized specification (T03) and A04 should provide candidate matches (T04). In performing T04, A04 is facilitated by two data-banks, namely DB01 and DB02. These data-banks are claimed to be an essential elaboration concerning the model. Using DB02, A04 gets direction what procedures to apply (and where to find them) in connection to a particular need expressed by a customer. For example, if a customer needs an auto-insurance, following a procedure helps to adequately direct a further searching through companies (in this particular case — searching in their property insurance departments and/or ‘schemes”). Based on such an orientation achieved, A04 could effectively direct its search for relevant (insurance-companies-related) information, using the bank DB01. It contains information about the (insurance) companies registered with FM, the (financial) products offered by them, and also other related details. Thus, using these (mentioned) two data-banks, A04 should be able to provide to A02 a list of candidate-matches with regard to the particular (customer) request. Therefore, based on the request specification (delivered by A03) and this candidate-matches list, A02 must realize the match-making. However, this should be done according to a particular criterion (like reliability or quality of service, for instance). It should be specified by the customer. Having received this information,
A02 should apply particular procedures in approaching the ‘matching’ information (this information would be considered in one way if the cheapest (financial) product is the goal and in another way if the most reliable product is to be selected). With respect to this, A02 is facilitated by the data-bank DB03. It allows A02 to know what procedure (or a combination of procedures) to apply to the ‘matching’ information based on a criterion chosen by the customer. The data-banks and related information will be considered in more detail further on within the current sub-section, when we come through the data business component aspect.

Also, it is seen from Figure 7.14 that A05 is to realize the subscription (T05) of a (insurance) company wishing to use FM. Before a subscription could be handled, a company profile is to be built (T06) by A06. This includes adding data to the data-bank DB01 which was mentioned already.

As also seen from the figure, the realization of the T01 Transaction includes providing (relevant) information to A02, A03, and A04 (the dotted lines between T01 and these roles indicate this). A02 is to receive the criterion (chosen by the customer) according to which to perform the match-making. A03 should receive the (full) information submitted by the customer. A04 should be provided with the type of the customer need (for instance: ‘auto-insurance’).

We have done, thus, the basic elaboration on the DEMO model and will then (following the guidelines in Chapter 6) add further elicitation using Semiotic Norms. This is a logical continuation of the norm derivation characterizing the earlier analysis phases (these phases addressed in the previous section, have presented derivation of more general Semiotic Norms intended to ‘govern’ the ones to be identified).

Since the role of norms and their derivation have already been explained in previous chapters, we will directly go to the content of the identified norms attached to the DEMO Transactions. Since they concern the essential level, we will add identification to them as follows: a string consisting of ‘E’ (from ‘Essential’), ‘N’ (from ‘Norm’) and a number of the particular Transaction. The derived norms are below:

EN1
**Whenever** S02 has requested advice
**If** A02 has realized match-making
**Then** A01
**Is obliged to** formulate and deliver an advice.

EN2
**Whenever** S02 has requested advice
**If** A03 has delivered standardized customer specification AND A04 has delivered candidate matches
**Then** A02
**Is obliged to** realize match-making.

EN3
**Whenever** S02 has requested advice
**If** A03 has received submitted customer information
**Then** A03
**Is obliged to** delivered standardized customer specification.

EN4
**Whenever** S02 has requested advice
**If** A04 has received information about the type of a customer need
**Then** A04
**Is obliged to** deliver a candidate-matches list.
EN5
Whenever S03 has requested subscription
If A06 has built a (relevant) company profile
Then A05
Is obliged to realize subscription.

EN6
Whenever S03 has requested subscription
If A06 has received submitted customer information
Then A06
Is obliged to build a company profile.

---

**Figure 7.14: The FM Business CoMponent (Advice View) – Structural aspect**

Based on the DEMO Coordination Structure Model and following the DEMO mapping mechanisms, we derive a model (Figure 7.15) representing the communicative view on the addressed business reality.
Figure 7.15: The FM Business Component (Advice View) – Communicative aspect

As seen from the figure, we have added elaboration (concerning the communicative aspect) by applying the Transaction pattern (Chapters 3 and 4) to each of the Transactions (Figure 7.14). This has been explained in the previous chapters. For this reason, we will not offer explanations on how we have derived the model based on the DEMO Coordination Structure Model.
As seen from Figure 7.15, two sub-processes are to be considered - one of them relates to the FM's delivering advice to a customer, the other one relates to the FM's realizing subscription to an (insurance) company. This is indicated by two starting points (on the figure): starting point 001 and starting point 002.

As also seen from the figure, the dependence of T01 on the execution of T02, the dependence of T02 on the executions of T03 and T04, and the dependence of T05 on the execution of T06 - all these are accordingly reflected in the model. Therefore, we have considered so far the 'Advice view' over the FM Business CoMponent as far as the structural and communicative aspects are concerned. We continue analogously towards the consideration of the 'Contracting view' over the CoMponent. We proceed analogously and will not offer as detailed explanations as in the previous paragraphs.

The built Coordination Structure DEMO model is depicted on Figure 7.16:

![Diagram](image)

**Figure 7.16: The FM Business CoMponent (Contracting View) – Structural aspect**

As seen from the figure, the depicted functionality of FM concerns also customers and insurance companies - S02 and S03. There are internal actors as well: A07 (Contractor) and A08 (Payment Controller).

As seen from the figure, A07 could offer (T07) to S02 a contract (in doing this, A07 is facilitated by a data-bank (DB04) containing contract templates that concern particular companies). This could be realized only based on an approval (T08) of such a contract, from the concerned (insurance company).
It could be seen as well from the figure that S03 could deliver (T09) a (insurance) financial product to S02. However, this could be done only based on a submitted (to the company) customer-FM agreement (T10) based on an offer acceptance (T11) by S02.

It could also be seen from the figure that in some situations, a (insurance) company should realize payment to FM. Actually that is about any realized (through FM) product delivery. FM should be notified about each realized (through it) product delivery. An indication for this is the dotted line between T09 and A08. A08 has (therefore) the information (it is stored in the data-bank DB05) about what each (registered) company owes to FM. A08 is activated by itself periodically. Then it is to handle the payments accordingly.

We go further (as we already did in the above paragraphs) for norm elaboration. We will not do this for transactions T12 and T13 because they are straightforwardly understandable. The derived norms are below:

EN7
Whenever S02 has requested contract
If S03 has approved a contract proposed by A07
Then A07
Is obliged to deliver the contract.

EN8
Whenever S02 has requested contract
If A07 has offered a contract not contradicting with the policy of S03
Then S03
Is obliged to approve the contract.

EN9
Whenever S02 has requested a financial product
If A07 has submitted an agreement (about the product) concerning S02
Then S03
Is obliged to deliver the financial product.

EN10
Whenever S02 has requested a financial product
If S02 has accepted a corresponding contract
Then A07
Is obliged to submit to S03 the appropriate agreement.

EN11
Whenever S02 has requested a financial product
If A07 has offered a contract which does not contradict with S02’s interests
Then S02
Is obliged to accept the contract.
Figure 7.17: The FM Business Component (Contracting View) – Communicative aspect

Based on the DEMO Coordination Structure Model (Figure 7.16), we derive a model (Figure 7.17) representing the communicative view on the addressed business reality.

As seen from the figure, three sub-processes are to be considered – one of them relates to the FM’s offering a contract to a customer, the second one relates to an insurance company’s delivering a financial product, and the third one relates to the FM’s payments handling. This is indicated by three starting points (on the figure): 003, 004, and 005.
As also seen from the figure, the dependence of T07 on the execution of T08, the dependence of T09 on the executions of T10, and the dependence of T10 on the execution of T11 – all these are accordingly reflected in the model.

Therefore, we have considered so far both ‘Advice’ and ‘Contracting’ views over the FM Business Com ponent as far as the structural and communicative aspects are concerned.

Regarding the structural validation of the com ponent, as stated before, there are not particular activities (according to the SDBC application guidelines) especially addressing such a validation. The reason is that we rely on the soundness of DEMO concerning the reflection of Transactions in a structural model.

We continue with consideration of the dynamic and data aspects.

Financial Mediator – dynamic and data aspects

As for the dynamic aspect, it is considered by reflecting the built (so far) models in appropriate dynamic (work-flow) ones. We will use Petri Net (PN) notations (plus norm elaboration that concerns the PN model).

Chapter 3 has introduced PN. Chapter 5 has motivated the use of this modeling tool within the SDBC context. Chapter 6 has provided guidelines on how to derive a PN model in SDBC. Thus, we will not explain in detail this derivation.

We will first build a model corresponding to the ‘Advice view’.

A basic source for building this dynamic model (which is represented on Figure 7.18) is the (already) constructed communicative one (Figure 7.15).

As seen from the Figure (7.18), the two sub-processes, considered within the communicative model, are reflected in the dynamic one (‘Start 1’/‘Start 2’ from Figure 7.18 correspond to starting points 001/002 from Figure 7.15). This is logical because such fundamental issues should not change depending on the particular aspect view.

As it is also seen from the figure, the Transactions (Figure 7.15) are reflected in corresponding activities. These are,

regarding the first sub-process:
- ‘FM: Generate standardized (stn.) specification’
- ‘FM: Generate candidate-matches’
- ‘FM: Perform match-making’
- ‘FM: Deliver advice’

and regarding the second sub-process:
- ‘FM: Build company profile’
- ‘FM: Realize subscription’

The activities ‘FM: Generate standardized (stn.) specification’ and ‘FM: Generate candidate-matches’ are modeled through the useful ‘parallel process’ PN mechanism, reflecting the requirement (Figure 7.15) that they both are completed before the activity ‘FM: Perform match-making’ could be realized.
We have reflected also (as depicted in Figure 7.18) some particularly important (from the perspective of the work-flow of events) communicative acts, following the SDBC application guidelines:

- 'Customer: Request advice' is a reflection of the 'request' part of the Transaction T01. This is necessary to be considered as an activity within the PN model because what actually needs to take place in triggering the flow of events is that a customer requests to receive advice from FM.
- 'FM: Process information' concerns also T01; handling an advice delivery should include consideration and processing of the customer information (to be accordingly distributed within FM). This has not been modeled as a separate Transaction because it concerns the 'information' level, not the 'essential' one. However, from the perspective of the flow of events it should be considered.

- 'FM: Realize data search' actually concerns the execution part of the Transaction T04. Again, because of its concerning the 'information' level, it is not considered as a separate transaction although it has to be considered within the modeled flow of events.

- The same applies to 'FM: Apply procedures'.

- 'FM: Apply criteria-related rules' concerns in the same way the execution of the Transaction T02.

- 'Ins. comp.: Request subscription' is a reflection of the 'request' part of the Transaction T05. This is necessary to be considered as an activity within the PN model because what actually needs to take place in triggering the flow of events is that a company requests to be subscribed to FM.

As for the activities 'FM: check customer's allowance' and 'FM: Check company's allowance' (Figure 7.18), they reflect a requirement from the business proposition, according to which: 'A customer's using FM (either for advice, or contract, or product delivery) is to be limited to no more than five times per month. As for (insurance) companies' allowance, a company is allowed to subscribe to FM only if it is licensed according to the Dutch financial laws'. These are actually informational (not essential issues) since they concern information checking. For this reason, they are not reflected in the structural DEMO model (and also the communicative one). Since they affect the flow of events, they are to be reflected in the dynamic model: the allowance of customer/company should be checked and if a customer/company happens to be not responding to the mentioned requirements then the customer/company should not be allowed to use the services of FM, hence a direct move to the 'end' point should take place.

As for the activities 'C/IC: Submit request specification / (detailed) information', and 'FM: Check submitted information', they concern informationally Transactions T01 and T05, respectively. Information aspects concerning these Transactions are not to be reflected at the essential level but have to be considered within the workflow of events. This is because the information providing (by a customer/company) is a key activity from a workflow perspective. This applies also for the check whether the information provided is sufficient (if not, the particular customer/company is to be asked to re-submit the information; the '*3' means that after 3 unsuccessful entries the user is kicked off – analogous indications are used also in 'result unsatisfactory' in the same figure).

As for the 'Contracting view', we have derived a model (Figure 7.19) in an analogous way.
As for the norm elaboration which is suggested (Chapter 6), we have derived several 'information' norms (attached to the PN models) consistent with the 'essential' norms (identified in the previous paragraphs). In their identification we include 'I' (from 'Informational'), 'N' (from 'Norm') and a number. The norm (below) is an example of such a norm, concerning the activity 'FM: Check submitted information' (we have assigned this activity a number, namely: number 4):

Figure 7.19: The FM Business CoMponent (Contracting View) – Dynamic aspect
IN4
Whenever a customer has requested advice
If (s)he has submitted information to FM
Then FM
Is obliged to check the submitted information.

And finally, regarding the validation of the derived dynamic models, we apply
discrete event simulation, as is according to the SDBC application guidelines. We will
not demonstrate it for the FM Business CoMponent, because we have already
demonstrated (in the previous chapter) the validation of a dynamic Activity
Diagram/PN model via discrete event simulation (using the tool ARENA).

As for the data aspect, it is considered by reflecting the models built so far in
appropriate data ones. We will use the ORM notations for this purpose.

Chapter 3 has introduced ORM. Chapter 5 has motivated the use of this
modeling tool within the SDBC context. Chapter 6 has provided guidelines on how to
elaborate factually models built applying SDBC.

We will build a model corresponding to the ‘Advice view’. We will not elaborate
factually the models which concern the ‘Contracting’ view mainly because the data-
banks related to them (DB05 and DB05 – Figure 7.16) are to be considered at a later
stage, when FM would have started to function. For example, information about
delivered (through FM) financial products (concerning DB05), based on which
(eventual) payments would be activated, would have appeared after FM has started
its operation.

Regarding the ‘Advise view’, we turn to the fundamental link between the
models (built so far) and the data aspect – these are the data-banks DB01, DB02,
and DB03 (Figure 7.14). Following the SDBC application guidelines, we reflect these
models into data models by further modeling these data-banks, relying on the
consistency of DEMO as concerns the structural-data modeling relations.

Therefore, our ‘Advise view’ (ORM-driven) data model of the FM Business
CoMponent should include elaborations of the data-banks DB01, DB02, and DB03.

Before proceeding to such an elaboration, we need to add some data input to
the business proposition information, which is as follows:

- We have selected for consideration the following seven (insurance) financial
  companies situated in The Netherlands: Icomp (situated in a Dutch city, offering
  products as follows: 1011001 (these codes will be explained further on)); OHRA
  (situated in Arnhem, offering products as follows: 0001010); AEGON (situated in Den
  Haag, offering products as follows: 11101111); Nationale-Nederlanden (situated in
  Rotterdam, offering products as follows: 1001110); Euro Lloyd Verzekeringen
  (situated in Amsterdam, offering products as follows: 0100100); Unive Verzekeringen
  (situated in Zwolle, offering products as follows: 11111110); AXA (situated in Utrecht,
  offering products as follows: 1101001). Details about these companies have been
  summarized at:
  http://www.sdbc.tk/comp/detialsicomp.htm
  http://www.sdbc.tk/ohra/detailsohra.htm
  http://www.sdbc.tk/aegon/detialsae.png.htm
  http://www.sdbc.tk/nn/detailsnn.htm
  http://www.sdbc.tk/ev/detailsev.htm
  http://www.sdbc.tk/uv/detailsuv.htm
  http://www.sdbc.tk/axa/detailsaxa.htm

157
As for the possible customer needs (to be addressed by FM), they might be: 'auto-insurance', 'health-insurance', 'life insurance', and so on. Procedures (to be considered concerning them) and their URLs are as follows:

auto-insurance Procedure 1 http://www.sdbc.tk/pr/pr1.htm
health-insurance Procedure 2 http://www.sdbc.tk/pr/pr2.htm
life-insurance Procedure 3 http://www.sdbc.tk/pr/pr3.htm

As for the criteria consideration (facilitated by procedures), which has already been mentioned, the following procedures are to be used, corresponding to the four criteria considered (pay-back, reliability, quality of service, insurance costs):

Pay-back Procedure PB
Reliability Procedure RB
Quality of Service Procedure QS
Insurance Costs Procedure IC

Regarding the product codes (used above already), we would like to make the following elaboration: We have considered seven types of (insurance) financial products, namely Life-insurance-related products, Property-insurance-related products, Mortgage-related products, Pension-related products, Travel-insurance-related products, Personal-damage-insurance-related products, and Lawyer-assistance-insurance-related products. We have given the following numbers to these types of products:

Life ins.: 1
Pr. Ins.: 2
Mortg.: 3
Pens.: 4
Trvl.: 5
PersDmg.: 6
LwrAsstInc.: 7

Then we introduce a string of seven binary digits. Each position there corresponds to the number of a particular type of product.

Thus, the code 0000100, for instance, should let us know that the particular company (to which this code is attached) offers only travel insurances and related (financial) products.

We have presented all this information in Figure 7.20 concerning the data aspect of the FM 'Advice view'.

The top model on the figure concerns the data-bank DB01 (Figure 7.14); the bottom model concerns DB03. The model between them concerns DB02.

As seen from the figure, we have consistently conducted data elaboration on the DEMO structural model (Figure 7.14) considering adequately the factual case information.

Therefore, we have considered so far both 'Advice' and 'Contracting' views. Regarding the 'Advice model', we have elaborated it in structural, communicative, dynamic, and data aspects. Regarding the 'Contracting model', we have elaborated it in structural, communicative, and dynamic aspects.

So, we have demonstrated Business CoMponents' elaboration. In the following sub-section, we will address the reflection of a Business CoMponent in the specification of software.
Figure 7.20: The FM Business Component (Advice View) – Data aspect
7.3.3 Towards Software Specification

In the previous sub-section, we have demonstrated the SDBC-driven elaboration of a Business CoMponent. As mentioned at the beginning of the current section: in this sub-section, we will demonstrate how a UML-driven software specification model could be derived on the basis of a Business CoMponent (in this particular case, on the basis of the coMponent considered in the previous sub-section). As already mentioned, this model should reflect the Business CoMponent. However, it is necessary also that the model considers the user-defined requirements towards the software system-to-be. Said otherwise, this model must have two inputs:

1. A Business Process modeling input coming through a Business CoMponent(s);
2. A requirements input coming through the specification of what the (future) users of the software system-to-be require as an automation.

We need, therefore, to add a requirements specification to the business proposition done in the previous sub-section:

According to the user requirements: The FM must be automated (completely), representing an ICT application which must be accessible via the Internet. The application should have mechanisms for checking the data accuracy, before performing match-making. And also, the application should be facilitated by a database (containing all the information from data-banks DB01, DB02, DB03, DB04, and DB05) located on a database server in The Netherlands.

Therefore, in going through the further (software specification) steps, following the SDBC application guidelines, we will consider both the input Business CoMponent (Sub-section 7.3.2) and the user-defined requirements (the above paragraph).

Use case derivation

As it has already been discussed, use cases are modeling constructs that serve to link the application domain (the business world) to the software domain, regarding any UML-driven software specification. Therefore (and according to the SDBC application guidelines), the first step in reflecting the FM Business CoMponent into a (UML-driven) software specification model must be a use case derivation.

According to the SDBC application guidelines (represented in Chapter 6), a use case derivation is to go through three phases, namely:

- derivation of essential use cases;
- derivation of informational use cases;
- derivation of UDR use cases (we remind that UDR stands for 'User-Defined Requirements').

As explained already, Essential use cases are pieces of functionality, reflecting actions from a considered Business System, which are 'essential'.

Informational use cases are pieces of functionality, reflecting actions from a considered Business System, which are 'informational'.

UDR use cases are pieces of functionality added on the basis of a consideration of the user-defined requirements towards the software system-to-be.

The SDBC use case derivation concerning these three types of use cases, is depicted on Figure 7.21 and will be followed further on.
Deriving Essential use cases

Following the SDBC application guidelines (Chapter 6), we derive the Essential use cases (Figure 7.22) by mapping them straightforwardly from corresponding DEMO Transactions (Figures 7.14 and 7.16). As for the UML use case diagram, the actors there reflect straightforwardly the 'external' DEMO actor roles (role-types). The reason is that we are to automate FM completely. Therefore the FM perspective is to coincide with the perspective of the software system-to-be.

Figure 7.22: FM: Use case model (identification of the Essential use cases)
Chapter 7

**Deriving Informational and UDR use cases**

Based on the identification of the Essential use cases and having as a source the dynamic (PN) models (Sub-section 7.3.2), where we have reflected the 'informational' issues related to the FM Business CoMponent, we identify the following use cases:
- 'check allowance'
- 'check submitted information'
- 'process information'
- 'apply search'
- 'apply procedures'
- 'apply rules'
- 'submit information'
- 'check contract availability',

all (of which) reflecting (straightforwardly) corresponding PN processes/transitions (Figures 7.18 and 7.19). Next to that, we have added the use case:
- 'request additional information',

as an extension to the use case 'check submitted information', since in some situations (when the submitted information is insufficient), it might be necessary that additional information is submitted.

We have also identified the following two UDR use cases reflecting the (above specified) user-defined requirements:
- 'check data accuracy'
- 'add data in database'.

Thus, the complete UML use case model is depicted in Figure 7.23 where, as seen, the Informational use cases are back-guided in grey and the UDR use cases are back-guided in black.

Regarding the use case diagram: There are two actors: Customer and (Insurance) Company. Concerning Customer (Company) he(it) takes the decision, has the responsibility, has the goal to have an advice/contract/product(fin.) delivered (its information correctly added to the FM database and have a subscription facilitating in this way the distribution of its (financial) products). The diagram contains 23 use cases: 'deliver advice', 'add data in database', and so on. There are 15 «include» relationships (one of them concerns the use cases 'deliver advice' and 'perform match-making', indicating that the FM's delivering an advice to Customer requires performing a match-making (based on which the advice would be specified)) as well as one «extends» relationship (in some cases, as mentioned above, if submitted information is insufficient, before continuing its operation further, FM would need the submission of some additional information, so the basic use case is 'check submitted information', and it is extended with 'request additional information').
**Case Study**

Figure 7.23: FM: Thorough use case model (identification of the Informational and UDR use cases)

**Elaboration**

Based on the built UML use case model, it is possible to make any further elaboration concerning either particular use cases (specifying them in more detail) or the model as a whole.

We will proceed (below) with demonstrating how any particular use case of interest could be adequately specified. We follow the use case specification mechanism suggested in the previous chapter, as a part of the SDBC application guidelines. Our work on this mechanism has been inspired by the use-case-related work of Cockburn [Cockburn 2000] and has come through a further research in the direction of use cases specification and elaboration [Shishkov and Dietz 2002-4]. We will not repeat again how we conduct the use case specification. This is explained in Chapter 6. Below we will just demonstrate the specification of a use case from the (already built) model (Figure 7.23).
We have selected, for illustrative purpose, the use case 'add data in database' and the mentioned investigation is applied to it – Figure 7.24 (only those extensions related to activity six are depicted).

The use case is written at 'system' scope (as opposed to 'enterprise' scope) since it describes an interaction with a computer system. The indicated 'summary' level means that the use case is long running (executed over months or years), showing the context in which the user goals operate.

![Image of Figure 7.24: Specification of the use case: 'add data in database']

For further (dynamic) elaboration (and visualization) of the considered use case ('add data in database'), a UML Activity diagram could be straightforwardly derived based on the main success scenario + extensions (Figure 7.24). As seen from this figure, there are nine core activities (complemented with extensions), in the mentioned use case. Some of them are shown on Figure 7.25, as an overall UML Activity diagram.

![Image of Figure 7.25: UML Activity Diagram for the use case: 'add data in database']

As studied in Chapter 6, for still further (dynamic) elicitation and visualization as well as validation, it is straightforward to proceed (from the Activity diagram above) to computer simulation. We will not demonstrate this because it has been demonstrated in the precious chapter.
As mentioned above, one might need to elaborate either particular use cases (specifying them in more detail) or the model (Figure 7.23) as a whole. We demonstrated the first (above). As for the elaboration of the model as a whole, it might be dynamic or structural.

Regarding the dynamic elaboration of the model as a whole, it could be conducted by reflecting the dynamic Business Process models (Figures 7.18 and 7.19) into UML Activity diagram(s) [Eshuis and Wieringa 2003]. This is straightforward and will not be demonstrated below (it has been demonstrated in Chapter 6). However, in realizing such a mapping, one should add accordingly information connected to the user-defined requirements because this information is not reflected (of course) in the dynamic Business Process models.

![Partial UML Class Diagram](image)

**Figure 7.26: FM: Partial UML Class Diagram**

As for the structural elaboration of the model as a whole, it could be conducted by reflecting both the structural DEMO Business Process models (Figures 7.14 and 7.16) and the Use case model (Figure 7.23) into UML Class diagram(s). We will show below only a partial UML Class diagram (concerning just the use cases ‘realize subscription’ and ‘build profile’; we reflect also the two types of profile, as from the initial case information). We have discussed examples also in the previous chapter. The UML Class diagram is depicted in Figure 7.26.

### 7.4 CONCLUDING REMARKS

In this chapter, we have demonstrated the strengths of SDBC by illustrating through the ‘Icomp case’: the conceptual framework of the approach (presented in Chapter 4) and the guidelines for its application (introduced in Chapter 6). The application of SDBC has followed the selection of tools (suggested in Chapter 5).

This has helped us validate the essential contributions of the current research (it should be noted that besides this case, we have considered a number of smaller scale case examples as well [Shishkov and Dietz 2005-2; Shishkov 2004] including some within the previous chapters). However, we have not relied solely on case study research as a way to validate our results; we have brought evidence of the adequacy of some modeling views/considerations/steps as well as of the logical transformations towards further steps.

The following chapter is going to summarize the results of this research and propose an agenda for its continuation.
Chapter 8

CONCLUSIONS

The only source of knowledge is experience.
Albert Einstein

The current thesis delivers a research output related to the alignment between Business Process modeling and software specification. This output has evolved within the context of the research background formulated in Chapter 1. This research background consists of identified research problems, formulated research goals and questions, and adopted research approach.

The mentioned alignment between Business Process modeling and software specification is a weak point in most of the currently used Software Design methods. As discussed already, this relates to the great percentage of failures observed in software projects to date. There are numerous examples of mismatch between stated user requirements and the actual functionality of delivered software applications.

Addressing the above-mentioned problem is an actual issue with regard to current Business Systems, given the observed significant (and further increasing) complexity of these systems. It could be seen not only from the trends towards globalization (which often forces companies to link their (business) processes) but also from the frequent updates and actualizations in the products of companies, resulting from the increasing market demands and the severe competition. The mentioned increasing business complexity leads to the need of powerful software support to Business Systems.

Since the current level of software application support to Business Systems seems inadequate in this regard (this is to be seen from the mentioned examples of software project failures), it might be concluded that better (than the existing) software specification methods are required. An additional indication of this need could be seen in the fact that more and more research takes place on application development methodologies.

The above-mentioned problem could be adequately reflected in new application development methodologies only if precise methodological guidelines are proposed on how to soundly map between Business Process modeling and software specification. Otherwise (as it is currently observed), either purely business issues would be considered through a software perspective or Business Process modeling
outputs would be superficially translated into software concepts. In both cases, a sound business-software alignment could not be reached.

This research problem has been fundamentally approached through two research perspectives, namely the Component-Based (system) Development perspective (CBD) and the Language–Action Perspective (LAP). These perspectives have proved to be useful and applicable with respect to the problem.

CBD is an approach for building systems, which is based on the Object-Orientation (OO) paradigm. CBD allows for systems specification based on re-usable, replaceable, and easily modifiable components. It has been studied in the current research that CBD is applicable concerning both business and software issues. Therefore, identifying business components and reflecting them in derived software components would represent a well founded alignment between Business Process modeling and software specification, which is put on sound theoretical foundations.

LAP is a well-established theoretical orientation towards approaching the modeling of business processes by emphasizing the importance of interaction and communication. The theory recognizes that language is not only used for exchanging information, as in reports or statements, but that language is used also to perform actions, as in promises, orders, requests, and declarations. Such actions are the foundation of communities and organizations, and must be understood to create effective Information Systems. Hence, adequately capturing the communication aspects characterizing the considered Business System(s) would contribute to the creation of adequate Business Process models.

Therefore, considering these two mentioned perspectives (CBD and LAP) would allow for a better grasp of the essential issues characterizing the Business Processes under study as well as for a sound alignment between Business Process modeling and software specification.

Hence, the essential goal of the current research has been defined as investigation of the (LAP-driven) identification of (re-usable) business components and their use for specifying ICT applications which effectively support Business Processes. The stated research goal motivated the posed research questions which are discussed further on in this chapter.

Chapters two and three gave elicitation on the definitions, concepts, and some existing tools (and their strengths as well as limitations), relevant to the research.

As for the definitions, among the essential ones (proposed in the thesis) are the Business Component/Componen and Software Component/Componen definitions (Chapter 2). They are of fundamental importance for aligning (in a component-based way) Business Process modeling and software specification, because of their direct relation to the possibility to conceptually support the derivation of Software Components.

Regarding the existing tools relevant to this research, the focus was only on such ones concerning (directly or indirectly) both Business Process modeling and software specification. All popular tools (methods) that relate to either of these two aspects only, have not been included in the study scope. In particular, Tropos, Catalysis, and KobA have been addressed, as three of the most well-founded and popular research tools meeting the above stated demand. However, based on an investigation of the mentioned tools, it has been concluded that neither of them fully meets the demand for a sound and complete component-based alignment between Business Process modeling and software specification. Next to that, we have studied powerful Business Process modeling tools, such as DEMO, Semantic/Norm Analysis, Petri Net, ORM, and simulation tools, finding out that left alone, neither of them fully solves the desired task. The same applies to some relevant formal tools concerning
functional and/or data analysis. All this has justified our proposing a new software specification approach, namely SDBC (SDBC stands, as stated, for Software Derived from Business Components).

Chapters 4, 5, and 6 have subsequently: 1) introduced the mentioned approach (in particular, its conceptual framework) which allows for the specification of software on the basis of identified (re-usable) Business Components; 2) studied the usefulness of existing modeling tools with regard to the application of SDBC; 3) proposed step-by-step methodological guidelines concerning that application. And finally, Chapter 7 has validated the above-mentioned research achievements, exploring the applicability of SDBC in a large Dutch insurance company.

Hence the research questions in this thesis have been answered by proposing a software specification approach and suggesting methodological guidelines for its application, and also – by studying and further developing existing relevant definitions/concepts. The answers to the (research) questions have been justified by demonstrating how the SDBC approach could be applied in a real-life case.

Further on in this chapter, there follow three sections in which: the research questions are considered again in the light of their achieved answers, commenting as well on some limitations of the current research; the SDBC approach is briefly summarized, as an essential output of the research; an outlook for further research is proposed.

8.1 THE RESEARCH QUESTIONS

The current thesis has brought convincing answers to the research questions formulated in Chapter 1. This section returns to the questions and discusses briefly the achieved answers.

**Question 1** What benefits can the CBD way of thinking bring to the modeling of business systems?

The Introduction has briefly discussed the undisputable advantages of Object-Oriented (OO) and its related CBD paradigm, concerning the design and development of ICT applications. As stated there, by basing application building on encapsulated, individually definable, re-usable, replaceable, interoperable and testable (software) components, developers could create applications which possess durable configuration and a high degree of flexibility and maintainability. As stated also, some researchers and practitioners (such as Jacobson, for example) claim that analogous advantages could be brought about if the CBD way of thinking is applied to other systems, for instance Business Systems. Inspired by these claims, we made assertions (Chapter 1), posing the first research hypothesis (Figure 1.1), concerning some benefits that CBD could bring to Business Process modeling, among which adequate delimitation and re-use possibilities. These assertions have been further elaborated in Chapter 2, by introducing and explaining the Business Component/Component concepts. The research achievements of this thesis (both propositional (Chapters 2, 4, 5, and 6) and validation (Chapter 7), including the SDBC approach) are to some extent in support of the mentioned claims.

As studied in this thesis, by applying the CBD way of thinking, a modeler could identify particular Business Subsystems, facilitating in this way the task of providing a relevantly delimited Business Process modeling output for the specification of the

168
software system-to-be. Identifying a Business Subsystem that comprises exactly one Business Process, could bring particular re-use values. Such a Business Subsystem is called 'Business Component'. Models of such subsystems could be, as it has been studied in the previous chapters, useful with regard to the conceptual support to be realized in specifying an ICT application. Thus, CBD brings benefits to the modeling of Business Systems mainly in two directions – the direction of re-using models of parts of such systems for modeling purposes and the direction of producing a Business Process modeling output adequately usable for the specification of software. This has been justified within the current thesis, supporting the claim within Hypothesis 1 (Chapter 1).

However, we do not find it appropriate just reflecting straightforwardly all software-related CBD concepts and views to Business Process engineering. As it could be seen from the above explanations (and from the content of the current thesis in general), we have adopted only some particular CBD views, claiming their relevance not only to Software Design but also to Business Process engineering.

**Question 2**  
**What must be understood by 'business component'?**

This question has been answered through the study conducted in Chapter 2. It has explored as well the **Business Process** concept, studying also the considered elementary building blocks with regard to business processes, namely Transactions. The **Business System** definition (which is based on the classical system definition of Bunge – Chapter 2) is of significant importance in this regard. According to Chapter 2, 'a system should be considered being a **Business System** if and only if it is composed of humans collaborating among each other through actions which are driven by the goal of delivering business products to entities belonging to the environment of the system'.

As for Transactions, according to Chapter 2: 'A **Transaction** is a finite sequence of coordination acts between two actors, concerning the same production fact. The actor who starts the Transaction is called the **initiator**. The general objective of the initiator of a Transaction is to have something done by the other actor, who therefore is called the **executor**. As also studied, Transactions are related to each other in a tree-structure. The top of the tree is called the **starting Transaction**. It is a Transaction that is not caused directly by another Transaction (from the particular tree) but triggers the execution of other Transactions (within the tree). According to Chapter 2, 'a **Business Process** is a structure of (connected) Transactions that are executed in order to fulfill a starting Transaction'.

On the basis of the above-considered concepts, the **Business Component** concept (Chapter 2) could be introduced: 'a **Business Component** is a business subsystem that comprises exactly one Business Process'.

However, due to the fact that usually (and also according to Chapter 2) Software Components are viewed as implemented pieces of software, the Business Component concept and the Software Component one seem hardly alignable – Business Components are Business Subsystems while Software Components are implemented pieces of software. Hence, led by the goal of supporting conceptually (through business process concepts) the software specification process, in Chapter 2 we have introduced two essential concepts: **Business Component** and **Software Component**. These concepts represent, respectively, models of a Business Component and a Software Component. Thus, the **Business Component** concept is an essential complement to the **Business Component** one within the current research.
According to Chapter 2: ‘a Business CoMponent is a model of a Business Component, which is elaborated in at least four perspectives, namely structural, dynamic, factual, and communicative’. These perspectives have been explained and discussed within the thesis, following Hypothesis 2 (Figure 1.2) whose claim has also been supported by the research analysis and propositions.

Thus, we view a Business Component as a Business Sub-system and we make use of this concept via a modeling one, namely the Business CoMponent concept.

What we have not realized in this research is to offer mechanisms allowing modelers to distinguish among different complexity levels characterizing Business Components/CoMponents. It might be that Business Component A comprises a Business Process consisting of several transactions and Business Component B comprises a Business Process consisting of more than a hundred transactions, for instance. Therefore, treating both components in the same way is probably not most appropriate.

**Question 3** How can (re-usable) business components fill the gap between business process modeling and software specification?

Chapter 2 as well as the chapters concerning SDBC (Chapters 4, 5, and 6) are relevant to the answer of this essential research question.

It relates to the misconception that a Business System is a kind of Information System. Instead, as explained in Chapter 2, they are systems in different categories: social and rational, respectively. Thus, in order to fill the gap between Business Process modeling and software specification, one would need to consider two types of concepts, namely Business Engineering and Software Design concepts; one should also consistently map between these two types of concepts.

We have embraced this way of thinking in the current thesis, introducing the SDBC approach. It has been shown that the approach allows for a component-based alignment between Business Process modeling and software specification, where Business Engineering concepts are soundly reflected in Software Design ones. How SDBC achieves this, is briefly discussed in the following section of this chapter.

Conceptually, as explained in Chapter 4, we realize this via the Business Component/CoMponent and Software Component/CoMponent concepts. A Business CoMponent reflects a part of a Business System, comprising a particular Business Process. Hence, it is a purely Business Process Engineering concept. Using Business CoMponents, we support the derivation of Software CoMponents (models of Software Components, as defined in Chapter 2). This support comes through an adequate reflection of the structural, dynamic, factual, and communicative business issues in the software functionality under specification. Next to that, the particular requirements are considered.

Regarding the application of the SDBC conceptual framework, we have proposed methodological guidelines (Chapter 6) on how to realize the mentioned reflection, proposing particular modeling tools, as studied in Chapter 5: DEMO, Norm Analysis, Petri Net and so on, from the Business Process modeling side, and use cases, UML Class diagram, UML Activity diagram, UML State diagram, and so on, from the software specification side.

Therefore, we have addressed the gap between Business Process modeling and software specification on both conceptual and modeling levels, and have proposed an approach allowing for filling this gap. Using the SDBC approach, one could take a pure business description and a requirements specification, and adequately derive
(out of such an input) a software specification model which is consistent with the current software design standards. Re-usable Business Components/CoMponents play an essential role in this derivation because they are a proper reflection of the original business reality and are therefore adequately usable as a software specification input.

An issue which has not been sufficiently elaborated in our way of alignment between Business Process modeling and software specification is the consideration of the initial user information and its transformation into relevant Business Process modeling concepts. We have demonstrated in several cases and examples how we succeed in structuring the initial information. However, in real-life problems, it is possible that this information is incomplete and full of errors. We partially address this by the structured way in which we construct our Business Process models. Anyway, more rigorous rules regarding this could add value.

**Question 4**  
*How can business components be re-used for the purpose of specifying different software applications?*

As it is well-known, re-use is a factor of utmost importance for any modeling method. Hence, this question is of essential importance.

SDBC approaches the re-use issue (with regard to the software specification task) through the consideration of three particular re-use levels – 1) re-use of Software Components; 2) re-use of Business/Software CoMponents; 3) re-use of Business Processes models.

As far as the re-use of Business/Software CoMponents is concerned, this has been addressed in Chapter 4. It introduced conceptually two types of re-usable building blocks, namely *general* (capturing some core functionality and therefore needing extension) and *generic* (having embedded in them several (optional) functionalities, with the possibility to choose among them through adjustment (parameterization)) ones. This has been reflected in the conceptual framework of SDHC, as a modeling mechanism. Therefore, within the SDHC approach, Business/Software CoMponents can be re-used through identifying general and/or generic Business/Software CoMponents, and extending and/or parameterizing them further on.

Although the current thesis provides guidelines on the re-use of Business/Software CoMponents, the questions concerning the decision when is it most appropriate to consider general or generic CoMponent (or not consider a re-usable CoMponent at all) need more in-depth studies. This would additionally facilitate modelers in their decisions concerning the use of such re-usable modeling patterns.

**Question 5**  
*How can identified business components be validated in terms of structure and dynamics?*

This question has been answered in Chapters 4, 5, and 6. Chapter 4 elicited the conceptual validation ideas behind SDHC. Chapter 5 proposed relevant tools to be used for validation within SDHC (such as Activity diagram and ARENA). Chapter 6 suggested particular steps for realizing this.

Within SDHC, we do not consider any separate modeling activities concerning the structural validation. The reason is that SDHC offers a straightforward mechanism for deriving a DEMO Coordination Structure Model (CSM) and also for reflecting it
into a UML one. The building itself of a DEMO CSM includes validation. Hence, concerning the validation of the structure of developed models, SDBC relies on the soundness of the employed modeling tools, mostly DEMO and UML.

As for the dynamic validation, according to SDBC, it is to be conducted via discrete event simulation. It has been studied and demonstrated (Chapter 6) how a dynamic (DEMO-driven) Business Process model could be mapped towards a UML model (Activity diagram, in particular) and reflected further (straightforwardly) in ARENA notations. The resulting ARENA discrete event simulation model not only represents the original 'dynamic' information in sound graphical (and animation facilitated) notations but also allows for an in-depth analysis empowered by the possibility to make real-time changes in the model and observe their effect.

A definite shortcoming of SDBC is however its not considering formal validation techniques which are more convincing than the semi-formal ones simply because of their formality. Consideration of some currently popular (for example BDI-related) model checking techniques [Bordini et al. 2003] could add value in this respect.

**Question 6**

Which of the existing modeling environments and tools could be employed for effectively aligning business process modeling and software specification in a component-based way?

This question has thoroughly been answered in Chapter 5. It has concluded about particular modeling tools and environments, including UML and DEMO, being suitable with regard to goal of making SDBC operational. Chapter 6 has shown further on how modelers could employ these particular tools in the application of SDBC. The proposed selection of tools complementing each other, includes: DEMO (Coordination Structure Model and Business Process Model), Norm Analysis, UML Use case Diagram, UML Activity Diagram (and other UML diagrams as well), ARENA Discrete event simulation tool, and also other tools.

A shortcoming in selecting these tools (and also in specifying the SDBC application guidelines) is the lack of direct reference to and consideration (in doing this) of the OMG Model-Driven Architecture (MDA). MDA [MDA 2001] is fully consistent with UML and is currently widely applied within the software community for component-based software specification. A direct reference to MDA would add value concerning the potentials for combining SDBC with other methods and/or tools.

### 8.2 SDBC

The SDBC approach is an essential part of the output of the current research. For this reason, we will briefly summarize the outline of SDBC, with the help of Figure 8.1. Concerning the used abbreviations, we note that there is a difference with the ones used in Figures 4.18 and 4.19 where bc stands for Business Com ponent; in contrast, on Figure 8.1, bk stands for Business Com ponent; we make this change because now we depict both Business Components and Business Com ponents.
Abbreviations:
bc – Business Component
bk – Business Component
gbck – General Business Component
gbck – Generic Business Component
ssm – Software specification model
sc – Software Component
sk – Software Component

Figure 8.1: SDBC - outline

As seen from the figure, we consider a Business System from which a Business Component(s) is to be identified, using the SCI chart and other related modeling tools (Chapters 6 and 7). The component should be then reflected in a relevant model – a Business Component. Another way for arriving at a Business Component is by applying re-use: either extending a general Business Component or parameterizing a generic one. DEMO and other related modeling tools are relevant as far as Business Components are concerned. The Business Component should then be elaborated with the domain-imposed requirements, in order to add elicitation on the particular context in which its corresponding Business Component exist within the Business System. Then, a mapping towards a software specification model should take place (Chapters 4, 6, and 7), driven by a DEMO-UML transformation (characterized mainly by the derivation of use cases and Activity diagram). The mentioned modeling deliverables as well as the user-defined requirements are to be both considered here, since the derived software model should reflect not only the original business features but also the particular requirements towards the software system-to-be. The (UML-based) software specification model needs a precise elaboration so that it provides sufficient elicitation in terms of structure, dynamics, data, and communicative aspects. It needs also to be decomposed into a number of Software Components reflecting functionality pieces. These Components then are to
undergo realization and implementation, being reflected in this way in a set of Software Components. Some Software Components could also be purchased (in this case, of course, their specification does not result from the application of the SDBC approach). The Software Components are implemented using Software Component technologies, such as .NET or EJB, for instance. Finally, the (resulting) component-based ICT application would support informationally the target Business System, by automating anything that concerns the considered Business Component (identified from the mentioned system).

Within the thesis, the proposed SDBC approach (outlined in this way) has been elicited conceptually (Chapter 4), supported by a selection of relevant modeling tools (Chapter 5) and application guidelines (Chapter 6).

8.3 OUTLOOK FOR FURTHER RESEARCH

There are perspectives for further study on Business Systems and the creation of software for their support. This could be seen from the great number of projects currently focusing on this, including projects which bring new relevant research perspectives and horizons, for example the 'Multi-Agent' [Woolridge 2002] and 'Context-awareness' [AWARENESS] perspectives. This reaffirms the actuality of the current research and motivates further activities directed towards overcoming some shortcomings of its, developing in this way the achieved research output. With respect to this, the following research agenda is proposed:

- Further work is necessary on making SDBC sufficiently consistent with the latest software standards. The achieved consistency with UML is the necessary background for this. However, SDBC needs to be clearly positioned also in the context of some widely accepted modeling facilities (considered as de facto architectural standards within the software community) basically associated with the Model-Driven Architecture (MDA) and the Open Distributed Processing architecture (ODP). Such a MDA/ODP positioning of SDBC would make it easier combinable with other existing software-related tools.

- Study of the appropriateness of formal model checking techniques to be applied within SDBC; such a model validation, incorporated in SDBC, would be more convincing for potential users of the approach because of its formality.

- Extending the SDBC approach in the direction of Multi-Agent Systems, including further consideration of the ‘Business Component/CoMponent’ concepts and their alignment to suitable ‘agent’-related ones. This would allow SDBC to be applicable in solving more complex problems.

- The application of SDBC in solving real-life problems should be further considered – more case studies should be conducted in order to examine thoroughly the particular advantages and disadvantages of the approach in the light of its practical application.

- A bank of re-usable general/generic Business CoMponents concerning a particular business domain could be created. This would allow for a number of small-scale case studies (which involve application of SDBC) to be conducted in the domain, justifying further the re-use values of SDBC and eventually revealing issues needing improvement.
Bibliography


Bibliography


[DEMO] Design and Engineering Methodology for Organizations (DEMO), http://www.demo.nl


[ORM] Object Role Mmodeling (ORM), http://www.orm.net


Bibliography


Bibliography


184


[Wi-Fi] Wireless Fidelity (Wi-Fi), http://www.wi-fi.org
Bibliography


De huidige Informatie- en Communicatie-technologie (ICT) is van grote betekenis voor een goed verloop van processen behorend bij verscheidene business domeinen. Software-applicaties moeten het gebruik van ICT voor het genoemde doel faciliteren. Daarom vindt meer en meer onderzoek plaats naar applicatie-ontwikkelingsmethodologieën, en ook meer en meer industriële projecten blijken hiermee gerelateerd. Niettemin worden zulke projecten vaak gekarakteriseerd door niet-behaalde doelen, lage gebruikerstevredenheid en budgetoverschrijdingen.

Het is vastgesteld dat een veel voorkomende oorzaak van het mislukken van een softwareproject de slechte afstemming is tussen de eisen die de business stelt en de daadwerkelijke functionaliteit van de geleverde applicatie. Dit probleem houdt verband met het misverstand dat een business-systeem een soort informatiesysteem is. In plaats daarvan zijn het systemen in verschillende categorieën, respectievelijk sociaal en rationeel. Daarom, om tot een afspiegeling van de benodigdheden in het de toekomstige software te komen, dient men duidelijk de vorming van het business proces, en de software specificaties te onderscheiden, het louter business georiënteerde model brengen naar de specificaties van het software systeem. Om ervoor te zorgen dat business eisen goed zijn afgestemd met het te ontwikkelen softwaresysteem, is het daarom noodzakelijk om het modelleren van bedrijfprocessen af te stemmen met het specificeren van software, waarbij een zuiver bedrijfsgespecialiseerd model wordt afgebeeld op de specificatie van een softwaresysteem. Het realiseren van een dergelijke afstemming op een componentgebaseerde manier lijkt haalbaar en nuttig, omdat op componenten gebaseerde business en software modellen hergebruik en goede ‘modeling trace-ability’ toelaten, en toepasbaar en flexibel onderhoud vergemakkelijken. Daarnaast zou een grondige afbeelding een goede basis zijn voor business software afstemming, omdat een business component zich verhoudt tot businessproces vormende concepten, terwijl een software component zich zou verhouden tot software concepten; derhalve. Daarnaast zou een grondige afbeelding een goede basis zijn voor business software afstemming. Immers, een business component reflecteert concepten voor bedrijfprocesmodellering, terwijl een softwarecomponent juist softwareconcepten reflecteert. Echter, tot nu toe is het component paradigma alleen echt doorgedrongen tot de implementatie en voorbereidende fasen van de software levenscyclus, en speelt het nog geen rol van betekenis in de vroege analyse en ontwerp activiteiten van grote software projecten. In de software context worden componenten hoofdzakelijk geassocieerd met de huidige ‘fysieke’ component technologieën (bijv. .NET, CORBA, EJB).

De SDBC benadering zal gepresenteerd worden (SDBC staat voor ‘Software Derived from Business Components’), die zich richt op dit thema door ‘logische’ componenten, die de logische bouwstenen vertegenwoordigen van een software systeem, te beschouwen. Vanuit deze positie stelt SDBC een mechanisme voor ten behoeve van de business-software afstemming. In het bijzonder maakt deze benadering het mogelijk dat businessproces modellen (Business CoMponents) worden afgeleid en weergegeven in conceptuele (UML gestuurde) software specificatie modellen (Software CoMponents). Bij de identificatie van Business CoMponenten volgt de SDBC een multi-aspect business perspective dat volledigheid garandeert. Bij de Business CoMponent – Software
Component afbeelding volgt de SDBC grondige regels om een adequate afstemming te garanderen. Omdat SDBC UML bestuurd wordt is SBDC afgestemd op de huidige standaarden voor software-ontwikkeling. De toepassing van SDBC wordt op dit moment onderzocht bij een groot Nederlands verzekeringssbedrijf.
Acknowledgements

I spent inspiring time working on my thesis. Now I look back with gratitude to those who supported me.

First of all, I am indebted to my promotor Jan Dietz who gave me the opportunity to carry out this PhD project and who supported me in all those years, being dedicated, tolerant, and understanding. I am grateful to him for his excellent supervision. I am particularly grateful for his criticism and his never ending demands for sharpness in presenting scientific views and ideas. I thank him for his willingness to spend many evenings and weekends on reviewing chapters of my thesis. I deeply respect our scientific collaboration which was inspiring for me, stimulating me to improve as researcher.

I owe special thanks to Henk Lof for supporting my work on the Icomp Case. Without his help the case would have been impossible. I appreciated his competence and knowledgeable as well as his devotion and enthusiasm which inspired me tremendously to progress in my research. Thanks, Henk! I do appreciate your contribution so much.

Thanks are due to the members of my Graduation Committee: Joaquim Filipe, Kecheng Liu, Roel Wieringa, Waltraud Gerhardt, and Henk Koppelaar. Thank you for investing time to read my work and for providing me with support and stimulating feedback. Your suggestions contributed greatly to the final improvements of my thesis. I especially thank Kecheng Liu also for inviting me to work in his group in Stafford and Reading. I spent exciting time there, because of his warm welcome and great collaboration. I am grateful as well to Joaquim Filipe for the interesting discussions we have had during the last several years as well as for his friendship and supportiveness concerning my work. I would like to thank Henk Koppelaar for welcoming me to Breda several times, where he was kind to devote time and enthusiasm for discussions on my thesis as well as on my future research plans.

I thank Ilian Ilkov for his friendship and support. He was always willing to share his knowledge and provide me with help in my first steps as researcher. Next to that, he was the one who has helped me more than anybody else in establishing my life in The Netherlands. And last but not least, I liked so much our talks on many subjects, which we used to have on the 8th floor of the EWI/TUD building as a distraction from the hard work.

I would like to thank Alexander Verbraeck, Val Jones, Gaspard Boot, Joseph Barjis, and Pascal van Eck for supporting in different ways my work on the thesis.

I would like to thank Marten van Sinderen for the opportunity to join his team in Twente. This was not only a chance for me to continue my academic development but also an inspiring challenge, given the interesting, exciting and dynamic project in which I am currently involved. I am privileged to work under his guidance, benefiting from his critical, sharp, and supportive feedback. I leave each of our Wednesday meetings full of enthusiasm. All this makes my stay in Enschede so pleasant. This has definitely contributed to the inspiration I needed when submitting the final manuscript of my thesis.

I thank my dear Iliana for her support and encouragement in difficult times during my PhD research.

I owe gratitude to my loved parents, especially to my father who was my source of inspiration when I decided to start an academic career. In general, I am thankful to them for teaching me which are the real values in life and also for their unconditional help and support during hard times. I thank my mother for drawing, for the first time in my life, my attention on computer systems. I thank her as well for making the beautiful design for the cover of the thesis.

Boris Shishkov
Enschede, August 2005
About the author

Boris Shishkov was born in Sofia, Bulgaria on the 22\textsuperscript{nd} of January, 1973. In 1992 he graduated from Sofia's prestigious '114\textsuperscript{th} High School'. Afterwards he studied Management Information Systems at Sofia University St. Kliment Ohridski and received his M.Sc. diploma in 1999, graduating with distinction. In the course of his studies, Boris Shishkov gained an ERASMUS EU scholarship and studied at Aristotle University of Thessaloniki, Greece in the period January - June, 1997.

He began his research on Software Engineering at Delft University of Technology in the year 2000. His research has resulted in more than 30 publications in books, journals, and conference proceedings. Next to that, Boris Shishkov established and developed successfully research collaboration with the group of Professor Kecheng Liu (Information Systems) at the University of Reading, United Kingdom; he realized successful research visits there, being awarded two times Honorary Research Associateship. Boris Shishkov has also established a useful collaboration concerning an EU-funded research project hosted by the Bulgarian Academy of Science. He has also taken organization responsibilities including the organization of a workshop on the DEMO methodology - The First DEMO Workshop, July 3, 2003, Tilburg, The Netherlands. The work of Boris Shishkov in Delft has been additionally inspired by contacts with Dutch companies, such as Weast Ven Man B.V.

Since 2005 he is working as a researcher at the Computer Science Department of the University of Twente, focusing on context-aware applications, mobile services, and service infrastructures.
SIKS Dissertatiereeks

1998-1, Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2, Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3, Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4, Dennis Breuker (UM)
Memory versus Search in Games

1998-5, E.W. Oskamp (RUL)
Computerondersteuning bij Straftoemeting

1999-1, Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2, Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3, Don Beal (UM)
The Nature of Minimax Search

1999-4, Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5, Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6, Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7, David Spelt (UT)
Verification support for object database design

1999-8, Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation

2000-1, Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2, Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3, Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief

2000-4, Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5, Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval

2000-6, Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7, Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8, Veerle Coup (EUR)
Sensitivity Analysis of Decision-Theoretic Networks

2000-9, Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10, Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture

2000-11, Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

2001-1, Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2, Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models
2001-3, Maarten van Someren (UvA)
Learning as problem solving
2001-4, Evgeni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets
2001-5, Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style
2001-6, Martijn van Welie (VU)
Task-based User Interface Design
2001-7, Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization
2001-8, Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.
2001-9, Pieter Jan 't Hoorn (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes
2001-10, Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice
BRAHMS: a multiagent modeling and simulation language for work practice analysis and design
2001-11, Tom M. van Engers (VUA)
Knowledge Management: The Role of Mental Models in Business Systems Design
2002-01, Nico Lassing (VU)
Architecture-Level Modifiability Analysis
2002-02, Roelof van Zwol (UT)
Modelling and searching web-based document collections
2002-03, Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval
2002-04, Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining
2002-05, Rada Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents
2002-06, Laurens Mommers (UL)
Applied legal epistemology: Building a knowledge-based ontology of the legal domain
2002-07, Peter Boncz (CIW)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications
2002-08, Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas
2002-09, Willem-Jan van den Heuvel (KUB)
Integrating Modern Business Applications with Objectified Legacy Systems
2002-10, Brian Sheppard (UM)
Towards Perfect Play of Scrabble
2002-11, Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications
2002-12, Albrecht Schmidt (Uva)
Processing XML in Database Systems
2002-13, Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications
2002-14, Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems
2002-15, Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling
2002-16, Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications
2002-17, Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance
2003-01, Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments
2003-02, Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems
2003-03, Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy
2003-04, Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology
2003-05, Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach
2003-06, Boris van Schooten (UT)
Development and specification of virtual environments
2003-07, Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks
2003-08, Yongping Ran (UM)
Repair Based Scheduling
2003-09, Rens Kortmann (UM)
The resolution of visually guided behaviour
2003-10, Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture
2003-11, Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks
2003-12, Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval
2003-13, Jeroen Donkers (UM)
Nosce Hoscem - Searching with Opponent Models
2003-14, Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations
2003-15, Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems
2003-16, Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses
2003-17, David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing
2003-18, Levente Kocsi (UM)
Learning Search Decisions
2004-01, Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic
2004-02, Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business
2004-03, Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving
2004-04, Chris van Aart (UvA)
Organizational Principles for Multi-Agent Architectures
2004-05, Viara Popova (EUR)
Knowledge discovery and monotonicity
2004-06, Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques
2004-07, Elise Bottjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes
2004-08, Joop Verbeek (UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiën gegevensuitwisseling en digitale expertise
2004-09, Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning
2004-10, Suzanne Kabel (UvA)
Knowledge-rich indexing of learning-objects
2004-11, Michel Klein (VU)
Change Management for Distributed Ontologies
2004-12, The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents
2004-13, Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play
2004-14, Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium
2004-15, Arno Knobbe (UU)
Multi-Relational Data Mining
2004-16, Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning
2004-17, Mark Winands (UM)
Informed Search in Complex Games
2004-18, Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models
2004-19, Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval
2004-20, Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams
2005-01, Floor Verdenius (UvA)
Methodological Aspects of Designing Induction-Based Applications
2005-02, Erik van der Werf (UM))
AI techniques for the game of Go
2005-03, Franc Groote (RUN)
A Pragmatic Approach to the Conceptualisation of Language
2005-04, Nirvana Meratnia (UT)
Towards Database Support for Moving Object data
2005-05, Gabriel Infante-Lopez (UvA)
Two-Level Probabilistic Grammars for Natural Language Parsing
2005-06, Pieter Spronck (UM)
Adaptive Game AI
2005-07, Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems
2005-08, Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications
2005-09, Jean Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages
2005-10, Anders Bouwer (UvA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments
2005-11, Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search
2005-12, Csaba Boer (EUR)
Distributed Simulation in Industry
2005-13, Fred Hamburg (UL)
Een Computertmodel voor het Ondersteunen van Euthanasiebeslissingen
2005-14, Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics
2005-15, Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes
2005-16, Joris Grauernans (UJJ)
Usability of XML Query Languages
2005-17, Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components