Software architecture decision-making in organizational settings

by

Daniel Gross

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Faculty of Information
University of Toronto

© Copyright by Daniel Gross, 2011
Software architecture decision-making in organizational settings

Daniel Gross

Doctor of Philosophy

Faculty of Information
University of Toronto

2011

Abstract

The purpose of the architecture of software systems in business organizations is to support those organizations in achieving business goals. In software development organizations the design of an architecture is a collective effort that involves various organizational stakeholders and designers, who identify, interpret, and reason about intents, and communicate, delegate, commit to, and implement intents and decisions. Current architectural design descriptions are by-and-large based on block-and-arrow notations representing "coarse-grained" solution elements of the system. They lack explicit representation for modeling and analyzing the decision-making of stakeholders and architectural designers who hold different organizational responsibilities, and pursue conflicting and/or synergistic business or system goals, while collectively pursuing organizational objectives. This thesis considers the proposition that a distributed intentionality perspective is applicable in the design of software system architectures. During architectural design, relationships between intentional actors define the context in which intentional actors pursue business and system goals and in which they negotiate architectural decision-making. The objective of this research is to investigate what an Intentional Architecture Language (IAL) could be like that utilizes intentional and organizational modeling and analysis concepts to support architectural decision-making efforts in organizational settings. Drawing from prior work on organizational modeling and analysis, this thesis first defines a core IAL, and then explores its use to
model and analyze architectural decision-making both reported in the literature and empirically observed at a number of commercial projects in industry. Drawing from these explorations, this thesis proposes a number of extensions to the core IAL, discusses lessons learned, and points to the advantages and limitations in using an IAL to model and analyze architectural decision-making in an organizational setting.
Acknowledgments

On a bright morning about a decade ago during a spontaneous visit to Toronto, I called on a whim the computer science department at the University of Toronto, to ask about the courses given at graduate level. I had already taken some half a dozen months off from working, while traveling for the first time in my life in North America. During random visits to different cities, wherever I found accommodation amongst friends or friends of friends, etc., I made it a habit to look up interesting researchers at local universities to learn about their work and enjoy an intellectual enriching discussion – adding “academic tourism” to cultural tourism.

I had not heard of the University of Toronto before, and, in fact, I had until my visit not known much about Toronto or Canada. However, here I was in Toronto, and there was a university in town, and I had nothing planned for the day.

The graduate department secretary read to me over the phone the syllabus, eventually mentioning a Requirements Engineering course. Hearing about an RE course immediately arouse my interest. Just a few months before while visiting San Francisco I had first learned of this emerging discipline. I happened to get an invitation to participate in the Viewpoint ‘96 conference workshop held in SF, which was attended by, among others, Anthony Finkelstein (the organizer), Axel van Lamsweerde, Naftali Minsky, Jeff Kramer, and Sol Greenspan, who, unbeknown to me, were all distinguished researchers in the emerging RE field. I had left San Francisco with much interest in this subject, and with an invitation in hand to apply to the doctoral program in RE at the City University in London, UK.

Therefore, hearing about an RE course, a course quite uncommon in those early days of RE, held the promise of a highly interesting discussion, and I asked to put my call through to the professor who offered the course. Professor Mylopoulos immediately took the call and without ado agreed to meet me for an informal chat at his office. I had not heard of Prof. Mylopoulos before and was not aware of his eminent stature in the RE research community.

We met and I had a fascinating discussion during which Prof. Mylopoulos also warmly recommended the RE program at City University. At the end of our meeting, Prof. Mylopoulos suggested talking to a former student of his, Eric Yu, who was now Professor at the Faculty of Information Studies. Following the suggestion, I met with Prof. Yu the next day. We had in intriguing discussion of overlapping areas of interest as well as of Prof. Yu’s work on intentional agents.

A day or two later I left for London to apply to the doctoral program at the City University. However, my discussions with Prof. Mylopoulos and Prof. Yu had left a very strong impression on me, and about a
week later, I emailed Prof. Yu asking how he felt if I would apply to the doctoral program at University of Toronto, with him as my thesis advisor. Prof. Yu agreed, and with the application deadline a week later, I swiftly couriered my application package to Toronto. Several months later, I was admitted to the doctoral program at the faculty of information studies.

I would thus first like to thank Prof. Mylopoulos for taking that telephone call on that bright sunny morning, and despite his busy schedule, for meeting with an anonymous visitor to the city, who happened to have an emerging interest in RE. I also want to thank Prof. Mylopoulos for his ongoing interest and support of this work and for serving on my final doctoral examination committee.

I would like to thank Prof. Anthony Finkelstein, who, together with the other excellent presenters at Viewpoints ’96, opened to me new viewpoints and perspectives (no pun intended) on requirements modeling and analysis, which set me on the path towards earning a doctorate in this exciting field.

I am much indebted to the academic and technical staff at the Faculty of Information (formerly the Faculty of Information Studies) for their ongoing support, in particular, to the late Prof. Ethel Auster, former chair of doctoral studies committee and to the recently retired Assistant Dean, Judy Dunn. Their doors were always open to me to discuss professional and personal problems that naturally emerge during such a complex and long termed endeavor; their attitude was always that obstacles are there to be overcome, and problems to be solved.

I want to thank Prof. Chun Wei Choo, who served on my doctoral and doctoral examination committee. His suggestions and support, in particular during the early stages of my research, were invaluable. I also want to thank Prof. Steve Easterbrook and Prof. Kelly Lyons for serving on my doctoral examination committee.

I would like to thank the many colleagues, with whom I had the pleasure to interact with during this journey. To name just a few: Niloo Hodjati, Lin Liu, Luiz Marcio Cysneiro, Markus Strohmaier, Jennifer Horkoff, Daniel Amyot, Gunter Mussbacher, Tom Gray, Serge Mankovskii, Xiping Song, Sharon Volk and Sharon Al-Al.

There are no words to express my respect and gratitude to Prof. Eric Yu, who during all these years actively and unwaveringly supported my doctoral research work, both from a professional and personal point of view. Without his immense patience and commitment, and his continuous support and guidance, this work would never have come to fruition.

I would also like to thank my parents Shoshana and Shimon Gross who after surviving the war in Europe, started from scratch, raised five children in the most difficult personal and financial circumstances. With their immense hard work, they enthusiastically supported each of their five
children in pursuing their individual aspirations, in law, in teaching, in systemic family therapy, in
business, and in software engineering.

Finally, to my wife Meira Josephy; without her support, in the good and the difficult times, in
particular when life threw us those challenging curve balls, this thesis research would have been
abandoned a long time ago. To cite the words of the Talmudic sage Rabbi Akiva, who, because of his
wife’s magnanimous support and sacrifices, was able to spend a significant amount of time to become a
scholar and teacher (Babylonian Talmud, 62B): “all that is mine and yours [his students], is hers”. 
Table of Contents

ABSTRACT......................................................................................................................................................... II

ACKNOWLEDGMENTS ......................................................................................................................................... IV

TABLE OF CONTENTS ........................................................................................................................................ VII

LIST OF TABLES.................................................................................................................................................. XI

LIST OF FIGURES................................................................................................................................................ XII

1. INTRODUCTION.............................................................................................................................................. 1
   1.1. MOTIVATION .................................................................................................................................................. 1
   1.2. RESEARCH OBJECTIVES .................................................................................................................................. 4
   1.3. RESEARCH APPROACH ...................................................................................................................................... 6
   1.4. RESEARCH CONTRIBUTION AND LIMITATIONS........................................................................................... 7
       1.4.1. Research Contribution Overview ............................................................................................................... 7
       1.4.2. Study limitations .......................................................................................................................................... 13
   1.5. CONCEPTUAL FRAMEWORK ............................................................................................................................ 13
   1.6. CASE STUDIES OVERVIEW ............................................................................................................................. 17
   1.7. EXPLORATION OF INTENTIONAL ACTOR CONCEPT ..................................................................................... 20

2. RELATED WORK.............................................................................................................................................. 22
   2.1. ARCHITECTURE AND ORGANIZATION ............................................................................................................ 22
   2.2. DESIGN RATIONALES AND DESIGN PATTERNS ............................................................................................... 26
   2.3. REQUIREMENTS TRACEABILITY ........................................................................................................................ 27
   2.4. REQUIREMENTS NEGOTIATIONS ......................................................................................................................... 27
   2.5. SOFTWARE ARCHITECTURE AND DECISION-MAKING ................................................................................... 27
   2.6. SOFTWARE SYSTEM MODULARIZATION .......................................................................................................... 28
   2.7. ACTOR AND GOAL-ORIENTED MODELING APPROACHES ........................................................................... 29
   2.8. CONCLUSION..................................................................................................................................................... 31

3. A PROTO-INTENTIONAL ARCHITECTURE LANGUAGE (IAL)........................................................................ 32
   3.1. REQUIREMENTS FOR THE LANGUAGE ............................................................................................................. 32
   3.2. OVERVIEW OF THE LANGUAGE ........................................................................................................................ 33
   3.3. COMPARING IAL TO APPROACHES APPLYING i* TO THE ARCHITECTURE DESIGN .................................... 35
   3.4. CORE LANGUAGE META-MODEL AND EXTENSIONS ...................................................................................... 39
3.4.1. Core IAL constructs ........................................................................................................ 39
3.4.2. Extensions to Deal with Intentional Collectives .......................................................... 48
3.4.3. Extensions to Deal with Architectural Artifacts ......................................................... 51
3.4.4. Extensions to support Representing and Applying Generalized knowledge .......... 55
3.4.5. Extensions to Reason with Intentional Viewpoints .................................................... 59
3.4.6. Extensions to Deal with Packaged Solutions ............................................................ 61
3.4.7. Extensions to Deal with Integrating Qualitative and Quantitative NFR Reasoning .... 62
3.4.8. Extensions to Deal with Terminology Ambiguity ....................................................... 70
3.5. Conclusion ...................................................................................................................... 72

4. IAL APPLIED TO THE KWIC SYSTEM LITERATURE CASE STUDY ........................................ 74

4.1. Introduction ..................................................................................................................... 74
4.2. Description of Case Study ............................................................................................. 75
4.3. Placing Architectural Design Discussions into an Organizational Setting .................... 77
4.4. Responding to Research Questions ............................................................................. 82
4.5. Modeling Discussion ..................................................................................................... 86
4.6. Conclusion ..................................................................................................................... 87

5. IAL APPLIED TO THE “WML” INDUSTRIAL CASE STUDY ............................................... 88

5.1. Introduction ..................................................................................................................... 88
5.2. Description of Case Study ............................................................................................. 89
5.3. Placing Architectural Design Discussions into an Organizational Setting .................... 92
5.3.1. Representing Stakeholders and Designers Who Are the Source of Architectural Design Goals .... 92
5.3.2. Representing and Analyzing Architectural Design Solutions and Tradeoffs .......... 96
5.3.3. Evolving Architectural Responsibilities Linked to Architectural Decision-making ...... 98
5.3.4. From Generalized to Concrete Organizational Architectures .................................... 105
5.4. Responding to Research Questions ............................................................................. 107
5.5. Modeling Discussion ..................................................................................................... 116
5.6. Conclusion ..................................................................................................................... 117

6. IAL APPLIED TO THE “PHOENIX” INDUSTRIAL CASE STUDY ............................................ 119

6.1. Introduction ..................................................................................................................... 119
6.2. Description of Case Study ............................................................................................. 120
6.3. Representing and Analyzing Design Reasoning from Contrasting Points of View .......... 123
6.4. Identifying Higher-Level Stakeholders and Decision Makers ........................................ 126
6.5. Analyzing Viewpoint Reasoning in Relation to Higher-level Goals, Prioritizations, and Tradeoff-Making .. 128
6.6. RESPONDING TO RESEARCH QUESTIONS ................................................................. 131
6.7. MODELING DISCUSSION .......................................................................................... 134
6.8. CONCLUSION ........................................................................................................... 136

7. IAL APPLIED TO THE “TELELARM AB” DESIGN PATTERN LITERATURE CASE STUDY ........... 137

7.1. INTRODUCTION .................................................................................................... 137
7.2. DESIGN PATTERNS .............................................................................................. 138
7.3. DESCRIPTION OF CASE STUDY ......................................................................... 139
7.4. UNDERSTANDING, ANALYZING, AND APPLYING THE DEVIATION DESIGN PATTERN ........................................................................................................... 140
7.5. UNDERSTANDING, ANALYZING, AND APPLYING THE POINT PATTERN ................. 148
7.6. RESPONDING TO RESEARCH QUESTIONS ............................................................ 152
7.7. MODELING DISCUSSION ....................................................................................... 154
7.8. CONCLUSION ........................................................................................................ 155

8. IAL APPLIED TO THE MODELING AND ANALYSIS OF QUALITATIVE AND QUANTITATIVE NFRS DURING AN INDUSTRIAL PROJECT ........................................................................ 156

8.1. INTRODUCTION .................................................................................................... 156
8.2. DESCRIPTION OF CASE STUDY ......................................................................... 157
  8.2.1. Difficulty in specifying quantitative NFRs for an application platform .................. 157
  8.2.2. Industrial control systems .................................................................................. 159
  8.2.3. NFRs of Platform, Platform Application, and Domain Application .................... 161
  8.2.4. Analyzing Processing Throughput and Processing Capacity ................................ 164
8.3. GOAL-ORIENTED ANALYSIS TECHNIQUE ......................................................... 165
8.4. SCENARIO-ORIENTED ANALYSIS TECHNIQUE ................................................. 168
8.5. ANALYZING SERVICE ORIENTATION PROCESSING OVERHEAD ......................... 172
8.6. DERIVING PLATFORM NFRS FROM NFRS OF EXISTING CONTROL SYSTEMS ............ 176
8.7. RESPONDING TO RESEARCH QUESTIONS ............................................................ 179
8.8. MODELING DISCUSSION ....................................................................................... 183
8.9. CONCLUSION ........................................................................................................ 184

9. IAL APPLIED TO DEALING WITH TERMINOLOGICAL AMBIGUITY DURING ARCHITECTURAL DESIGN .... 186

9.1. INTRODUCTION .................................................................................................... 186
9.2. WHY A SEMIOTIC TERMINOLOGY ANALYSIS? ..................................................... 187
9.3. AFFORDANCE AND SEMIOTIC AGENT MODELING OF TERMS ................................. 191
9.4. SEMIOTIC AGENT ANALYSIS OF POINT-TO-POINT INTEGRATION ......................... 192
9.5. A FULLER SEMIOTIC AGENT ANALYSIS OF DISCUSSED TERMS ......................... 194
9.6. RELATING SEMIOTIC AGENT MODELING TO INTENTIONAL AGENT AND GOAL MODELING ........................................198
9.7. CONCLUSIONS ..................................................................................................................................................200

10. CONCLUSIONS ...............................................................................................................................................201
10.1. SUMMARY OF FINDINGS ..............................................................................................................................201
10.2. SUMMARY OF CONTRIBUTIONS ..................................................................................................................211
10.3. FUTURE DIRECTIONS ..................................................................................................................................213

APPENDIX A. ADDITIONAL CASE STUDIES .................................................................................................219

APPENDIX B: NOTATION LEGEND ..................................................................................................................222

11. REFERENCES ..................................................................................................................................................226
List of Tables

Table 1-1: First set of research questions
Table 1-2: Second set of research questions
Table 1-3: Case studies, focus and publications
Table 3-1: Comparing IAL and two actor-oriented approaches applied to software architecture
Table 3-2: Semiotic composition structure to define operational term
Table 3-1: KWIC Comparison of Solutions Approaches (Garlan & Shaw, 1994)
Table 8-1: Processing capacity formula elements for single invocations of processing steps
Table 8-2: Processing capacity formulas for single invocations of processing steps with web service overhead
Table 8-3: Deployment and loading conditions
Table 8-4: Total processing capacity needs -- ranking
Table 8-5: Comparing WS- and non-WS-enabled capacity needs
List of Figures

Figure 2-1: Layers of organizational decision-making behavior (B. Curtis et al., 1988) ........................................... 23
Figure 2-2: Architectural Business Cycle (Bass, et al., 1998) ............................................................................. 24
Figure 3-1: Core meta-model. Actors and Actor Relationships ............................................................................. 40
Figure 3-2: Core meta-model – Intentional elements to support goal directed reasoning .................................. 41
Figure 3-3: Core meta-model - softgoal structure (dotted links indicate subset constraints) ............................. 44
Figure 3-4: Partitioning of dependum between depender and dependee .............................................................. 45
Figure 3-5: Partitioning Intentional Element into two Intentional Element concepts ........................................... 46
Figure 3-6: Intentional collective extension ....................................................................................................... 50
Figure 3-7: Linking to architectural artifacts .................................................................................................. 52
Figure 3-8: Responsibility Assignment ........................................................................................................... 54
Figure 3-9: Allocating intentional roles (example 1) .......................................................................................... 55
Figure 3-10: Allocating Intentional Roles (example 2) ..................................................................................... 56
Figure 3-11: Allocating intentional roles (example 3) ........................................................................................ 57
Figure 3-12: Allocating intentional roles meta model ........................................................................................ 58
Figure 3-13: Meta-model fragments to represent intentional viewpoints ......................................................... 60
Figure 3-14: Packaged solutions meta-model fragment ..................................................................................... 61
Figure 3-15: Specifying quantitative NFRs ........................................................................................................ 63
Figure 3-16: Representing quantitative NFRs .................................................................................................... 64
Figure 3-17: A sample scenario model ............................................................................................................... 66
Figure 3-18: Scenario template associated with a design task ........................................................................... 67
Figure 3-19: Linking scenario models to goal graph models ............................................................................. 68
Figure 3-20: IDEF0 diagram of Analysis method steps overview ....................................................................... 69
Figure 3-21: Semiotic agent meta-model ......................................................................................................... 71
Figure 4-1: Goal graph to support selecting among solution alternatives (Chung, et al., 1995) ....................... 76
Figure 4-2: Organizational context for an e-catalog application ......................................................................... 78
Figure 4-3: Organizational context for advancing a reuse requirement ............................................................. 80
Figure 4-4: Dealing with conflicting requirements ............................................................................................ 81
Figure 5-1: A telephone system architecture .................................................................................................... 90
Figure 5-2: WML/WAP reference architecture ................................................................................................. 91
Figure 5-3: Where to place the WML browser component ................................................................................ 92
Figure 5-4: Source of non-functional design goals, alternative modeling representation .................. 94
Figure 5-5: WML team, a broader organizational context ................................................................. 95
Figure 5-6: Strategic rationale graph representing alternative architectural choices ....................... 97
Figure 5-7: Establishing organizational responsibilities ..................................................................... 99
Figure 5-8: Creating new organizational responsibilities for stateless shared controller .................. 100
Figure 5-9: Creating new organizational responsibilities for stateless shared controller (continued) ... 101
Figure 5-10: Creating new organizational responsibilities for stateful shared controller ................. 102
Figure 5-11: Creating new organizational responsibilities for master controlled WML infrastructure ... 103
Figure 5-12: distributed design reasoning ......................................................................................... 104
Figure 5-13: Generalized shared device organizational architecture ................................................ 105
Figure 5-14: Instantiating generic organizational architecture ......................................................... 106
Figure 6-1: Intentional Viewpoint of the Consumer designer ............................................................. 122
Figure 6-2: intentional viewpoint of the SOA architect ................................................................. 124
Figure 6-3: Side-by-side comparison of viewpoint, in context of higher-level stakeholders ............. 125
Figure 6-4: Goals originating from higher-level stakeholders and designers ................................. 127
Figure 6-5: Higher-level prioritization ............................................................................................. 129
Figure 7-1: Overview of alarm system (Molin & Ohlsson, 1998) ................................................... 139
Figure 7-2: Solution structures included in a control node ............................................................. 140
Figure 7-3: Deviation pattern, first design approach reasoning ....................................................... 141
Figure 7-4: Deviation pattern second design approach and design reasoning ............................... 142
Figure 7-5: Deviation pattern, third design solution and reasoning ............................................... 143
Figure 7-6: Deviation pattern approaches and reasoning structures ............................................... 144
Figure 7-7: Functional and non-functional elaboration of alarm system processes ......................... 145
Figure 7-8: Applying deviation pattern during design ....................................................................... 146
Figure 7-9: Deviation pattern included in alarm system solution structure ..................................... 147
Figure 7-10: Using abstract base class to access I/O devices .......................................................... 149
Figure 7-11: Point pattern reasoning structure .................................................................................. 150
Figure 7-12: Applying the Point Pattern .......................................................................................... 151
Figure 7-13: System solution structure after applying the Point Pattern ....................................... 152
Figure 8-1: Context diagram of control systems .............................................................................. 159
Figure 8-2: Anatomy of a typical control system ............................................................................. 161
Figure 8-3: Defining NFR specifications, an overview .................................................................... 162
Figure 8-4: Qualitative analysis of throughput requirements .......................................................... 166
Figure 8-5: NFR reasoning linked to quantitative processing overhead estimation ...................... 168
Figure 8-6: Abstract processing model of an industrial controller ................................................ 170
Figure 8-7: NFR goal graph fragment to analyze interoperability and related NFR tradeoffs, showing use of Web Services versus Direct Processing ........................................................................ 171
Figure 8-8: Processing model annotated with web services indicators ........................................ 173
Figure 8-9: Expansion of WS annotation into more detailed structures and processes ............... 176
Figure 9-1: High level service-oriented architecture in business enterprises ............................ 187
Figure 9-2: Semiotic analysis of point-to-point integration style .................................................. 193
Figure 9-3: Fuller semiotic agent analysis .................................................................................. 195
Figure 9-4: Revisiting hardgoal and design task elements based on semiotic terminology analysis .... 199
“The talent myth assumes that people make organizations smart.
More often than not, it's the other way around.”

(Gladwell, 2009)

1. Introduction

1.1. Motivation

The architecture of a software system in business organizations is inherently intentional. It is designed with specific purposes in mind, most specifically, to support the organization in achieving current and future business goals. Today, it is widely accepted that the architecture of a software system is a key determinant of whether, how, and how well a system will meet its design intents (Bass, Clements, & Kazman, 1998; Barry W. Boehm, 1978; P. Clements & Bass, 2010; Engels et al., 2008; M. Shaw & Clements, 2006; O. Zimmermann, Gschwind, Küster, Leymann, & Schuster, 2008).

In development organizations, both intents and decision-making to achieve those intents are distributed. Architectural design to meet business goals is a collective effort that involves various organizational stakeholders and designers, who identify, interpret, and reason about intents, and communicate, delegate, commit to, and implement intents and decisions (Conway, 1968; Grinter, 1999; Herbsleb & Grinter, 1999a, 1999b). For example, upper management evaluates current and future client needs and decides on the strategic directions of the development organization. Product managers interpret and translate these to product features. Enterprise architects ensure that current and future business goals can be met at reasonable development, maintenance and evolution costs, while individual system and component developers are called upon to implement architectural decisions in software systems.

Current architectural design descriptions are, by-and-large, non-intentional; they are mostly based on block-and-arrow notations or representation languages representing “coarse-grained” solution elements of a system (Bass et al., 1998; Buhr & Casselman, 1996; P. C. Clements, 1996; Garlan, Monroe, & Wile, 2000; Perry & Wolf, 1992; M. Shaw & Clements, 2006; Mary Shaw & Garlan, 1996). They lack explicit representation of goals, alternative design approaches and tradeoffs, and of the architectural and architectural-relevant (P. Clements & Bass, 2010) decision-making that participants put forward to collectively address goals. Consequently, non-intentional descriptions lack the ability to support the modeling and analysis of architectural decision-making in its organizational setting where various
organizational participants individually and collective work towards meeting business and system goals.

The need to support dealing with intents and decision-making in organizational settings was already observed during early empirical studies of software development in organizations in the 1980s and 1990s. These showed that the organizational context is part of, and influences, architectural decision-making. For example, the distribution of knowledge across designers, design and managerial levels, and across organizations; the managing of “hand-off” of design work; and the coordination of decision-making between organizational participants greatly influence the design decisions that architects make (B. Curtis, Krasner, & Iscoe, 1988; W. Curtis, Krasner, Shen, & Iscoe, 1987; Grinter, 1999; Herbsleb & Grinter, 1999b).

Some researchers, recognizing the limitations of non-intentional descriptions, proposed to include intents and decision-making in the descriptions of the architecture, thus linking architectural solution approaches to their software engineering properties, and in particular to their non-functional properties (Amyot & Mussbacher, 2003; Bachmann, Bass, Klein, & Shelton, 2005; Buschmann, 1996; Chung, Nixon, & Yu, 1995, 1997; R Kazman, Klein, & Clements, 2000; Perry & Wolf, 1992). Some researchers have also focused on more systematically deriving architectural structures and processes from non-functional requirements, usually in conjunction with preexisting architectural knowledge (Beck & Johnson, 1994; Bosch, 1999; Buschmann, 1996; Chung et al., 1995; Niemelä & Immonen, 2007; Tyree & Akerman, 2005). Other works have recognized the need to view architectural design and decision-making as addressing business goals of the enterprise organizations (Bass et al., 1998; Engels et al., 2008; R Kazman & Bass, 2005; Narman, Johnson, & Nordstrom, 2007). More recently, researchers have proposed several decision-oriented modeling and analysis approaches (Jansen & Bosch, 2005; Zannier, Chiasson, & Maurer, 2007; O. Zimmermann, Koehler, Leymann, Polley, & Schuster, 2009). In these works architectural decisions are first-class modeling entities which link architectural rationales and justifications to architecture solution artifacts (Tang, Han, & Vasa, 2009).

However, in a recent keynote, Philippe Kruchten, a noted researcher and practitioner in the software architecture domain, while pointing to some of the aforementioned approaches, acknowledged that despite much interest and research in software architecture, much architectural knowledge, including architectural decision knowledge, is still tacit in organizations. It is still a challenge to provide the right information at the right time to the right person, including non-architects, in the organization. Kruchten concludes that there is a need for approaches that go beyond mere documentation that make available architectural knowledge in a manner relevant to different organizational participants, and that support the reasoning of the architects and developers (P. Kruchten, Babar, & Gorton, 2010).
Thus, while the introduction of intents and decision-making into the modeling notation offers important advancements over block-and-arrow notations and description languages, these do not go far enough when the need arises to deal with the needs and interests of the organizational participants who pursue intents and are involved in decision-making. Omitting such an organizational context in models and analyses limits the utility of these approaches to support dealing with decision-making of different stakeholders and designers in the organization, each of whom is responsible for different parts or aspects of the business or the system’s architecture, while collectively aiming to achieve the organization’s objectives.

There is need for an approach that supports the modeling and analysis of decision-making of stakeholders and architectural designers, who hold different organizational responsibilities, and pursue conflicting and/or synergistic business or system goals. In other words, there is a need for supporting the representation and analysis of architectural decision-making as a distributed decision-making process of intentional decision-makers in organizational settings (Rasmussen, Brehmer, & Leplat, 1990; Scacchi, 1984).

In the requirements engineering domain, Yu has been investigating the need to deal with distributed intents and decision-making during the early requirements analysis processes of organizational information systems (E. Yu, 1994; 2009). To support a requirements analysis of organizational information systems in their organizational setting Yu proposed an organizational modeling and analysis framework based on a distributed intentional actor concept and paradigm. In the same spirit, this thesis considers the proposition that a distributed intentionality perspective is also applicable in the design of software system architectures. The need for supporting a distributed intentionality perspective is based on the central premise that the activity of creating architecture in an organizational setting is inherently intentional and socio-organizational: organizational relationships between organizational participants define the context in which organizational participants pursue business and system goals and make business and architectural decisions.

The objective of this research is to investigate what an Intentional Architecture Language (IAL) could be like using Yu’s intentional actor modeling and analysis approach, developed in the requirements engineering domain, as a starting point. Such an IAL would offer a toolbox of concepts to support the modeling and analysis of the intentions underlying architectural decision-making, as well as the modeling and analysis of the distribution of those intents and decision-making in an organizational setting.

More specifically, this thesis reports on the modeling and analysis of several cases of architectural
decision-making using Yu’s intentional actor modeling approach as the starting point. Cases were drawn from both architectural decision-making reported in the literature and from data collected about architectural decision-making at industrial projects. During these case studies, the use of organizational modeling and analysis was explored to model and analyze the intentional and the organizational dimension of architectural decision-making, and suggestions of adaptations and extensions to the modeling notation, were made, where it was found necessary, to support situation-specific modeling and analysis needs.

The emphasis of this thesis research is to thus to explore the intentional and organizational dimension of architectural decision-making using an existing intentional actor modeling approach developed in the requirements engineering domain. In the course of this exploration this thesis also identifies limitations of the existing modeling approach when applied to architectural decision-making and, where deemed necessary, proposes extensions to the language to dealing with additional modeling and analysis requirements derived from the case studies in the architectural design domain.

1.2. Research Objectives

The overall objective of this research is to offer a perspective on intentional and socio-organizational modeling and analysis of architectural decision-making in an organizational setting. This objective was pursued by asking a set of research questions (see tables 1-1, 1-2) based on the following premises:

Architectural design as distributed decision-making: Architectural design often occurs as distributed decision-making in organizations, and as part of broader organizational decision-making processes. During such distributed decision-making processes different organizational players, occupying different responsibilities, often at different levels in the organization, collectively address the organizational objectives.

Business goals as ultimate purpose for architectural design: For organizations to succeed in achieving their goals both in the short term and the longer term, they require supporting technological capabilities that can grow and change with evolving needs of the business. A carefully designed software system architecture is a key enabler for such technological change capabilities, while supporting the ongoing operational business needs (Engels et al., 2008).

Non-functional requirements (NFRs) as linking between business goals and architectural design: Software architectures are specifically designed to exhibit non-functional properties, which in turn support achieving current and future business goals (P. Clements & Bass, 2010). NFRs thus not only play a crucial role in determining the architecture, but also mediate between business goals and architectural
Distinguishing between quantitative and qualitative NFRs: Typically, NFRs are treated either quantitatively or qualitatively. In a quantitative treatment, NFRs are specified using numbers and metrics, such as the requirement to process 30,000 changes of input values per second. Quantitative NFRs are typically either fully achieved or not at all. For example, the system either supports the aforementioned processing of 30,000 changes of input values or not.

NFRs, that are difficult or even impossible to quantify, are usually treated in a qualitative manner. For example, NFRs such as Security, and Modifiability are usually not quantified but are specified and dealt with qualitatively. During architectural design, such qualitative NFRs are also not achievable absolutely, but only in a “good enough” manner. For example, a system is said to have sufficiently, or insufficiently, addressed security, rather than absolutely achieved security. Due to their different nature during architectural design, qualitative NFRs and quantitative NFRs are considered and dealt with differently.

The following main research questions (table 1-1, table 1-2) were addressed during each case study.

<table>
<thead>
<tr>
<th>Table 1-1: First set of research questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1.1. What role do business goals play during architectural decision-making?</td>
</tr>
<tr>
<td>Q1.2. What role do qualitative non-functional requirements (NFRs) play during architectural decision-making to meet business goals?</td>
</tr>
<tr>
<td>Q1.3. What role does the organizational setting play during architectural decision-making to meet business goals?</td>
</tr>
<tr>
<td>Q1.4. What role do quantitative NFRs play during architectural decision-making to meet business goals?</td>
</tr>
</tbody>
</table>

The main research questions included in table 1-1 can be elaborated as follows.

**Q1.1.** When architects determine the software system architecture, how are business goals taken into account? For example, do architects enquire about the business goals that the architecture aims to support? Are architectural choices justified by referring to business goals? Are there multiple sources in the organization for business goals? Do architects and stakeholders make a deliberate effort to trade-off competing business goals?

**Q1.2.** How do architects deal with qualitative NFRs during architectural design? Do architects and stakeholders specifically define business goals and architectural design goals as qualitative NFRs? Is
there pre-existing knowledge about how to address qualitative NFRs such as pre-existing design principle and practices that can be used?

Q1.3. Considering that architecture is a collective design effort that often involves one or more teams, what organizational considerations contribute to architectural design? For example, how does the organizational setting in which the architecture and its various parts are constructed influence architectural decisions?

Q1.4. How do architects deal with quantitative NFRs during architectural decision-making? How are they traded-off, for example, with NFRs that are not quantifiable?

The overall thrust of these first four research questions is to help identify and assess the kind of modeling and analysis constructs that could be useful to assist architects in their work. Table 1-2 includes questions about appropriate modeling constructs, and corresponds to each of the research questions in table 1-1.

Table 1-2: Second set of research questions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2.1.</td>
<td>What modeling constructs are appropriate for making business goals and the role they play during architectural decision-making explicit?</td>
</tr>
<tr>
<td>Q2.2.</td>
<td>What modeling constructs are appropriate for making qualitative NFRs and the role they play during architectural decision-making to meet business goals explicit?</td>
</tr>
<tr>
<td>Q2.3.</td>
<td>What modeling constructs are appropriate for making the organizational setting and the role the organization setting plays during architectural decision-making to meet business goals explicit?</td>
</tr>
<tr>
<td>Q2.4.</td>
<td>What modeling constructs are appropriate for making quantitative NFRs and the role they play during architectural decision-making to meet business goals explicit?</td>
</tr>
</tbody>
</table>

1.3. Research Approach

The research approach used during this thesis research was exploratory and based on conceptual modeling (Mylopoulos, Jurisica, & Yu, 1998) which was performed during several single case studies (Kitchenham, Pickard, & Pfleeger, 1995; Yin, 2009). Cases of architectural decision-making were drawn both from the literature and from industrial projects at research partner organizations. For each of the cases studied architectural decision-making was modeled and analyzed.

Since the focus of this research is on architectural and architecturally relevant decision-making in an
organizational setting, this research adopts as a strawman\(^1\) intentional architecture language (IAL) the i* modeling language (E. Yu, 1994; S. E. Yu, 2009), which embodies the distributed intentionality principles proposed by Yu’s intentional actor paradigm (E. Yu, 1994, 2001b). The IAL is then used to explore the utility of intentional modeling and analysis of architectural decision-making in an organizational setting. To deal with the qualitative nature of design argumentation during architectural design, IAL also adopts concepts from Chung’s NFR Framework (Chung, 1993). More specifically, this research addresses the research questions through a number of case studies by:

a) defining a core intentional architectural language (IAL), drawing concepts from i* and the NFR framework, and

b) using this core language to explore the intentional and socio-organizational perspective of architectural decision-making during each of the case studies, and

c) identifying limitations and inadequacies in the core IAL, to model and analyze specific decision situations during the case study, and

d) proposing modifications or extensions to the core IAL to overcome these inadequacies.

Since it is not practically possible to explore all features included in the IAL during one case study, several case studies were performed, each focusing on a subset of IAL features. The utility of the IAL and its extensions to support architectural decision-making were evaluated by obtaining feedback from stakeholders and designers in industrial projects, on the usefulness of the modeling and analyzing performed of architectural decision-making in their respective organizational setting.

1.4. Research Contribution and Limitations

1.4.1. Research Contribution Overview

The main contribution of this thesis research is the use and adaption of intentional actor modeling and analysis concepts developed to support early requirements analysis to model and analyze architectural decision-making in organizational settings. This contribution includes exploring and illustrating by use of case studies the advancement of the IAL over current architectural description approaches and the identification of limitations and inadequacies of core IAL to support the intentional modeling and

---

\(^1\) With “strawman” version, we mean a first and likely insufficient version of the IAL, as a starting point for the research exploration.
analysis of architectural decision-making. A secondary contribution of this thesis is the identification of intentional concepts that support going beyond the expressiveness offered by core IAL (which draws its main concepts from i* and the NFR frameworks). This contribution is considered secondary, due to the exploratory nature of this thesis research and the small number of industrial case studies performed.

Each case study chapter (chapters 4 to 9) is based on a published paper. The following briefly discusses the contribution of each publication (selection criteria for choosing the case studies are discussed in section 1.6). Each of the paper focuses on some part of the IAL, illustrating the use of core IAL to represent and reason about architectural decision-making, as well as identifying and proposing some extensions to the core IAL. The table below summarizes each case study related publications.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Emergent focus</th>
<th>Publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mitel WML</td>
<td>Industrial</td>
<td>Mainly collective decision making using goal graphs; extrapolated distributed decision-making using actors; some knowledge capturing for reuse using actors</td>
<td>STRAW’01 workshop paper (Daniel Gross &amp; Eric Yu, 2001a)</td>
</tr>
<tr>
<td>Mitel OAMP</td>
<td>Industrial</td>
<td>Mainly collective decision making using goal graphs; extrapolated distributed decision-making using actors</td>
<td></td>
</tr>
<tr>
<td>Mitel Console</td>
<td>Industrial</td>
<td>Mainly collective decision making, using goal graphs</td>
<td></td>
</tr>
<tr>
<td>The Phoenix Insurance</td>
<td>Industrial</td>
<td>Distributed decision making, using actors and goals graphs</td>
<td>ISTAR’01 workshop paper (D. Gross &amp; E Yu, 2010)</td>
</tr>
<tr>
<td>Siemens Control System</td>
<td>Industrial</td>
<td>Qualitative and Quantitative NFR integration modeling and analysis, using goal graphs and scenario modeling</td>
<td>Book chapter (Daniel Gross, et al., 2011)</td>
</tr>
<tr>
<td>App. Platform NFRs</td>
<td>Industrial</td>
<td>Hypothetic distributed decision-making, using actors and goal graphs</td>
<td>WICSA1 workshop paper (Chung, et al., 1999)</td>
</tr>
<tr>
<td>KWIC system</td>
<td>Literature</td>
<td>Separation of concern decision-making, using actors and goal graphs</td>
<td>TEFSE’02 workshop paper (Daniel Gross &amp; Yu, 2002)</td>
</tr>
<tr>
<td>Prototype tool analysis, own</td>
<td>Personal</td>
<td>Collective decision-making, using actors and goal graphs</td>
<td>– was eventually aborted</td>
</tr>
<tr>
<td>industrial development effort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM Eclipse, Application</td>
<td>Industrial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lifecycle Framework (ALF)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
“Architectural design to meet stakeholder requirements” (Chung, Gross, & Yu, 1999): This paper reports on the modeling and analysis of an architectural design discussion of the Keyword in Context (KWIC) system when placed in a larger organizational context. The architectural design discussion of the KWIC system was first reported by Parnas (1972), and then expanded on by Garlan and Shaw (1994). Chung and colleagues subsequently modeled and analyzed the expanded design discussion using a goal-oriented modeling approach (Chung et al., 1995). This paper extends the goal-oriented modeling and analysis with actor-oriented modeling and analysis illustrating the analysis with three organizational scenarios. Each scenario illustrates how architectural decision-making of a KWIC component architect is differently influenced when considered in context of requirements and design goals and priorities of different stakeholders and architectural designers in a broader organizational context.

The main contribution of this paper is the illustration of an intentional actor concept applied to architectural decision making in an organizational setting. Architectural, and architecturally relevant, decision-making of stakeholders and designers is encapsulated and intentionally interconnected. Intentional dependencies represent how the stakeholders and designers depend on each other for business and design goal achievement during decision-making.

This paper was a first of a number of papers which applied the intentional actor concept to the software architecture domain [e.g. (Castro, Kolp, & Mylopoulos, 2001; Grau & Franch, 2007; Daniel Gross & Yu, 2001a, 2002, 2010b; Kolp, Castro, & Mylopoulos, 2001)]. Section 3.3 compares and contrasts the approach taken with two representative approaches proposed by other authors, and indicates how this thesis approach to intentional actor modeling in the architectural design domain is still unique.

The contribution of the paper included the defining, the modeling and the analysis of each of the organizational scenarios, by applying the intentional actor concept to architectural decision-making that was placed in a hypothetical larger organizational context, as well as the participating in the writing of the scenario related chapter and sections. Chapter 4 is based on this paper and section 4.1 discusses how chapter 4 revises and expands the modeling and analysis reported in the paper.

“Evolving System Architecture to Meet Changing Business Goals: an Agent and Goal-Oriented
Approach (Daniel Gross & Yu, 2001a): This paper reports on a case study performed at the Mitel telecommunication company, a multinational corporation with its headquarter in Kanata, Ontario. During this case study key stakeholders and designers were interviewed about an ongoing design discussion related adding support for WML browsing capabilities on telephone sets for a main telephone switching system product. The design discussions took place while the organization undertook an overall architectural evolution effort of the organization’s systems and products.

The main contribution of this case study was the intentional actor and goal modeling and analysis of the architectural design discussions amongst organizational stakeholders and designers, as well as the obtaining of positive feedback from the participating stakeholders and designers, on the utility of the models to represent and communicate key design discussion arguments. Another contribution of this paper includes the use of intentional actors to represent and apply generalized design responsibility knowledge to guide architectural design in an organizational setting.

The thesis author performed all interviews, transcribed the interview recordings, and modeled and analyzed the design discussions using intentional actor and goal concepts, presented the models to the project team, and obtained feedback during the presentation. Chapter 5 in this thesis is based on this paper, and section 5.1 discusses how the thesis chapter revises and expands the modeling and analysis reported in the paper.

Supporting the evolution of software architectures in development organizations using intentional agents (Daniel Gross & Yu, 2010b): This paper reports on a case study performed at the IT department of The Phoenix insurance company. The Phoenix insurance is a main insurer in Israel. During this case study, the CTO and the enterprise-wide SOA architect were interviewed. During the interviews, the CTO and SOA architect reported an ongoing design discussion between the SOA architect and an enterprise component designers related to the adoption of a key SOA design principle during the design and integration of a consumer component into enterprise wide systems; the design discussion was represented and modeled using intentional actors and goals.

The main contribution of this paper was the use of intentional actors to represent and analyze competing design arguments of the different architectural designers, and the linking of the design arguments to goals and decision-making of higher-level (in the managerial sense) organizational stakeholders. This contribution further includes the identification of an intentional viewpoint concept to

---

2 A short version of this paper appeared in (Daniel Gross & Yu, 2001b)
represent the distinct but overlapping design arguments of the architectural designers. The CTO and SOA saw much utility in the support for communicating amongst the designer the competing design arguments, as well as the support for linking of the architectural decision-making to the organization’s higher-level business goals -- a feature that was seen by the CTO and SOA architect as a contribution to the organizational governance processes.

This thesis author introduced the CTO and SOA architect to actor and goal-oriented modeling, and then performed the interviews, and then summarized and modeled and analyzed the reported discussions using intentional agent and goal modeling. Chapter 6 is based on this paper. Section 6.1 discusses how chapter 6 further revised and expanded the paper.

From Non-Functional Requirements To Design Through Patterns: (Daniel Gross & Yu, 2001d)

This paper reports on the modeling and analysis in terms of NFRs of architectural design patterns (Gamma, Helm, Johnson, & Vlissides, 1995; Molin & Ohlsson, 1998) and of the application of design patterns during architectural design. Architects and designers often referred to design approaches using higher-level design terminology such as when referring to known design patterns, which compactly communicates known design approaches and design tradeoffs. This gave rise to the idea to investigate the design reasoning underlying such “packaged” design solutions.

The main contribution of this paper includes the application of a goal-oriented modeling and analysis technique to model and analyze design patterns and to model and analyze the successive application of design patterns during architectural design. This was the first paper that reported on the use of a goal-oriented modeling approach to represent and analyze the design tradeoffs and alternative design approaches discussed in design patterns in terms of their non-functional properties, as well as when applying design patterns during design. Chapter 7 in this thesis is based on this paper, which further revises the modeling and analysis approach to integrate the design reasoning with packaged solutions into the IAL meta-model.

The second author of the paper identified the pervasiveness of NFRs in the description of design patterns, and suggested modeling and analyzing design patterns using a goal-oriented modeling approach. The author of this thesis identified and selected the design patterns reported by Molin and Ohlsson (Molin & Ohlsson, 1998) as a case study, and performed the modeling and analysis of the design patterns reported in their paper.

---

3 A shorter version appeared in: (Daniel Gross & Yu, 2000)
Developing Non-Functional Requirements for a Service-Oriented Application Platform – a Goal and Scenario-Oriented approach (Daniel Gross, Yu, & Song, 2011): This book chapter reports on a case study performed at Siemens Corporate Research to enhance an existing method for analyzing non-functional requirements of a service-oriented application platform with a conceptual modeling and analysis approach. The main objective of this case study was to develop a conceptual modeling and analysis technique that supported deriving quantitative and qualitative NFRs for an application platform for control systems, from NFRs that were elicited from existing control system applications that were developed for different application domains (such as transportation or building security domain).

During this case study a requirements document for the application platform was analyzed together with excel spreadsheets that summarized the quantitative non-functional requirements of individual control systems. Chapter 8 in this thesis is based on this book chapter.

The main contribution of this paper was a proposal for integrating the modeling and analysis of qualitative and quantitative NFRs through scenario modeling, and the illustration of such an integrated modeling and analysis by use of the case study data. This integration approach also included the definition of a meta-model to support the specification of quantitative NFRs and their units, including support for converting between quantitative NFRs specified using different units.

Resolving artifact description ambiguities during software design using semiotic agent modeling (Daniel Gross & Yu, 2010a): This paper proposes a novel approach to dealing with possible ambiguity in the use of design terminology amongst designers. During “The Phoenix” case study (Chapter 6) it became apparent to this thesis author that the design terminology used during design discussions conflated two related by independent design concepts. To support defining design terminology in a design notation independent manner this author turned to semiotic in general and organizational semiotic in particular, and adapted an existing semiotic modeling approach for defining terminology during requirements analysis to support defining terminology during architectural design. Chapter 9 in this thesis is based on this paper.

The main contribution of paper is the adaptation and application of the semiotic modeling approach to the architectural design domain, and the linking of this modeling approach to the goal-oriented modeling approach used in this thesis.

4 This thesis author developed a prototype tool (an MS-Excel plug-in written in visual basic.net) to demonstrate the proposed analysis method.
Dealing with system qualities during design and composition of aspects and modules: an agent and goal-oriented approach: (Daniel Gross & Yu, 2002): This paper illustrates the use of intentional actors to represent and modularize design reasoning associated with the design of software modules and aspects (Kiczales et al., 1997; D.L. Parnas, 1972). More specifically, this paper illustrates the non-functional design reasoning during the design of modules and aspects has an effect on the non-functional properties of the overall system during the composition of modules and aspects during design and runtime.

The main contribution of this paper is proposal of a concern reasoning agent, an application of intentional actor concept, to model and analyze the addressing of concerns and related design of modules and aspects (Stanley M. Sutton Jr. & Tarr, 2002; Tarr, Harrison, & Sutton, 1999). A summary of the case study reported in this paper is included in the Appendix of this thesis.

1.4.2. Study limitations

This research was limited due to the nature and availability of research data with industrial case studies. For some case studies, only written requirements and/or architectural design documentation was available for conceptual modeling and analysis, limiting the scope of modeling and analysis possible. In other case studies, the focus, scope, and level of architectural design and decision details were largely determined by the level of access given to the researcher.

Access to industrial projects was usually limited to studying a particular decision problem that was addressed by a predetermined number of stakeholders and designers during a set timeframe. In these cases, the research data collected was limited to team discussions of alternative design choices, with discussions converging towards acceptable decisions; often little data was available about the organizational context and about related distributed decision-making. Additional data had to be drawn from other sources, such as written documentation provided by the industrial partner, or to extrapolate from the given data, and then obtain feedback from project participants.

1.5. Conceptual Framework

A conceptual framework is a system of concepts, assumptions, expectations, beliefs, and theories that support and inform a research undertaking (Maxwell 1996). The conceptual framework that supports this research draws from Yu’s characterizations of agents-in-the-world (Yu 2001; Yu 2001; Yu 2009). The term agents-in-the-world refers to human decision-makers who are actively involved in decision-making in organizational settings. By attributing to agents-in-the-world socio-psychological
properties such as intentionality, autonomy, strategic reflectivity, rational self-interest, sociality, identity, and boundaries, Yu explains decision behavior of humans in organizational settings.

Yu developed the agents-in-the-world concept to support the analysis of the decision-making of organizational stakeholders during collective business-processes redesign efforts. These efforts usually take place while developing or improving information systems – an analysis activity Yu calls early requirements (Yu 1993; Yu and Mylopoulos 1996).

A key premise during early-requirements analysis is that stakeholders, who are members of different organizations, or organizational units, have a degree of freedom to choose whether and how to do, or redesign, their work. Along with this freedom, though, is the uncertainty that stakeholders have as to whether their work redesign proposals, that impinge on the manner work needs to be done by others, will be accepted. However, if a particular stakeholder knows what motivates other stakeholders, and if he/she has knowledge of the social web of interests that connects between related stakeholders, then the stakeholder can reasonably anticipate the kind of work redesign adjustments that others are likely to make or adopt. When decision-making behavior can be anticipated, redesigned work processes, and related systems capabilities, are more likely to succeed than when it is not.

This thesis suggests that the agents-in-world concept is applicable to describing and analyzing architectural decision-making behavior of stakeholders and designers in development organizations. Following is a brief discussion of the main agents-in-the-world properties when viewed from the perspective of software architectural design in development organizations.

**Autonomy:** Traditionally, architects use block and arrow diagrams to describe different software system structures and processes. In an organizational setting, a key question is whether a proposed architecture can be implemented. For example, there often exists uncertainty whether different departments, each responsible for different parts of the architecture, will adopt architectural choices prescribed to them. Uncertainty often derives from conflicting demands that originate from different organizational stakeholders. Conflicting demands occur, for example, during organizational change efforts, when, on the one hand, current operational needs act as forces to retain existing architectural structures, while at the same time, future strategic needs require introducing architectural changes (Engels et al., 2008; Daniel Gross & Yu, 2001c). Introducing autonomy into descriptions of architectural models allows accounting for and analyzing such kinds of uncertainty, as for example, by supporting the modeling and analysis of autonomous decision-makers, their organizational design responsibilities, and the demands put on them by other autonomous decision makers that have an effect on architectural structures and system processes. Such modeling and analysis allows a deeper understanding of possible
decision-behavior of stakeholders and designers during architectural design in development organizations.

**Intentionality**: Creating a software system architecture often requires specialized knowledge in different technical fields. The architectural design task is therefore broken down and allocated to different designers or design teams, each responsible for different parts, or aspects of the emergent architectural design of a system. Intentional concepts allow describing desired design results, while leaving to others the actual design tasks. Introducing intentionality into an architectural modeling language thus allows specifying design allocation and delegation relationships in the organizational setting.

Another aspect of intentionality relates to the purposefulness of architectural artifacts. Asking why a design artifact exists, is structured, or functions in a particular manner, reveals the underlying design intents of stakeholders and designers. In effect, intents, such as business and design goals, justify and guide design choices.

**Sociality**: A key approach to dealing with the complexity of architectural design is to “divide and conquer”. Architects repeatedly decompose architectural descriptions into parts and composition rules. The architectural design of each part and of each composition rule is then addressed separately. The assumption behind such an approach is that the behavior and the architectural properties of the whole are fully determined by the behavior and architectural properties of the parts.

However, in an organizational setting in which the architectural design of parts is allocated to decision-makers, who have some degree of autonomy during decision-making, it is often not possible to predict the architectural design of the parts and hence the architectural properties, and even the behaviors, those parts would exhibit. This is in particular the case when designers of parts need to deal with conflicting demands, originating from different stakeholders and designers in the broader organizational context.

The architectural properties of the whole are therefore dependent on various direct or indirect reciprocal social and technical relationships between stakeholders and the architectural designers of the parts. Such relationships influence decision-makers in their choices. This notion of sociality thus points to richer kinds of relationships between architectural parts and the whole than are accounted for in block and arrow types of diagrams and mechanistic compositions or decompositions assumptions. There is a need to investigate such richer socio-technical relationship between stakeholders and designers, to anticipate decision-behavior, and consequently, the architectural properties of the system and its parts.

**Strategic reflectivity**: During architectural design, stakeholders and architects explore and evaluate
alternative architectural approaches and respective design tradeoffs. Furthermore, due to changing organizational needs and goals, today’s adopted approaches often become tomorrow’s rejected ones. Exploring alternative approaches is an ongoing reflective and strategic process wherein stakeholders and designers evaluate how well alternative approaches support their individual strategic objectives.

To support strategic reflectivity, an architectural description approach needs to allow representing alternative approaches—the ones explored but rejected and the one adopted. Furthermore, it needs to support dealing with the temporal aspect wherein today’s accepted approaches become tomorrow’s rejected ones.

**Rational self-interest**: During architectural design, stakeholders and designers in the development organization each present individual preferences. These are then merged into the collective design reasoning effort to develop the architectural design solutions. During these design reasoning and decision-making processes, each stakeholder and designer aims to promote or select those choices that best serve his/her individual interests. An architectural description thus needs to support representing and analyzing individual preferences of stakeholders and designers.

Keeping track of individual preferences is also useful to anticipate decision-making behavior. For example, an enterprise architect who introduces an asynchronous messaging infrastructure for enterprise applications needs to anticipate whether this choice will be adopted by application designers in the organization. To anticipate the adoption of a design approach by designers, the architect can evaluate whether the approach serves, in addition to his own needs and preferences, some important needs of the other designers, who need to adopt an asynchronous messaging approach in their own designs. For example, if asynchronous messaging simplifies the messaging-related design of applications, and simplifying design is indeed an important advantage to designers, then the architect has found some supporting evidence for anticipating that adoption of this design approach could be forthcoming.

This line of reasoning relies on the assumption that stakeholders and designers pursue their own interest in a rational manner. By assuming such a rational self-interest property, it is possible to reason about their decision behavior if their motivations and interests are known. The decision behavior then becomes more (but not fully) predictable, keeping in mind that not all interests may have been elicited and that stakeholders and designers have a degree of autonomy to make own choices.

**Identity and boundary**: In development organizations, one main purpose of architectural design is the identification of needed system capabilities and the allocation of these capabilities within the system’s structure. In organizations, stakeholders and designers often reason about what capabilities architectural entities ought to have, and which organizational unit or teams should be made responsible
for the design of those capabilities.

The identity of an organizational unit can be conceived as the total of all objectives it is committed to fulfill. If an organizational unit is made responsible for the design of additional capabilities, then this effectively changes the design objectives, and hence the identity, of the organizational unit. During architectural change and evolution, for example, architects often need to reallocate capabilities, thereby shifting the boundaries of design responsibilities of the organizational units.

**Summary:** The aforementioned properties support and reinforce each other to explain and thus help anticipate the decision behavior of agents-in-the-world. First, autonomy is an inherent property of agents-in-the-world, which makes the agent’s decision behavior problematic and in principle hard to anticipate to outside observers (i.e. other agents-in-the-world). At the same time, intentionality is a key driving force behind decision behavior. To explain or anticipate decision behavior, one can look for intents that motivate decision-making of agents-in-the-world. Intents, together with the assumption of rationale self-interest, helps outside observers to reasonably link between the intents externally ascribed to an agent-in-the-world, and his or her anticipated choices among alternatives. If an agent-in-the-word would be irrational, then it would be impossible for an outside observer to explain or predict the agents’ decision-making, even if the agent’s motivations and needs are known.

Sociality assumptions, together with strategic reflectivity, helps place the decision behavior of agents-in-the-world into a broader social and organizational context. Decision behavior of an agent-in-the-world is not an isolated phenomenon, but affects, and is affected by, the anticipated decision behavior of the other agents-in-the-world included in agents-in-the-world the broader social and organizational environment. The combination of strategic reflectivity and sociality indicate that agents-in-the-world’s own intents as well as intents of others matters; and that each agent-in-the-world aims to go about achieving their intents, usually, in agreement and cooperation with others.

Finally, identity and boundary give a sense of the reasonable limit to the responsibility that an agent-in-the-world is likely to adopt or be willing to delegate. Given the right incentive, agents-in-the-world may decide to expand the goals they wish to pursue, thereby broadening their identity and boundary. On the other hand, agents-in-the-world will sometimes seek to narrow their boundary of responsibility, such as when consolidating their focus on core competences.

### 1.6. Case Studies Overview

This research used two sources for studying architecture decision-making. One was architectural decision-making reported in the literature, and the second was data collected from industrial projects,
such as design documentation, as well as interviewing of stakeholders and designers involved in architectural decision-making. In general, criteria for selecting cases to study were as follows:

- The subject matter involved architectural decision-making
- Architectural design alternatives would be identified and discussed in terms of relevant tradeoffs
- Design reasoning was attributed or could be traced to different stakeholder and designers within and/or across organizations
- Business considerations were included in design discussions

Since it was not practically possible to apply all modeling features included in the IAL during each case study, different case studies focus on different aspects of the language. Also, since some case studies replicated results, not all case studies are reported in the main body of this thesis document. Appendix A briefly describes the illustrations and case studies omitted from the main thesis body.

During this research, four example illustrations were developed based on the analysis of architectural decision-making reported in the literature. The first illustration (Chapter 4) involved analyzing the “Key Word in Context” (KWIC) architectural design example, extending prior illustrative work by (Chung et al., 1995; D.L. Parnas, 1972; Mary Shaw & Garlan, 1996). In this first illustration, the reasoning about alternative architectural approaches for a KWIC system was placed into an organizational context by linking architectural decision options to intents of higher-level stakeholders and designers in the organization.

The second illustration (Chapter 7) involved analyzing patterns in a design pattern language and the application of those patterns during the design of a distributed real-time alarm system. The pattern language was developed while eliciting key design knowledge during an industrial reengineering project (Molin & Ohlsson, 1998). This second illustration focused on reasoning with coarser grained architectural decision units, such as those exemplified in design patterns, and showed how such coarse-grained units of design reasoning can be represented, applied, and analyzed during architectural design.

The third illustration (Appendix A) involved the architectural analysis of a high-performance software cache that was designed to improve the performance of IBM’s high-performance web servers (Degenaro, Iyengar, & Rouvellou, 2001; Iyengar, 1999; S.M. Sutton Jr. & Rouvellou, 2000). This illustration focused on reasoning about alternative architectural choices and on the corresponding partitioning of the architectural component structures into organizational units of design responsibilities associated with different software cache’s design concerns. In this third illustration it was assumed that main architectural cache components, each addressing a specific cache concern, correspond to main
organizational design responsibilities. A hypothetical higher-level organizational context was added to illustrate the linking between business-related considerations to architectural design goals and decision-making.

Finally, the last illustration (Appendix A) involved analyzing a design decision during an effort (by this thesis author) to design and develop a research data analysis tool written in Visual Basic.NET and the SWI-Prolog language. This illustration focused on representing design reasoning structures that shaped the design of aspects and modules that cut across each other (Kiczales et al., 1997). This illustration is briefly discussed in Appendix A.

In addition to these illustrations, several industrial case studies were analyzed at industrial partner organizations: three at the Mitel Corporation, one at Siemens Corporate Research, and one at The Phoenix Insurance.

The first three case studies at the Mitel Corporation involved studying architectural decision-making related to a) the design of an administrative software management platform for telephone switching systems, b) the design of an Attendant Console product, and c) the design of internet browsing capabilities for telephone sets of a major switching system (Chapter 5). Of those three case studies, only the last one is reported in the main body of this thesis. The other two are summarized in Appendix A.

A fourth industrial case study at the Siemens Corporation involved studying the requirements and high-level architectural design of an application platform for industrial control systems (Chapter 8). The focus of this study was on integrating the analysis of design goals that are expressed in both qualitative and quantitative manners, and determining how they can be related to each other through alternative high-level architectural descriptions of a software system.

Finally, the fifth case study involved studying architectural decision-making at an insurance company during the evolution of enterprise systems towards service-orientation (Chapter 6). This last industrial case study focused on dealing with overlapping design responsibilities in IT organizations and the linking between design and organizational responsibilities of higher level (management) stakeholders.

During the analysis of the last case study, a novel terminology modeling and analysis approach based on a semiotic modeling technique was also developed and proposed (Chapter 9). This modeling technique supports dealing with ambiguity that may occur when using design terminology during design reasoning discussions amongst designers. Using this approach terminology referring to architectural approach can be analyzed to reveal if operational meaning of the term is shared.
1.7. Exploration of intentional actor concept

A focal point of this research was the clarification of the intentional actor concept – the focal concept during organizational modeling and analysis – when applied to the architectural domain. At the beginning of this thesis research, the meaning was not clear. Should intentional actors represent development organizations, organizational units, teams and individual designers in the development organization? Should intentional actors represent architectural abstractions such as systems, subsystem, modules and components under design, and if so, in what way are intentional actors a different representation than traditional architectural representations? What is the meaning and use of intentional dependencies during architectural design, in particular when intentional actors represent architectural abstractions?

During the first case study, when analyzing the KWIC architecture in an organizational context (Chung et al., 1999), intentional actors were interpreted as organizational responsibilities in a development organization. Instead of having an intentional actor represent the KWIC component, an actor represented the KWIC development team responsible for the design of the KWIC component.

During the WML project case study (Daniel Gross & Yu, 2001a), the intentional actor interpretation chosen was closer to traditional architectural abstractions. Intentional actors represented architectural modules or components associated with design goals. Design reasoning performed by stakeholders and designers was captured outside of intentional actors in a separate goal graph view. This introduced a global intentional actor responsible for architectural reasoning and decision-making. This view of intentional actors while supporting the allocation of design goals to architectural artifacts lacked the notion of distributed actor design reasoning and decision-making. This view was thus closer in spirit to agent-oriented modeling and analysis approaches such as Tropos (Castro et al., 2001) and architecture as organizations approaches (Kolp et al., 2001).

Later work looked at another conceptual approach to intentional actors that focused on the modularizing of design reasoning related to different design concerns (Daniel Gross & Yu, 2002), whereby concern reasoning agents captured the design reasoning related to different design concerns. The composition of concern reasoning agents led to an analysis of the emergent global quality properties of the software system. This interpretation of intentional actors conflated the notion of concern (Stanley M. Sutton Jr. & Tarr, 2002; Tarr et al., 1999) with the notion of an organizational design responsibility. Since concerns are considered inherently separated, concern reasoning agents supported separating out design reasoning related to dealing with design concerns. Although different concerns
could be allocated to different organizational stakeholders, concern reasoning agents are not primarily an organizational concept that supported dealing with distributed design reasoning in development organizations.

During the WML project case study (Daniel Gross & Yu, 2001a), the intentional actor interpretation chosen was closer to traditional architectural abstractions. Intentional actors represented architectural modules or components associated with design goals. Design reasoning performed by stakeholders and designers was captured outside of intentional actors in a separate goal graph view. This introduced a global intentional actor responsible for architectural reasoning and decision-making. This view of intentional actors while supporting the allocation of design goals to architectural artifacts lacked the notion of distributed actor design reasoning and decision-making. This view was thus closer in spirit to agent-oriented modeling and analysis approaches such as Tropos (Castro et al., 2001) and architecture as organizations approaches (Kolp et al., 2001).

Later work looked at another conceptual approach to intentional actors that focused on the modularizing of design reasoning related to different design concerns (Daniel Gross & Yu, 2002), whereby concern reasoning agents captured the design reasoning related to different design concerns. The composition of concern reasoning agents led to an analysis of the emergent global quality properties of the software system. This interpretation of intentional actors conflated the notion of concern (Stanley M. Sutton Jr. & Tarr, 2002; Tarr et al., 1999) with the notion of an organizational design responsibility. Since concerns are considered inherently separated, concern reasoning agents supported separating out design reasoning related to dealing with design concerns. Although different concerns could be allocated to different organizational stakeholders, concern reasoning agents are not primarily an organizational concept that supported dealing with distributed design reasoning in development organizations.

During the final stages of research, earlier work, and in particular the work done on modeling and analyzing the WML project, was revisited and reinterpreted using the interpretation of the intentional actor as organizational responsibilities.
2. Related work

2.1. Architecture and Organization

Already in the 1960’s Melvin Conway identified a correspondence between the design structure of a system and the organization’s communication structures that emerged between organizational participants when creating and evolving the system (Conway, 1968; Daniel Gross & Yu, 2002), a correspondence now known as Conway’s Law.

Conway asserted that an organization should be structured according to the evolving communication and coordination needs of the organizational participants who are creating and evolving the system. In particular, the organizational structure needs to be designed flexibly, so that communication structures remain manageable while the organizational participants change and evolve the system. Conway further observed that the need to deal with communication and coordination in organizations leaves an indelible mark on the structure of the developed system (Conway, 1968; Herbsleb & Grinter, 1999b). According to Conway, during architectural decision-making, architects are simultaneously dealing with establishing and evolving artifacts, as well establishing the organizational communication structures amongst the stakeholders and designers responsible for different parts of the system.

Building on similar insights, Parnas et al. proposed criteria for modularization during system design based on principles of information hiding (D.L. Parnas, 1972; D.L. Parnas, Clements, & Weiss, 1984). Parnas explained that a “module” represents a work assignment to a programmer or a team of programmers. A module structure represents the decomposition of a program into modules, while at the same time, the structure limits the assumptions that each module team can make about the design choices included in other modules (D.L. Parnas et al., 1984). The aim of information hiding is thus to encapsulate design decision knowledge within modules, thereby ensuring that the assumptions each programming team can make about decisions included and/or changed in other modules is minimized.

Parnas emphasized that work assignments and decision-making are intertwined: they are “two sides of the same object” (D.L. Parnas, 2010). Furthermore, according to Parnas, modules not only represent decisions already made, but also deferred decisions (D. L. Parnas, 1976). Consequently, in an organizational setting, a module structure of a system corresponds to a structure of organizational units, each tasked with the adoption, and possible evolution of a decision, or tasked with some future design and decision-making.

In the late 1980s, Curtis et al. noted that to develop methods and tools that significantly improve
quality and productivity during software development, it was necessary to study and address organizational problems during software requirements and design (B. Curtis et al., 1988; W. Curtis et al., 1987). Curtis et al. focused on organizational “people” problems during decision-making, since earlier studies on quality and productivity indicated that focusing only on solution technologies, such as the application of new engineering tools and methods to projects, had only a 30% impact on reliability, and no impact on productivity (B. Curtis et al., 1988; W. Curtis et al., 1987), citing (Card, McGarry, & Page, 1987).

To study requirements and design activities in an organizational setting, Curtis et al. proposed using a layered behavioral model that focuses on the analysis of decision processes at several organizational levels: the individual, the team, the project, the company and the company’s milieu (figure 2-1). The authors explain that the layered behavioral model helps focus on the decision behavior of organizational participants who create software artifacts, rather than on the design and evolution of the artifacts themselves. The authors further explain that the layered behavioral model also helps in studying the links between individual decision behavior to the goals of the business, and between the goals and the organizational context that justifies those goals (B. Curtis et al., 1988; W. Curtis et al., 1987).

Figure 2-1: Layers of organizational decision-making behavior (B. Curtis et al., 1988)

After studying 19 projects in nine companies, Curtis et al. found that the most salient problems, in terms of additional effort and mistakes attributed to them, were related to a) the thin spread of application domain knowledge across organizational stakeholders and designers, b) the need to deal with fluctuating and conflicting requirements in an organizational setting, and c) the need to deal with communication and coordination breakdowns in the organization (B. Curtis et al., 1988; W. Curtis et al., 1987). Curtis et al. explain that each of these problems typically emerged at one organizational level
and then affected decision processes at other organizational levels.

For example, Curtis et al. found that architects responsible for the integration of different parts of a system at project level typically encountered difficulties due to the fragmentation of knowledge amongst individual designers (individual level). Often only architects understood why disparate system components needed to be integrated into particular kinds of architectural structures, and had to negotiate with the different component designers about the particular integration techniques utilized in a project. As another example, managers at company level responsible for linking between executive decision-making and technical decision-making of system engineers often expressed frustration at being removed from both decision-making activities, making their specific role in the organizational decision-making chain unclear. These early studies, however, stopped short of actually proposing approaches to support dealing with the organizational decision-related problems and issues identified in their studies.

![Figure 2-2: Architectural Business Cycle (Bass et al., 1998)](image)

In the last decade there was renewed interest in empirically understanding how software architecture is done in practice as a social and organizational process, and how architects can be supported by methods and tools (Bass et al., 1998; Grinter, 1999; Herbsleb & Grinter, 1999a, 1999b; Herbsleb & Moitra, 2001; Poile, Begel, & Nagappan, 2009; Rik & Hans van, 2009).

Bass, Clements and Kazman (1998) introduced the notion of an architecture business cycle (figure 2-2), which shows how “the relationship between business goals, product requirements, practitioners experience, architectures, and fielded systems form a cycle with feedback loops that a business can manage”. This approach emphasizes the relationship between the architecture that is created, the organization that produces it, and the organization for which it is produced.

In her study of how architects work, Grinter (1999) reported how architects facilitate and are involved in extensive coordination and communication efforts among the various organizational
members, to extract their concerns, present solutions, and synchronize the collective design and development efforts. Herbsleb and Grinter also identified coordination problems during the integration of product parts, in particular when developed in distributed locations. To alleviate integration problems, the authors found ample evidence on the importance, during design work “hand-off”, of providing a clear idea of which steps in the design have been taken and which have not, and, similarly, in coordinating when, how, who, and where the design is being handed off (Herbsleb & Grinter, 1999b).

Jan Bosch documented the organizational structures of several organizations involved in software product lines, and identified characteristics and applicability of different organizational structures for different kinds of software product line development efforts (Bosch, 2000).

In a series of ethnographic studies of architects in a large software development organization, Farenhorst and Vliet found that although architectural design is a primary responsibility, architects also spent a considerable amount of time explaining architectural design to other relevant stakeholders in the organization, and monitoring the quality achieved by the proposed architecture, as well as the compliance with the architecture in the development organization (Farenhorst & van Vliet, 2009).

With the globalization of development work, architectural problems related to distributed work and decision-making have become particularly pronounced (Herbsleb & Moitra, 2001; Herbsleb, Paulish, & Bass, 2005). For example, Herbsleb et al., studying global work practices at Siemens, identified key coordination problems with distributed work, including identifying the level of detail required when communicating to remote contractors. This particularly included taking into account the different levels of background knowledge and expertise that contractors brought to the project, and the need to avoid potential cultural misunderstandings of design requirements (Herbsleb et al., 2005).

In a recent study on the coordination of large-scale software development at Microsoft, Poile et al. studied organizational coordination behaviors deemed helpful or unhelpful across teams. One significant coordination issue identified, was the teams’ need to know about the status of, and changes in, decisions made by other teams, including the reasons for changes. Coordination behavior also included reallocation of work assignments. Teams needed to understand ownership (division of responsibilities), and the reasons for any change of responsibilities across individuals and teams (Poile et al., 2009). When such aforementioned knowledge was not communicated, it was perceived as unhelpful behavior.

The need to offer method and tools to support dealing with organizational issues that impinge on software development activities has recently become the focus of a workshop series called Cooperative and Human Aspects of Software Engineering (Li-te et al., 2008). In the workshop summary, the
organizers pointed to the growing body of research being published in software engineering venues that
deal with issues arising from the fact that software is developed by people in organizations, and hence
gives rise to different types of organizational issues. They emphasize the need to understand these
issues in order to improve the methods and tools for software engineering, and consequently improve
the creation and maintenance of software.

While prior research points to the importance of dealing with the intertwined nature of architectural
decision-making and the organizational settings in which decision-making occurs, current approaches to
describing architecture offer little representational and analytical support.

2.2. Design rationales and design patterns

The approach explored in this thesis has its roots in approaches for capturing design rationales during
software design, such as the design rationale frameworks proposed by Potts and Burns (1988) and
Conklin (1989), and the goal-oriented design rationale approach developed by Lee (J. Lee, 1992, 1997).
Drawing from these early works, Chung developed a goal-oriented modeling framework for dealing with
non-functional requirements during requirements analysis and architectural design (Chung, 1993;
Chung, Nixon, & Yu, 1994; Chung, Nixon, Yu, & Mylopoulos, 2000). More recently, Jansen and Bosch
(2005) and Zimmerman et al. (2007; 2008; 2009) proposed decision-oriented rationale modeling
approaches to capture architectural decision-making. In AI and the logics domain works have been
proposed for more formal argumentation support (Chesnevar, Maguitman, & Loui, 2000; Dung, 1995).
Jureta et al. (2009) adopting ideas from argumentation in AI proposing a novel logic based rationale
approach and algorithm to support validating requirements artifacts documented during requirement
elicitation discussions.

Over the last decade, design patterns, including architectural design patterns, have received much
attention as an informal way of capturing design rationales (Beck & Johnson, 1994; Buschmann, 1996;
Gamma et al., 1995; Wendorff, 2001; Olaf Zimmermann et al., 2007). More recently, formal notations to
describe design patterns were proposed (Conte, Fredj, Hassine, Giraudin, & Rieu, 2002; Eden, 2002; Kim
& Carrington, 2009; Sterritt, Clarke, & Cahill).

All these approaches support capturing design reasoning, decisions, and design solutions using
varying degrees of formality. However, no support is offered for dealing with architectural decision-
making that is distributed and allocated amongst different stakeholders and designers in development
organizations, which adds a layer of social and organizational uncertainty to the reasoning and decision-
making represented, captured and analyzed.
2.3. Requirements Traceability

Traceability models provide rich structuring concepts for linking requirements to design and implementation elements (Gotel & Finkelstein, 1994; Ramesh & Jarke, 2001). Such traceability links aim at “describing and following the life of a requirement, both forwards and backwards” (Gotel & Finkelstein, 1994).

While traceability frameworks support capturing the link between requirements, design, and implementation artifacts, their main purpose is to record links established between software engineering artifacts, rather than modeling and analyzing the goals and the design reasoning and argumentation processes that lead to establishing and linking between requirements and architectural design artifacts in organizations.

2.4. Requirements Negotiations

This thesis research shares some of the goals that approaches for supporting negotiations of requirements amongst multiple organizational stakeholders have (B Boehm & Bose, 1994; B. Boehm & Ross, 1988; B.W. Boehm & Ross, 1989; Rick Kazman, In, & Chen, 2005). For example, Boehm et al. propose a management theory (“theory-w”) which offers a conceptual argumentation ontology, and a structured negotiation process to support stakeholders in converging on a mutually satisfactory set of requirements (B Boehm & Bose, 1994). More recently Kazman et al., building on the work by Boehm et al., proposed a method that links requirement negotiations to architectural design decisions, thus bringing architectural implications of requirements to the forefront during negotiations (Rick Kazman et al., 2005).

While such negotiation approaches offer useful ontological concepts to support structuring negotiations and collectively prioritizing negotiation points, they do not support modeling and analyzing argumentation and decision-making that occur in distributed across different organizational participants in an organizational settings.

2.5. Software Architecture and Decision-making

The software architecture research community has been developing modeling notation, languages, and tools to deal with representing, analyzing, and evolving complex architectural structures of software systems (P. B. Kruchten, 1995; Perry & Wolf, 1992; Mary Shaw & Garlan, 1996). Researchers also focused on representing, documenting, evaluating, and analyzing architectural structures and properties — in particular the non-functional properties — of software architectures (Amyot & Mussbacher, 2003;
Bachmann et al., 2005; Bosch, 1999; Buschmann, 1996; Chung et al., 1995, 1997; R Kazman & Bass, 2005; Rick Kazman et al., 2005; R Kazman et al., 2000; Niemelä & Immonen, 2007; Perry & Wolf, 1992).

Recently, researchers in the software architecture community have become interested in focusing on the architectural decision as a focal concept during architectural design, proposing approaches for capturing and managing the architectural design decisions that shape software architectures (Jansen & Bosch, 2005; Tyree & Akerman, 2005). Tang et al. surveyed architectural rationale approaches (Tang, Babar, Gorton, & Han, 2006), and proposed a conceptual framework for capturing architectural rationales (Tang et al., 2009). Zimmerman et. al. proposed a comprehensive architectural decision management and reuse approach (O. Zimmermann et al., 2009). Some work has also focused on systematically deriving architectural structures and processes from non-functional requirements supported by preexisting architectural knowledge (Beck & Johnson, 1994; Bosch, 1999; Buschmann, 1996; Chung et al., 1995).

While the aforementioned approaches capture important architectural knowledge, they fail to support architectural decision-making as occurring in an organizational setting, in which business goals are addressed through decision-making of different organizational stakeholders and designers in the development organization.

2.6. Software System Modularization

A key architectural design task relates to the partitioning of software development tasks into smaller development tasks that are assigned to collaborating organizations, teams, and individuals. With the partitioning of systems into work assignments, communication and coordinating distributed work efforts become important. Already in the early 1970s, Parnas identified information hiding and modularization of work assignments as key approaches to reducing communication overhead during system development and evolution (D.L. Parnas, 1971, 1972; D.L. Parnas et al., 1984).

Over the years, software engineering research and practice has identified different approaches and units of modularization, such as components, objects, and—more recently—aspects and concerns. Aspects, for example, are modularization units to keep separate the decision knowledge embodied in requirements, design, and implementation artifacts, that otherwise would crosscut other requirements, design, and implementation artifacts, respectively (Clarke, Harrison, Ossher, & Tarr, 1999; Garlan et al., 2000; Kiczales et al., 1997; Stanley M. Sutton Jr. & Tarr, 2002; Tarr et al., 1999).

While partitioning and modularizing systems into work assignments is the basis for distributing design and development work in organizations, current approaches focus on dealing with the
modularization of solution artifacts, so that the design knowledge embodied in artifacts is hidden and kept separate from other solution artifacts. Current modularization approaches offer little support in analyzing the process of partitioning and assigning design work itself. This involves modeling and analyzing organizational design responsibilities and then allocating design responsibilities to stakeholders and designers to support collective design and decision-making.

2.7. Actor and Goal-Oriented Modeling Approaches

Typically, software systems in organizations are constructed to improve organizational situations that are considered problematic (Kavalki & Loucopoulos, 2004). Problems can often be characterized in terms of competing goals that an organization aims to address, and in terms of alternative approaches that are proposed to address those goals. Goal-oriented approaches aim to support representing, identifying, clarifying, and analyzing goals in order to help understand problems and alternative solution approaches (Anton, 1997; Bubenko, Rolland, Loucopoulos, & DeAntonellis, 1994; Bubenko & Wangler, 1993; Dardenne, van Lamsweerde, & Fickas, 1993; Jacobs & Roland, 1995; Kavalki & Loucopoulos, 2004; van Lamsweerde, 2002). More recently, the Pedigreed Attribute elicitation Method (PALM) was proposed, a systematic method to elicit and express business goals that are considered relevant to architectural design (P. Clements & Bass, 2010).

Some goal-oriented approaches introduce the “softgoal” concept along with a notion of “satisficing” (Simon, 1981) to describe and analyze goals that are sufficiently—as opposed to absolutely—achievable (Chung, 1993; Chung et al., 2000; E. Yu, 1994). Softgoals are used to capture business and system qualities — such as competitiveness, security, maintainability, flexibility, extensibility — that are hard or impossible to formalize, yet that often serve as criteria to explore and select amongst alternative design approaches.

Goal-oriented modeling approaches represent goals as global concerns of the organizations without attributing goal and goal achievement to specific organizational participants. This precludes the modeling and analysis of organizational situations where stakeholders and designers have different motivations and goals, and reason differently about possible alternative design approaches. To specifically deal with organizational situations that involve multiple organizational decision makers, actor-oriented approaches and in particular approaches adopting Yu’s intentional actor modeling paradigm have received much attention (see for example, the i* Wiki and the Tropos project websites, respectively: http://istar.rwth-aachen.de/tiki-view_articles.php, http://www.troposproject.org/).

Yu first established the intentional actor paradigm together with the i* modeling framework to
support the redesigning of business processes, and to support early requirements analysis (E. Yu, 1994, 2001b; S. E. Yu, 2009). Yu’s actors are typically distributed and intentional. They represent human decision makers who pursue individual motivations and intents, and who decide how to participate in collective organizational process (re)design efforts.

Some works apply i* to software architecture. For example, Castro et al. proposed a requirement-driven software development methodology that takes a social-organizational perspective during early and late requirements, software architecture, and detailed design (Castro et al., 2001). Kolp et al. and Giorgini et al. propose treating software architecture as analogous to social organizations, and apply organizational styles derived from organizational literature to structure computational components into alternative architectures (Giorgini, Kolp, & Mylopoulos, 2002; Kolp et al., 2001). Grau et al. proposes utilizing i* as an architectural description language, equating actors with components, and intentional dependencies with connectors (Grau & Franch, 2007). Bastos and Lucena each propose systematic rules to derive intentional actor models in the architectural design domain from actor models in the requirements domain (Bastos, 2005; Lucena, 2010).

Common to these approaches is that they deal with multiple organizational decision-makers only during requirements. Architectural design reasoning is supported by a global goal graph. These approaches thus simplify design reasoning that is in reality performed by different organizational participants in an organizational setting as if it is performed the organizational as a whole. These approaches thus assum a global intentional actor who summarizes the reasoning contributions of many individuals and makes design decisions. Using a global design rationale graph precludes representing and analyzing how architecture emerges through decision-making activities of different organizational participants, each of whom reasons differently about individual goals and choices. Furthermore, in the aforementioned works intentional actors represent computational components rather than human decision-makers, making the interpretation of the intentional actor concept in those works closer to the notion of software agents (Wooldridge & Ciancarini, 2001). Section 3.3 compares and contrasts in more depth IAL and the aforementioned approaches.

In recent years, agent-oriented approaches were proposed to support modeling and analyzing social interactions (Cheong & Winikoff, 2006; Desai, Mallya, Chopra, & Singh., 2005; Kumar, Huber, & Cohen, 2002; Odell, Van Dyke, & Bauer, 2001). Telang and Sing applied a novel concept of commitment to the Tropos method (Telang & Singh., 2009). However, these model and analyze behavioral interactions amongst agents, whereas this thesis focuses on an agent (actor) concept that encapsulates the human designer’s intentional reasoning, associated with assignable organizational responsibilities.
2.8. Conclusion

Although empirical studies of software development in organizations over the last decades revealed the importance of dealing with the organizational setting within which architectural and architectural-relevant decision-making occurs, a main focus of research to-date in the architecture domain deals with representing, modeling and analyzing architectural solution structures. More recently, works have been proposed to represent design-decisions and rationale in order to capture why decisions were made.

However, although these important advances, little work to-date has been done on supporting collective and possibly distributed decision-making of stakeholders and architectural designers within an organizational setting; as well as to analyze decision-making in light of stakeholders and designers goals.

This research aims to explore this organizational and intentional dimension during architectural decision-making during a number of case studies, and to explore how an intentional and organizational modeling language could look like to supports stakeholders and designers to model and analyze their decision-making within their respective organizational settings.
3. A Proto-Intentional Architecture Language (IAL)

3.1. Requirements for the Language

Decision-making in software development organizations is often ad-hoc and disconnected from relevant decision-making elsewhere in the organization (B. Curtis et al., 1988). There is often a focus on describing solution approaches. NFRs that are discussed often appear to emerge out of nowhere, and thus lack deeper justification. Much architectural decision-knowledge, and in particular how decision-making relates to higher-level system and business goals is often tacit and not well understood. Management tasked with governance of decision processes often lacks direct knowledge about how architectural decision-making affect higher-level organizational objectives and, vice-versa, how higher-level decision-making trickles down to architectural decision-making. Both often appear disconnected and as parallel decision processes without interrelationship (B. Curtis et al., 1988).

The purpose of an IAL is to support a deeper understanding of architectural decision-making in the organization. This includes how local decision-making affects, and is affected by, other decision-making in the organization; and how decision-making of different more or less independent organizational participants is “strung together” to achieve higher-level design and business goals.

There is a need to support dealing more directly with the NFRs and business goals during decision-making and to support dealing with the organizational context in which decision-making occurs. Dealing NFRs and business goals includes support for clarifying, refining, linking and prioritizing NFRs and business goals, along with identifying architectural design approaches that address NFRs, and dealing with tradeoffs that must be made between NFRs during decision-making (Chung, 1993; Chung et al., 1995; Chung et al., 2000; P. Clements & Bass, 2010). Dealing with the organizational setting includes support for representing organizational decision-making responsibilities as well as linking organizational responsibilities to the architectural structures and artifacts that are being designed and/or changed (E. Yu, 2001a). In addition, facilities are needed to reason about the allocation and delegation of goals and constraints amongst stakeholders and designers (E. Yu, 1994).

By characterizing architectural decision-making processes in this manner, it becomes possible to represent and analyze how architectural decision-making is threaded through different organizational responsibilities, starting from responsibilities associated with upper management and down to responsibilities allocated to architects and designers of different subsystems and components of the architecture.
3.2. Overview of the Language

This section overviews a meta-model that captures the concepts of the intentional architecture language. The language is considered a proto-intentional language since its main purpose is to support the exploration of the intentional dimension of architectural design rather than to offer a language proposal for practical use.

This chapter overviews the core IAL and IAL extensions using core meta-model and extensions to the meta-model. Although overviewed before the case studies in later chapters, proposed extensions were not preconceived. They emerged out of representational and analytical needs identified during the case studies. The need for each extension was first identified in one case study. Some extensions were then found useful in other case studies. For example, the intentional collective concept (described below) was identified while analyzing the KWIC case study, and was then applied during the WML case study also. However, the extension to deal with intentional roles to support generalizing architectural responsibility knowledge was identified during the WML case study, but was not used elsewhere.

It is to be anticipated that the extensions can be generalized to architectural design and decision-making in other decision situations, however, additional case studies would be necessary to corroborate their broader applicability and use.

**Core meta-model:** The core meta-model includes concepts from both the i* and the NFR frameworks (Chung, 1993; Chung et al., 2000; E. Yu, 1994). This includes i* concepts that support the strategic dependency and the strategic rationale models, as well as NFR Framework concepts that support modeling and analyzing NFR goal graphs. The core meta-model thus supports two styles of non-functional modeling and analysis: the strategic rationale modeling style, which focuses on functional decomposition driven by non-functional requirements, and the NFR goal graph style, which focuses on non-functional decomposition, with functionality treated separately.

**Extension to reason with intentional collectives:** Chapter 4 describes how core IAL was used to model and analyze the design of the KWIC system (D.L. Parnas, 1972) when placed within a broader organizational setting of decision-making stakeholders and designers. This case study helped identify the need for an “intentional collective” modeling construct, a new kind of intentional actor, to support representing an organizational responsibility that is shared among multiple organizational participants. Examples of intentional collectives are organizations, organizational units, projects, teams and the like. Such a representational feature is not specifically supported by the existing i* modeling approaches, such as by (Castro et al., 2001; Grau & Franch, 2007; Kolp et al., 2001; Lucena, 2010; S. E. Yu, 2009).
Section 3.4.2 elaborates on the distinction between intentional collectives introduced in this thesis and aggregate intentional agents and positions supported by i*.

**Extensions to deal with architectural artifacts:** During the analysis of case studies it was noted that architectural design reasoning refers to software architectural artifacts, such as when arguing about alternative design approaches. While informal architectural design discussions typically did not require detailed models of architecture artifacts, it was found that defining a conceptual link between design reasoning and architectural artifacts is of advantage when the analysis requires more formality, such as to support the analysis with software tools.

For example, the Siemens case study (Chapter 7) investigated integrating qualitative and quantitative requirements during requirements modeling and analysis. To support an integrated analysis approach more systematically, a more formal approach to modeling architecture artifacts was identified as necessary, including a high-level representation of architectural structure and process artifacts created and evolved during architectural reasoning and decision-making.

**Extensions to reason with intentional roles:** The WML\(^5\) case study analyzed at the Mitel Corporation (Chapter 5) illustrates an expanded use of intentional roles to support structuring and generalizing architecture design responsibilities for later reuse. This extension also supports the reasoned application of such generalized design responsibility knowledge during subsequent architectural design efforts.

**Extensions to reason with intentional viewpoints:** The Phoenix insurance case study (Chapter 6) illustrates a need for new intentional viewpoints concept to represent distinct organizational responsibilities of stakeholders or designers that have some overlapping goals. In chapter 6 intentional viewpoints were used to represent the decision reasoning of an application designers and an enterprise wide architect related to a design choice. The application architect and the enterprise wide architect have distinct organizational responsibilities, however, in context of this particular design choice, they their responsibilities overlapped, and they reasoned differently about the same architectural choices.

**Extensions to deal with packaged solutions:** Industrial case studies indicated that designers often use terminology that compactly refers to well-understood design problem and solutions. Design patterns (Buschmann, 1996; Gamma et al., 1995) are well-known examples of such kind of terminology usage. To further explore representing and analyzing such kinds of “packaged solution” terminology used during architectural reasoning, a case study studies and analyzes the definition and application of

\(^5\) Wireless Markup Language—a hypertext markup language specification for mobile phones

34
design patterns to the architectural design of a commercial alarm system (the “TeleLarm AB” system) as reported in the literature (Chapter 7). This case study examined how the design intents discussed in design patterns and the repeated application of design patterns during system design can be represented and analyzed in terms of design intents and solution choices.

**Extensions to deal with integrated modeling and analysis of qualitative and quantitative NFRs:** The Siemens case study (Chapter 8) examined how to deal with qualitative and quantitative NFRs in an integrated manner during requirements analysis. During this case study, a specification approach for quantitative NFRs was developed and linked via scenario modeling, of high-level architectural structures and processes, to qualitative NFRs included in NFR goal graphs. This in turn supported an integrated modeling and analysis approach of both quantitative and qualitative NFRs in the control systems domain.

**Extensions to deal with terminology ambiguity:** During the Phoenix case study (Chapter 9) the problem of ambiguity in technical terminology was identified. Despite using well-known service-oriented architectural terminology (Erl, 2007; Josuttis, 2007), different architects attributed different operational meaning to terms. To clarify the meaning of technical terminology an approach was developed that adapted and extended an existing semiotic analysis approach (K. Liu, 2000; Stamper et al., 2003) to specifically deal with architectural terminology during architectural decision-making.

### 3.3. Comparing IAL to Approaches Applying i* to the Architecture Design

Using Table 3-2 the following section summarizes key differences between the IAL approach and two representative approaches that apply i* to architectural design (Grau & Franch, 2007; Kolp et al., 2001). Following is a brief discussion of each table row.

**Requirements actor**: A requirements actor represents stakeholders relevant during requirements analysis, usually during early requirements (E. Yu, 1994). IAL as well as the other two approaches support modeling and analyzing requirements stakeholder and their respective decision-making.

**Architectural actor**: Architectural actors represent designers during architectural design. IAL represents architects and designers in the development organization as intentional actors. The

---

6 While this section distinguishes between Requirements, Architectural and Development Organization Actors, this thesis, on the whole, does not make a distinction between those types of intentional actors. During modeling, a single type of actor appears without distinguishing between “Requirements” and “Architectural” design stages or between organizational and technical organizational participants.
architectural actor represents and delineates design responsibilities allocated to them. Kolp et al. propose that intentional actors represent autonomous computational components that are capable, during the system runtime, of goal reasoning and of the formation of alliances with other autonomous computational components. Grau et al. propose that intentional actors represent architectural components during design time.

**Development organization actor:** In IAL relevant stakeholders in the development organizations are represented using intentional actors, and included in the analysis. This includes stakeholders such as upper management, product management and the like. These stakeholders are part of the broader architectural or architecturally relevant decision-processes. Neither Kolp et al. nor Grau et al. include a representation of such stakeholders in their architectural models.

**Architecture actor decision scope:** In IAL architecture actors encapsulate design time decision-making about the design of architectural artifacts. Kolp et al. propose that architectural actors encapsulate autonomous computational decision-making during runtime to achieve stated runtime goals. In Grau et al. decisions are made by an anonymous actor during design time; intentional actors do not encapsulate autonomous decision-making, neither during design-time nor during run-time.

**Architectural goals:** The IAL perspective suggests interpreting goals included in architectural-actor models as design-time goals. Decision-making during design-time can however defer design-time goals to goal-achievement mechanisms during runtime. Kolp et al. view goals allocated to architectural actors as allocation of runtime goals that are addressed by autonomous computational entities during the runtime of the system. Grau et al. view architectural goals as design-time goals linked to architectural elements. Grau et al. differentiate between *architectural* goals, which indicate the purpose(s) for which components communicate with each other during runtime, and *intentional* goals, which indicate stakeholder expectations over runtime behavior of components. Intentional goals are attributed during design time to design time artifacts, and are addressed by an anonymous global designer during design time.

**Requirements to architecture dependencies:** For IAL as well as for the other approaches intentional dependencies originating from requirement actors are viewed as design goals during subsequent system design. IAL suggests directing these intentional dependencies towards development organization actors and/or architectural actor representing stakeholders and designers in the development organization.

Both Kolp et al. and Grau et al. view these intentional dependencies as elements of a requirement specification directed towards an anonymous designer who does not appear in the model. The anonymous designer is responsible for the design of the intended software system, and then decides
what architectural actors to define and how to further allocate intentional dependencies amongst architectural actors.

Grau et al. propose avoiding the use of intentional tasks dependencies when modeling dependencies between requirements-domain stakeholders and architectural actors; according to Grau et al.’s approach, intentional tasks represent prescriptive component behavior, which should not be restricted during requirements modeling behavior of architectural components.

**Architecture dependencies:** IAL views intentional dependencies amongst architectural actors as design expectations that architectural designers have of each other. Kolp et al. view intentional dependencies as runtime expectations amongst autonomous computational elements to support them in achieving computational results. Grau et al. see intentional dependencies amongst architectural actors as representing runtime communication goals amongst architectural components.

**Actor creation during architectural design:** This thesis proposes that development organization and architectural actors can create new development organization and architectural actors in response to design and decision-making. For example, where a design choice introduces new architectural artifacts, over which new design responsibilities are defined. This thesis thus proposes that architectural and architectural relevant decision-making is intertwined with the definition and allocation of responsibilities, including design responsibilities, in development organizations. The approaches proposed by both Kolp et al. and Grau et al. define an anonymous, and hidden, architectural designer who represents a global responsibility over the entire architectural design. Since this anonymous designer is responsible for the introduction of all new architectural design artifacts, the notion of intentional actors creating intentional actors is not applicable. It is thus not possible to indicate that within the decision authority of a development team, the team decides to introduce a new “sub team”, and allocate to that new team distinct design responsibilities. Such creation and allocation of responsibilities to a new intentional “sub” actor is typically of no concern to the higher level actors in the organization, and can thus be hidden from higher-level organizational stakeholders and designer. Chapter 5 illustrates such an example.

**Vertical architectural dependencies:** IAL supports defining architectural dependencies between the “creator” architectural actor and the “created” architectural actors. These dependencies are called “vertical” since they present a hierarchical organizational relationship between creator and created actors. While vertical dependencies are not essentially different from “horizontal” dependencies between intentional actors, it is useful to distinguish between “vertical” dependencies that are justified by higher-level organizational decision-making and “horizontal” dependencies that are defined at a
particular organizational level. Since the approaches proposed by Kolp et al. and Grau et al. do not support actors creating actors, the only vertical dependencies in their approach are those that are defined between the requirements level and the architectural level as a whole.

**Summary**: IAL defines a notion of a development organization and architectural actor that is socio-organizational and socio-technical. Such intentional actors encapsulate the autonomous design reasoning and decision-making of human designers who are responsible for the design of particular architectural and architectural relevant artifacts in organizational settings. In contrast, the other approaches equate intentional actors with technical concepts only, thereby removing the social-organizational meaning, and the uncertainty that comes with the inherent social relationship that exists between intentional actors in organizational settings. Instead, these approaches suggest a computational interpretation for intentional actors.

This thesis suggests that socio-technical modeling and analysis concepts are essential in dealing with the organizational nature of architectural decision-making in a development organization. Without assuming a socio-technical relationship between intentional actors, it is impossible to deal with the organizational implication of architectural decision-making, including the uncertainties that arise from architectural decision-making that is allocated amongst different organizational stakeholder and designers, each pursuing distinct motivations and interests as mandated by their respective organizational responsibilities.

**Table 3-1: Comparing IAL and two actor-oriented approaches applied to software architecture**

<table>
<thead>
<tr>
<th></th>
<th>IAL</th>
<th>Kolp et al.</th>
<th>Grau et al.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Requirement Actor</strong></td>
<td>Stakeholders / human decision maker</td>
<td>Stakeholders / human decision maker</td>
<td>Stakeholders / human decision maker</td>
</tr>
<tr>
<td><strong>Architecture Actor</strong></td>
<td>Human architect / designer</td>
<td>Autonomous component during runtime</td>
<td>Architectural component during design time</td>
</tr>
<tr>
<td><strong>Development Organization Actor</strong></td>
<td>Human stakeholder in a development organization</td>
<td>Not Available</td>
<td>Not Available</td>
</tr>
<tr>
<td><strong>Architecture Actor Decision Scope (i.e. what is the subject matter given to choice)</strong></td>
<td>Design features of architectural artifact during design time</td>
<td>Computational choices during runtime</td>
<td>Architectural actors do not make choices. Instead, anonymous global designer makes design time choices.</td>
</tr>
<tr>
<td><strong>Architectural Goals</strong></td>
<td>Design-time goals</td>
<td>Architectural actors have runtime goals. An anonymous global design</td>
<td>Design-time and runtime goals linked to architectural components</td>
</tr>
</tbody>
</table>
3.4. Core Language Meta-Model and Extensions

3.4.1. Core IAL constructs

**Intentional actor:** The *intentional actor*, or actor for short, is the central concept in the meta-model (figure 3-1). Adopted from i*, an intentional actor represents an organizational responsibility that is given, or known to have, some autonomy to decide how to perform design tasks to achieve design goals. This is the final interpretation that was chosen during this thesis research.

**Intentional role, intentional position, intentional agent:** The concepts of intentional role and intentional position, also adopted from i*, are abstractions of organizational responsibilities. In a development organization an intentional role is typically a functional-oriented design responsibility, such as dealing with the design of a “Claims Handler” component, or of a “Messaging Client”. However, organizational roles dealing with non-functional concerns can also be defined, such as “Messaging Security”. The definition of intentional roles can also be guided by the identification of design concerns (Daniel Gross & Yu, 2002; Stanley M. Sutton Jr. & Tarr, 2002; Tarr et al., 1999).
The intentional position concept supports defining organizational responsibilities that are composed of two or more intentional roles. A key characteristic of an intentional position is that it is assignable to a single human designer, who then assumes all responsibilities indicated by the intentional roles linked to that intentional position. Yu and Liu illustrate alternative compositions of intentional roles into intentional positions for analyzing different security and trust properties of alternative organizational systems configurations (E. Yu & Liu, 2000). This thesis uses such a similar approach for modeling and analyzing alternative architectural design approaches.

The intentional agent represents a human stakeholder or designer in the organization. By using an intentional agent, it is possible to represent the personal goals of stakeholders and designers so that they can be made part of the design reasoning. For example, when intentional agents are made to occupy a position, the design goals and reasoning associated with that position (usually via intentional roles the position covers) can be considered together with the individual goals of the intentional agent.

**Intentional elements:** In i*, the “intentional element” concept includes the intentional task, goal and softgoal, the claim (also known as argumentation goal), and the resource. IAL adopts these intentional
elements, but to fit them to architectural terminology renames them design task, design goal (hardgoal and softgoal), belief and resource (figure 3-1). Besides renaming, this thesis also reinterprets the meaning of intentional elements, to apply them to the architectural design domain as follows:

A design task is a possible action by a designer that commits to a particular design approach. A design goal represents a resulting design state of architectural artifacts. A hardgoal represents a resulting design state that is achievable in an absolute manner. A softgoal represents quality requirements that the designer aims to achieve and for which it is difficult, impossible, or undesirable to define clear-cut criteria of achievement. A resource is an entity (physical or informational) that must be available for a design approach to be implementable.

![Figure 3-2: Core meta-model – Intentional elements to support goal directed reasoning](image)

This thesis distinguishes between the design task performed by an intentional actor and the design artifacts that are designed by that actor. To indicate this distinction IAL includes an artifact concept that represents the design artifacts being designed by design tasks (included in figure 3-3). This...

---

7 This includes a resulting design state that supports a capability that achieves some informational states
interpretation of the design task is in contrast to the interpretation of an intentional task in i*, which conflates within the intentional task concept the designer action and the designed artifact. In i* an intentional task represents the task an intentional actor performs as part of a business process and the particular manner in which the intentional actor chooses to perform that task.

In IAL the concept of a design task is conceptually closer to the concept of design options in design rationale approaches (Conklin, 1989; Potts & Bruns, 1988) and the concept of operationalizing softgoals in the NFR Framework (Chung, 1993). The meta-model indicates this closeness by making the operationalizing softgoal a subtype of the design task. An operationalizing softgoal is a design task that can exhibit a more formal structure. This structure is further discussed in section 3.7, which presents the integration of qualitative and quantitative NFRs through scenario modeling.

IAL could have represent software system behavior using a concept such as intentional operations, somewhat analogous to i*, where an intentional task represents human work behavior. However, because during the case studies it was apparent that architects often decide about non-behavioral artifacts such as architectural styles, components, interfaces, infrastructure and the like, a broader interpretation was considered more appropriate.

Another point to consider is that while the “intentional element” terminology was chosen to fit with the architectural design domain, the concepts of design task and design goal in this thesis are understood more broadly to refer to decisions associated with non-technical stakeholders also, such as upper and product management. The term “design” implies the ability to make rationalized choices, independently of the particular design domain (business, organizational, requirements, product, architectural, or otherwise).

**Hardgoals, softgoals:** Drawing from i* and the NFR Framework, design goals are classified into hardgoals and softgoals. Hardgoals represent design goals that can be either fully achieved or not achieved at all, hence the prefix “hard”. Hardgoals usually represent functional design goals. Since, functional behavior can often be achieved in more than one way, hardgoals serve as the focal point for

---

8 The particular manner a task is performed is captured in i* by the intentional tasks along with the routine concept. A routine represents a series of task decompositions along selected sets of alternatives. Alternatives are indicated by goals that decompose tasks, and that are focal points for alternative tasks

9 The analogy is only partially correct since intentional actors in i* also perform human behavior, while architectural actors design behavior of system components. Other i* approaches that apply the intentional actor concept to the architecture domain make the analogy closer, by interpreting intentional actors as autonomous computational elements. These approaches however omit the stakeholders and designers who perform the design in the models.
alternative design approaches, each approach represented by a design task.

Softgoals refer to design goals that are difficult or impossible to quantify, or that are, for the purpose of design reasoning, better treated in a qualitative manner (Chung, 1993; Chung et al., 2000). Softgoals typically represent non-functional requirements, but they can also be used to represent business goals. The softgoal concept supports treating business goals and non-functional requirements as design goals that need to be addressed throughout the entire design process (Chung, 1993; Mylopoulos, Chung, & Nixon, 1992).

Figure 3-3 shows that a softgoal can be analyzed into two components: NFR Type and NFR Topic. The type and topic distinction was first introduced by Chung (1993). NFR type represents a property such as performance, security or modifiability, while an NFR topic represents the subject matter over which the property is applied. For example, NFR performance of catalogue searching could be analyzed into the Performance NFR type and the Catalogue Search topic. During requirements and architectural design, the NFR topics usually refer to already existing artifacts. Figure 3-3 indicates this relationship through the “refersTo” association link between NFR topic and Artifact, which represents a modeling artifact of interests.

Figure 3-3 also includes a prototype softgoal concept. A prototype softgoal represents a softgoal in general whose definition can be reused and included as separate softgoal in each intentional actor. For example, consider the softgoal “performance of system” included in both the Consumer component designer intentional role and the SOA evolution architect intentional role. Conceptually there are two design goals represented, one in each intentional role. However, both are derived from, and linked to, the same prototype softgoal: “performance of system”.

**Belief:** The belief element represents a domain assumption that supports or refutes a modeling element or a contribution between modeling elements (including other beliefs, that is, supporting or refuting contributions targeting beliefs). The belief element is adopted from both the i* and the NFR Framework, and supports including domain specific knowledge into the reasoning structure of intentional models.

**Design Task Decomposition:** Figure 3-2 shows that a design task can be decomposed into one or more intentional elements. “Task decomposition” helps reveal how design tasks differ with respect to their contribution to softgoals by considering their more detailed design approach elements. Intentional task decomposition is part of the intentional analysis offered by i*.

Also, adopting from i*, IAL supports decomposing a design task into one or more hardgoals. For example, the design task “use a client/server architectural style” can be decomposed into the hardgoal
“define client/server communication style” to indicate that choosing a client-and-server communication approach is an open design issue that requires a design approach. Additional hardgoals could indicate the need to identify design approaches for each of the client and the server components.

**Means-end links:** Another concept adopted from i* is the means-end concept. The “means-ends link” relates alternative design tasks to the hardgoal they achieve. It is also possible for a hardgoal to be a means for achieving a higher-level hardgoal, demonstrating how one design result can be achieved by one or more alternative lower level design results.

![Core meta-model - softgoal structure and artifacts (dotted links indicate subset constraints)](image)

**Softgoal contributions, operationalizing contribution, correlation links:** IAL also adopts the notion of “softgoal contributions” from i*. A softgoal contributes positively or negatively, and sufficiently or partially to a parent softgoal. Operationalizing contributions are contributions from design tasks to softgoals. Using operationalizing contributions helps indicate how design approaches contribute to softgoals. Capturing the different contributions of design approaches to softgoals is the basis for representing design tradeoffs during architectural design.

A “correlation link”, another type of operationalizing contribution, indicates an unintended contribution of a design approach on softgoals. These side-effects can be conflicting or synergistic (Chung et al., 1995; Chung et al., 2000).
Intentional dependencies: IAL adopts “intentional dependencies” from i*. An intentional dependency is a directional link from a “depender” actor to a “dependee” actor (figure 3-1). Each intentional dependency has an intentional element defined as a “dependum”. The intentional dependency represents an expectation that a depender actor has on a dependee actor, which is indicated by the dependum. The expectation can be for achieving a design goal, performing a design task, ensuring the availability of a resource, or addressing a softgoal in a sufficient manner. Intentional dependencies are a key concept in i* for modeling and analyzing how intentional actors collectively perform work in an organizational setting. By establishing dependencies on each other, intentional actors take advantage of the expertise and decision-making of each other. However, intentional dependencies also expose vulnerability, such as, for example, in the case when a dependee actor fails to fulfill the expectations of the depender actor (E. Yu, 1994; S. E. Yu, 2009).

Partition: In IAL the dependum defined for dependency between a depender actor and dependee actor is conceptually considered as being partitioned between two equivalent intentional elements – one residing within the depender actor and one within the dependee actor. Figure 3-4 illustrates this partitioning, and figure 3-5 shows the related meta-model elements. In figure 3-4 softgoal s1 defined from the depender to the dependee role in the top half of the diagram is conceptually included in both the depender and the dependee role, as indicated in the bottom half of the diagram. Since softgoal s1 was delegated by the depender role to the dependee role, softgoal s1 is a leaf goal in the depender and a top-level goal within the dependee role.

Translation: When intentional actors delegate design tasks, design terminology may change. A depender actor may utilize terminology in one domain while the dependee actor may translate the terminology to fit another domain. For example, a depender actor may refer to artifacts using
terminology from an insurance policy domain, while the dependee actor may translate some of the terminology into enterprise-messaging terminology. The “translation” concept in figure 3-5 represents the translation of terminology between intentional elements. Considering figure 3-4, softgoal \( s1 \) within the depender role could be named using terminology of the depender domain of expertise, while the softgoal included in the dependee role would be named using terminology of the dependee role\(^{10}\).

**Intents of dependencies:** Intentional dependencies also indicate for what purposes (intents) a depender actor decided to establish a dependency on a dependee actor (hence *intentional* dependencies). This is indicated by a link from an intentional dependency to the design tasks and design goals of the depender actor. As for the dependee, intentional dependencies establish justifications for the design goals that dependee actors aim to achieve, design tasks they aim to adopt, and resources they aim to make available.

![Figure 3-5: Partitioning Intentional Element into two Intentional Element concepts](image)

**Part-of, INS and ISA relationships:** These relationships between intentional actors are adopted from the \( i^* \) framework (figure 3-1). A “part-of relationship” indicates that one intentional actor is part of another intentional actor. The part-of relationship can for example indicate that the design responsibility over a component is part of a larger design responsibility over a subsystem. However, the actual intentional meaning of the part-of relationship in terms of design expectations is captured by

\(^{10}\) Another notion of translation could be in the case when terminology used to specify a goal is translated to a different terminology when specifying alternative design tasks to achieve goals.
intentional dependencies between the whole and its parts. The part-of relationship is only a syntactic indicator of such a relationship.

The part-of relationship between intentional actors supports representing goal delegation in organizations. For example, a part-of relationship can describe an intentional actor that serves as an “interface” for its constituent actors, as in the case of an insurance company actor that can serve as the source and destination for intentional dependences from client organizations, without “exposing” to the client organizations the component actor inside the insurance organization that will be dealing with the incoming dependency. For example, once intentional dependencies between the insurance company and the client organization are defined, a separate analysis step can deal with evaluating and deciding on the allocation of intentional dependencies to different component actors within the insurance organization.

An “INS relationship” represents an “instantiation” relationship between two intentional actors. Instantiation relationships support representing generalized responsibility knowledge. For example, organizational expectations on new-product development in general could be represented as a generalized intentional actor, and the design team responsible for developing a particular new product, “X”, would be linked by an INS link to such a new-product development actor. Intentional dependencies directed to or originating from the new-product development actor would likely apply to the new-product “X” actor. Application of dependencies is however open to judgment, and likely part of a negotiation process.

Finally, an “ISA relationship” indicates that one intentional actor is a specialization of another. For example, not unlike reuse-through-inheritance found in object-oriented modeling (Fowler & Scott, 2000), reasoning elements that are relevant to several organizational responsibilities can be abstracted into an abstract intentional actor. Other intentional actors can then inherit the reasoning structures, thereby reducing the complexity of the actor model. The ISA relationships can be used for structuring intentional actor diagrams, but it does not carry organizational meaning. Since it is a modeling abstraction, parent and child actors that are linked via an ISA link cannot have intentional dependencies defined.

Supporting i* and NFR Framework styles of reasoning: The core meta-model supports two styles of reasoning. The first is an NFR Framework style of reasoning (Chung, 1993; Chung et al., 2000) in which an NFR goal graph is constructed. The NFR Framework style places the softgoal as the focus of analysis. Softgoals are identified and refined until design approaches are identified that address the softgoals (“operationalizing softgoals” in NFR terminology). With this style of reasoning, softgoals are the focal...
points of analysis, while the functional requirements are dealt with separately using other modeling approaches, such as UML (Fowler & Scott, 2000).

The second style of reasoning is the \textit{i*} strategic rationale modeling style of reasoning (E. Yu, 1994). The strategic rationale style places functionality constrained by NFRs as the focus of analysis. Functionality can be represented by a design task. If the functionality can be designed in more than one way, the design task is decomposed into a hardgoal. If non-functional requirements constrain the design task, then these are also linked via task decomposition links to the design task. To represent the alternative design approaches, the hardgoal is decomposed into alternative design tasks, each linked by a means-end link to the hardgoal. Each design task is then linked by an operationalizing contribution link to softgoals, to indicate respective qualitative contributions.

### 3.4.2. Extensions to Deal with Intentional Collectives

The \textit{i*} modeling notation defines three types of intentional actors: role, agent, and position. The term “actor” refers generically to the other more specialized types of intentional actors. Yu explains (E. Yu, 1994):

A role is an abstract characterization of behavior of a social actor within some specialized context of domain of endeavor. Its characteristics are easily transferable to other social actors. Dependencies are associated with a role when these dependencies apply regardless of who plays the role.

An agent is an actor with concrete, physical manifestations, such as a human individual. We use the term agent instead of person for generality, so it can be used to refer to humans as well as artificial (hardware/software) agents\textsuperscript{11}. An agent has dependencies that apply regardless of what role he/she/it happens to be playing. These characteristics are typically not easily transferable to other individuals, e.g. its skills and experiences, and its physical limitations.

A position is intermediate in abstraction between a role and an agent. It is a set of roles

---

\textsuperscript{11} The IAL proposes to limit the concept of agent to represent human designers only, since the purpose of this study is to explore the intentional dimension attributable to the designers of artifacts. The intentionality of artifacts is then derived from the intentionality that guided the designers during decision-making. Artificial (Hardware and software) agents are represented as artifacts that are designed by humans. The human designers of an artificial agent would be directly or indirectly represented as an agent, role, position or other type of intentional actor.
typically played by one agent (e.g. assigned jointly to that one agent). We say that an agent occupies a position. A position is said to cover a role.

Yu further explains that roles, positions and agents can each have subparts, however:

Aggregate actors are not compositional with respect to intentionality. Each actor, regardless of whether it has parts, or is part of a larger whole, is taken to be intentional. Each actor has inherent freedom and is therefore ultimately unpredictable. There can be intentional dependencies between the whole and its parts, e.g. a dependency by the whole on its parts to maintain unity.

A key modeling question identified during some case studies was how to represent organizational units that are collective organizational responsibilities such as software development organizations, development projects, or teams. Several approaches are possible: a) let a composite intentional agent represent the collective and make other intentional actors its parts; b) let an intentional position represent the collective and make other intentional role or positions its parts; c) introduce a new concept of representing collective organizational responsibilities. In IAL the third option was chosen. Following is an explanation why the other two approaches were rejected for IAL.

**Why not composite intentional agents:** in IAL (and i*) the intentional agent represents a physical human decision-maker. Two or more physical human decision makers when joined together usually form an organizational collective with distinct responsibilities distributed amongst the members. Typically, a member of an organizational collective can leave the collective and others can join without changing the identity of the organizational collective. The identity of an intentional agent is, however, defined by the particular individual human being represented. Using a composite agent to represent a collective would require relaxing the definition of identity for the composite intentional agent, making the identity of the composite independent of the identity of the individual human members. Such a construction would essentially represent an organizational abstraction. It is therefore more consistent with the intentional agent definition to represent the composition of two or more intentional agents by a construct that represents an organizational abstraction.

**Why not intentional position:** The main reason for rejecting this choice for IAL is that in i* only one intentional agent is assigned to an intentional position to assume all its roles. In organizations, there are, however, examples of collectives that require relaxing this constraint. For example, in Chapter 5 a “WML
team” was characterized by a number of intentional roles, with each role assigned to different human stakeholder or designer in the organization.

![Figure 3-6: Intentional collective extension](image)

In i* it is also typical that the intentional position is fully defined by the assigned roles. Goals and reasoning is allocated to roles of a position and not to the position itself. In the case study in chapter 5 the reasoning of all WML team members was consolidated into one reasoning structure, to represent the reasoning of the team as a whole rather than of the individual responsibilities allocated to the team. The WML team is thus a collective where the individual roles do not fully define the collective. Such a “collective level” reasoning is typically not captured using the intentional position concept.

It thus appears a better choice for IAL to have intentional position concepts with crisply defined constraints, and hence introduced the intentional collective concept.

**Why not composite intentional positions:** One modeling question was however still outstanding. What construct to use to represent higher-level collectives such as the Mitel organization? The Mitel organization could in principle be represented by a composite intentional position, defined as an intentional position that is composed of two or more constituent intentional positions. While the constituent intentional position would keep to the aforementioned constraint, the composite intentional position would relax those constraints, such as allow assigning different intentional agents to its different constituent intentional positions.

However, there is also a need to represent a decomposition of intentional positions into component intentional positions, whereby only one intentional agent can be assigned to the whole decomposition structure.
Since both types of intentional position decomposition structures are needed, in IAL the choice was made to keep the constraints of the composite intentional position structure consistent with the constraints of an “atomic” intentional position, and introduce the intentional collective concept to represent aggregate organizational responsibilities that can have more than one intentional agent assigned.

**Why Intentional collective:** unlike IAL i* does not specifically distinguish between actors that represent human individuals or collectives of humans. Such distinctions are usually only made known in the naming of actors. IAL on the other hand uses the intentional collective concept to specify collectives and the intentional role, position and intentional agent to specify individual humans and individual organizational responsibilities. During architectural design making this distinction clear has the benefit that it indicates where the responsibility over an artifact can be further broken down into different orthogonal collectives or individual responsibilities, each pursuing distinct design objectives, priorities, reasoning and (socially constrained) autonomous decision-making. Having intentional collective concept helps make explicit and keep track of such possible decompositions of responsibilities.

Figure 3-6 shows that an intentional collective is a type of intentional actor, and that it can be composed of other intentional collectives, roles, or positions. From a modeling point of view, each intentional role or position linked to an intentional collective can have a distinct intentional agent assigned.

### 3.4.3. Extensions to Deal with Architectural Artifacts

This section discusses extensions that support linking intentional modeling and analysis to a conceptual representation of architectural artifacts. Figure 3-7 shows several conceptual extensions to the core meta-model for representing and linking between architectural artifacts and intentional modeling constructs.

**Design task introducing design artifacts:** Design tasks when selected introduce new design artifacts into the solution space. For example, a design task to use a client/server approach to structure information processing introduces into the design space client and server artifacts as well as artifacts representing the communication mechanism between them. The introduction of new design artifacts into the design space is indicated in figure 3-7 by the “introduces” association from DesignTask and Artifact. To simplify the model we assume that design tasks that change design artifacts in effect introduce new change artifacts.

**Artifacts as basis for defining softgoals:** Softgoals represent non-functional requirements (NFRs).
NFRs are often defined as qualifications over functional requirements or over existing, but incomplete, architectural structures. For example, the NFRs “modifiability of search functionality” and “security of purchase process” qualify the search functionality and purchase process functionality, respectively. According to Chung (Chung, 1993; Chung et al., 2000), such NFRs can be analyzed into “NFR type” and “NFR topic”. For example, “modifiability” and “security” are NFR types; “search functionality” and “purchase process” are NFR topics. Figure 3-3 illustrated the association from the NFR Topic component of softgoal concept to the artifact concept.

**Figure 3-7: Linking to architectural artifacts**

**Artifacts as a basis for defining hardgoals and design tasks**: Existing artifacts can be the basis for defining design tasks. For example, a design task can refer to an artifact, such as a Use Case Map (Buhr & Casselman, 1996), that represents a computational process across several architectural components. Having a design task refer to an artifact indicates that the artifact is incomplete and requires additional design and decision-making, usually to address as of yet unaddressed quality requirements. In figure 3-7 the refersTo association between a DesignTask an Artifact is inherited from IntentionalElement. Once a design task refers to an artifact, its further design is indicated by decomposing the design task into one or more design tasks and/or hardgoals, with each hardgoal representing a focal point for elaborating...
alternative design approaches.

It is also possible to have a hardgoal refer to an artifact thereby indicating that the design of the artifact involves two or more alternative design approaches.

**Concrete Artifact, Artifact Class:** Case studies showed that architectural design reasoning often involves references to classes of artifacts in addition to concrete artifacts (i.e. instances). For example, an architect who reasons about the performance of communication processes between a consumer component and a provider component—without referring to actual, specific consumer or provider components of the system under design—refers to “classes” of artifacts. In contrast, an architect who refers to the performance of the “submit new insurance policy” process refers to a specific, concrete, architectural artifact. In figure 3-7 ArtifactClass represents classes of artifacts, while ConcreteArtifact represents concrete artifacts. ArtifactClass classifies one or more ConcreteArtifact instances.

Modeling the relationship between artifact classes and concrete artifacts affords several benefits. One is the ability to construct models that refer to generalized architecture knowledge, thus supporting the reuse and selective application of architectural reasoning knowledge during new design efforts. Generalized reasoning knowledge can also facilitating the consistent application of design approaches, which promotes dealing with decision compliance during architectural design and implementation.

**Artifact type, notation:** Artifacts constructed during architectural design are typically part of a modeling notation. Thus, an artifact can be classified into some “type” that is associated with some “notation”. For example, a “new policy component” defined in a Use Case Map (UCM) diagram is a UCM “component” artifact type. Actor and goal-oriented modeling and analysis usually refer to artifacts belonging different architectural notations. ArtifactClass supports classifying ConcreteArtifacts in a manner relevant to the actor- and goal-analysis effort that is orthogonal to the classification provided by the modeling notation.

**Artifact relation, artifact relation type, artifact relation class:** ArtifactRelationType represents the type of relationship(s) a notation defines between artifacts. For example, a Use Case Map is defined by a sequence of computational responsibilities, and the UCM notation defines both “Use Case Map” and “computational responsibility” artifact types with an “is composed of” artifact relation type between them. At the instance level, a particular UCM event and responsibilities, such as “ProcessSensorEvent” and “RetrieveEventData”, are captured as concrete artifacts with concrete artifact relationships between them. “ArtifactRelationClass” classifies artifact relations according to criteria relevant to actor- and goal-oriented analysis.

Capturing the relationships and relationship type amongst artifacts allow the analysis and refinement
of softgoals. The relationships amongst artifacts are often affected by the manner softgoals can be refined into sub-softgoals. For example, a performance softgoal that defines a quality property for a UCM, can be refined into performance softgoals for each computational responsibility included in the UCM; such a refinement step is guided by the definition of relationship types for the UCM notation.

**Intentional actor creating intentional actor, Responsibility Assignment:** A design task that introduces a new artifact can specify the allocation of the subsequent design of the artifact to an existing or newly created intentional actor. The **Responsibility Assignment** concept (figure 3-8) links between the design task, the created artifact, and the intentional actor responsible for the subsequent design of the artifact. Visually, this relationship is rendered by a “new” link from the design task to the intentional actor made responsible for the newly introduced artifact (e.g. figure 5-11). Unlike the goal-oriented KAOS approach (Dardenne et al., 1993) the assignment of responsibility to an intentional actor implies the expectation that the intentional actor will aim to address design goals through decision-making.

![Responsibility Assignment Diagram](image)

**Figure 3-8: Responsibility Assignment**

The **Responsibility Assignment** concept is also used to indicate in strategic dependency diagrams the artifacts that intentional actors are made responsible for and the intentional actors assigning the responsibility. Both uses are complementary. In a strategic dependency diagram, the Responsibility Assignment concepts supports broadly indicating what actor is responsible for an artifact, and what
actor assigned the responsibility. During strategic rationale modeling the specific design task can be identified that introduced the artifacts into the design space, and that represents the responsibility allocation task of the depender intentional actor to the dependee intentional actor.

3.4.4. Extensions to support Representing and Applying Generalized knowledge

The i* modeling framework supports representing and analyzing different strategies to allocate roles to positions (L. Liu & Yu, 2001). This thesis extends the idea to include reasoning about and applying generalized knowledge during individual design efforts by a) representing generalized knowledge about architectural design responsibilities, and b) by reasoning about the application of generalized knowledge to develop models of concrete architectural design responsibilities. In this thesis, generalized architectural responsibility knowledge captured in a model or a model fragment is referred to as a “generalized organizational architecture”, while a model or model fragment representing concrete architectural design responsibilities is called a “concrete organizational architecture”. Concrete organizational architectures instantiate generalized organizational architectures. Figure 3-8 illustrates generalized and concrete organizational architectures using intentional roles. While the following illustrations focus on intentional role, other intentional actor types can also be used when generalizing and instantiating organizational architectures.

In figure 3-9, intentional roles A, B, B1, and B2 are part of a generalized organizational architecture, while the roles E and D are part of a concrete organizational architecture. To indicate that roles are generalized they include a “<g>” prefixed in their name. According to figure 3-9 the role B is composed of two organizational responsibilities: roles B1 and B2. During the instantiation of the generalized organizational architecture, it was decided that B1 and B2 is amalgamated into one organizational...
responsibility represented by role \( \mathbf{D} \).

The “\(<\mathbf{g}>\)” prefix essentially indicates a classification. The “generalization” terminology is used instead of a “classification” terminology (e.g. role class, and the use of a “\(<\mathbf{c}>\)” prefix), since the application (instantiation) of generalized knowledge is less about instantiating a class, such as providing values to defined properties, and more about the mapping of incoming and outgoing intentional dependencies from generalized actors to the instantiated ones.

Figure 3-10 illustrates an alternative role allocation. Here, roles \( \mathbf{D}_1 \) and \( \mathbf{D}_2 \) are instances of roles \( \mathbf{B}_1 \) and \( \mathbf{B}_2 \). The incoming softgoal dependency, \( \mathbf{s}_1 \), is allocated to role \( \mathbf{D}_1 \), indicating that the organizational responsibility of dealing with \( \mathbf{s}_1 \) is now made part of role \( \mathbf{D}_1 \), instead of role \( \mathbf{D} \). To further indicate restrictions of allocation, the intentional modeler can replace the intentional role \( \mathbf{D} \) with an intentional position or an intentional collective. Replacing role \( \mathbf{D} \) with an intentional position indicates that “component” role \( \mathbf{D}_1 \) and \( \mathbf{D}_2 \) are to be allocated to the same designer, whereas, replacing role \( \mathbf{D} \) to an intentional collective indicates that roles \( \mathbf{D}_1 \) and \( \mathbf{D}_2 \) may be allocated to different designers. Keeping role \( \mathbf{D} \) as an intentional role defers this decision.

Figure 3-11 illustrates a concrete organizational architecture that instantiates two organizational reference architectures. Here, role \( \mathbf{D} \) is an instance of both roles \( \mathbf{B} \) and \( \mathbf{X} \). Role \( \mathbf{X}_1 \) is amalgamated into role \( \mathbf{D}_2 \), while role \( \mathbf{X}_2 \) is instantiated into new role \( \mathbf{F} \). The dependency link \( \mathbf{s}_2 \) could be linked to role \( \mathbf{D} \) or role \( \mathbf{D}_1, \mathbf{D}_2 \) and/or role \( \mathbf{F} \). Note that Role \( \mathbf{Y} \) is not generalized, and can thus not be instantiated. Instead, Role \( \mathbf{Y} \) is part of the generalized organizational architecture, and needs to be included in a concrete organizational architecture and the depender role.

Such aforementioned allocation of roles and dependencies requires decision-making, and suggests concrete alternative organizational solution architectures.

*Building a concrete organizational architecture from a generalized organizational architecture:* The
construction of a concrete organizational architecture can be seen as a series of modeling actions. For example, to construct the concrete organizational architecture in figure 3-9 includes performing the following modeling actions: creation of Role E; linking Role E to Role A with an instantiation link; creation Role D; creation of three instantiation links from Role B to Role D, Role D to Role B1 and Role D to Role B2; creation of intentional dependency from Role E and Role D with a softgoal dependum s1; creation of an instantiation link between this newly created intentional dependency and the intentional dependency from Role A to Role B with the softgoal dependum s1 -- note that this latter instantiation is not visually shown in the figure.

![Figure 3-11: Allocating intentional roles (example 3)](image)

Since modeling actions are usually performed in response to design choices, each one or more modeling actions can be associated with a design task. The meta-model in figure 3-12 shows this association. In figure 3-12 a design task is associated with Knowledge Instantiations. Each instances of Knowledge Instantiations groups together one or more items of Knowledge Instance. A Knowledge Instance represents one of the following modeling actions: the creation of an Actor (Actor creation), the creation of an actor relation between two Actors (Actor relation creation), the creation of a dependency between two Actors (Dependency creation), and indication that one Intentional Dependency is an instantiation of another intentional dependency (Dependency instantiation).

The meta-model in figure 3-12 further shows that Actor creation is associated with the Intentional Actor that is created; the Actor relation creation is associated with an Action Relation; the
Dependency creation is associated with one Intentional Dependency; while the Dependency instantiation is linking a parent Intentional Dependency to a child Intentional Dependency. Note, however, that Actor creation is a design task associated with the instantiation of knowledge, rather than with the decision of an actor to create a new organizational design responsibility. The latter is indicated by the Responsibility Assignment concept in figure 3-8.

![Diagram](image)

**Figure 3-12: Allocating intentional roles meta model**

Finally, figure 3-12 also shows that Knowledge Instantiations is also associated with one or more Knowledge Instantiations. This supports grouping knowledge instantiations into composite modeling actions, each accomplishing a higher level-modeling step. For example, the creation of role E and the instantiation link created from role E to role A can be indicated as a composite instantiation action to instantiate role A with role E.

**Mapping constraints:** To ensure that concrete organizational architectures are constructed there are mapping constraints defined. For example, one constraint states that only generalized intentional actors can be instantiated (intentional actors whose isGeneralized property is set to True). Other constraints govern the use of Is-Part-of link in conjunction with instantiation. For example, an intentional dependency can only be drawn between a depender and a dependee actor who either directly or indirectly instantiate from corresponding generalized depender and dependee actors. For example, in figure 3-11 the intentional dependency between role E and role D1 is valid because role instantiates the
corresponding depender role A and role D1 instantiates via its Is-Part-Of link to role D the corresponding dependee role B.

**Capturing organizational architectural models:** The meta-model in figure 3-12 includes the organizational architecture model concept, which supports capturing a configuration of intentional actors included in a model. Several models can be so captured, each at a different level of abstraction. Instantiation links between models define a generalization ordering between models.

**Concern reasoning agents:** An application of intentional roles to the aspect-oriented domain, called “concern reasoning agents”, was developed during another case study to represent and reason about multiple concerns during architectural design (Daniel Gross & Yu, 2002). The term “concern reasoning agents” was adopted to align intentional role terminology with terminology used in the aspect-oriented research community (Stanley M. Sutton Jr. & Tarr, 2002; Tarr et al., 1999).

Concern reasoning agents are intentional roles with the added constraint that they are linked to a design artifact under design. The notion of concern reasoning agents is thus more specific than the notion of intentional roles in that it specifically supports dealing with the integration of design choices across different modules in architectural structures. From an organizational perspective, concern reasoning agents can be seen as representing a matrix style of organization (Galbraith, 1971, 2002) where some organizational responsibilities that require specialization are allocated to cut across other, more functional-oriented, responsibilities. Concern reasoning agents thus offer support for representing and analyzing multiple reporting structures that occur in matrix organizations with responsibilities over the design of cross-cutting architectural structures and processes. A summary of the “concern reasoning agent” case study is included in the Appendix A.

### 3.4.5. Extensions to Reason with Intentional Viewpoints

During the Phoenix case study (Chapter 6), overlapping organizational responsibilities were observed during design reasoning and decision-making. One architect was responsible for an enterprise application, while an enterprise-wide architect was responsible for all the enterprise applications combined. Each architect reasoned about the design choice from a different viewpoint. This thesis proposes the use of “intentional viewpoints”, a specialization of the intentional role concept, to represent the reasoning of organizational participants with overlapping design responsibilities. This extends the current notion of intentional roles that represents non-overlapping responsibilities only.

**Intentional viewpoint, intentional agent, intentional viewpoint group:** The meta-model in figure 3-13 extends the core meta-model by adding the elements of Intentional Viewpoint and Intentional...
**Viewpoint group** and linking these to the intentional agent concept. The intentional agent represents a human decision-maker or a coherent group of decision makers who are assigned design responsibilities. To represent overlapping responsibilities at least two intentional viewpoints are defined, both belonging to the same intentional viewpoint group. Each intentional viewpoint is assigned one intentional agent. One meta-model constraint included in figure 3-13 ensures that the same intentional agent cannot be assigned to different intentional viewpoints belonging to the same intentional viewpoint group. The other meta-model constraint included in figure 3-13 ensures that overlapping hardgoals are included in the intentional viewpoints belonging to the same intentional viewpoint group. This ensures that all viewpoints reason about the same design matter at hand.

Given two hardgoals g1 and g2: they are considered overlapping if g1 is in some way included in g2. For example the following (non-exhaustive) list of inclusion criteria can be considered: if g1 is an instance of g2; if g1 is a subtype of g2, if g1 is part of g2; if g1 is an implementation of g2.

An exact definition of overlapping is usually open to the interpretation of the modeler, since IAL does not provide a semantic for the labels of goals. However, in case a more formal link between goals and

![Figure 3-13: Meta-model fragments to represent intentional viewpoints](image-url)
related architectural artifacts is defined, and if the notion of goal in relation to the artifact is defined (such as temporal state achievement goals) then the artifact notation and semantics could allow defining more formal overlap relationships between goals.

Usually the same alternative design tasks are included in all intentional viewpoints belonging to the same intentional viewpoint group.

3.4.6. Extensions to Deal with Packaged Solutions

The NFR Framework’s modeling and analysis facilities supports dealing with design options that address a single NFR. However, during design discussions in case studies, architects often referred to coarser-grained design approaches. Design patterns (Buschmann, 1996; Gamma et al., 1995) are a typical example of such coarser-grained solution approaches.

A “packaged solution” is a design task that refers to higher level reasoning structures that can be “zoomed into” to reveal a decomposition structure of design tasks that are more detailed, and with their own links to the design goals. Designers can refer to these packaged solutions by name, without needing to explain the detailed design reasoning anymore. The meta-model in figure 3-14 extends the core meta-model using a packaged-solutions concept that supports representing coarser-grained solution approaches.

![Figure 3-14: Packaged solutions meta-model fragment](image)

A packaged solution includes design goals, suggests possible solutions approaches, and offers a preferred solution approach that best balances the stated design goals (forces in pattern terminology).
Figure 3-14 shows that a packaged solution is a type of design task that typically includes (i.e. is composed of) two or more design tasks. The preferred association between the packaged solution and the design task indicates which design task is preferable.

Packaged solutions can also be characterized by the set of high-level softgoals they come to address and to trade-off. Finally, packaged solutions can be associated with a hardgoal to indicate the overall design functionality that the packaged solution offers. This hardgoal is derived from a higher-level design task. In pattern text descriptions, this hardgoal/higher-level design task often appears in the intent section. For example, “to make remote system status locally available” could be part of a functional intent described in a design pattern.

3.4.7. Extensions to Deal with Integrating Qualitative and Quantitative NFR Reasoning

Both the i* and the NFR frameworks focus on dealing with qualitative NFRs. However, beside qualitative NFRs, software architects often must deal with quantitative NFRs, such as measurable response times, costs, and other quantifiable system properties. This section proposes concepts for specifying quantitative NFRs, as well as dealing with both qualitative and quantitative NFRs in an integrated manner during architectural design reasoning and decision-making. The extensions were developed while studying guidelines for analyzing NFRs for a reusable application platform. The application platform in particular was required to support the development of scalable applications in multiple problem domains that could be deployed in the form of an embedded system, or as small, medium, large, or very large workstation-based systems.

The meta-model in figure 3-15 shows concepts for specifying quantitative NFRs. The concepts are adopted from Fowler’s measurement analysis pattern (Fowler, 1997). Consider, for example, the definition of the quantitative NFR: “Maximum throughput per server shall support 3000 alarms per second”. The QuantitativeNFRType in figure 3-15 represents the term “Maximum throughput per server”. The unit alarm per second is represented by the CompoundUnit concept with “Alarm” SimpleUnit as the nominator and “Second” SimpleUnit as the denominator. The number “3000” is captured as ScalarQuantity. Finally, the quantitative NFR itself is an instance of the QuantitativeNFR concept that links together all the aforementioned definitions.

Figure 3-16 extends the meta-model to support standardizing terminology and metrics for NFRs across different domain applications and different deployment scenarios. A “domain” represents a business or problem area for which a “domain application” is developed. For example, building security
or train safety are two typical domains. Although domains are distinct, a development organization that
develops applications for different domains often has common underlying technological know-how and
software assets for developing their domain applications. Each domain application in turn can be
deployed using different configurations. For example, an alarm system can be deployed as either a small
embedded alarm system for small apartments, as a larger alarm system for larger buildings, or as a
multi-site alarm system for securing several buildings on a larger premise.

![Diagram of specifying quantitative NFRs](image)

Figure 3-15: Specifying quantitative NFRs

A key distinction included in figure 3-16 is the differentiation between two quantitative NFR types:
**LoadConditionType** and **DeploymentParameterType**. A load condition type indicates a quantitative
requirement that specifies a processing load that a software system must be capable of supporting.
“Maximum throughput per second” and “Change of values per second” are two typical examples.
According to figure 3-16, the maximum load condition throughput per server for the small deployment
configuration of the “TT” domain application needs to support “3000 alarms per second”, is
conceptually defined as follows: A **LoadConditionType** “Max alarm throughput per server” is defined,
which is associated with the **CompoundUnit** “Alarm per second”. The compound unit is itself defined as
a ratio between the simple units “alarm” and “second”. A scalar quantity of “3000” is linked to the
“alarm per second” unit. A token **QuantitativeNFR** instance, representing the quantitative NFR is linked
to the “Max alarm throughput per server” **LoadConditionType**, and is associated with the “3000”
“alarm per second” scalar quantity. Finally, the quantitative NFR is associated with one
**DeploymentConfigurationType**, such as “small system deployment” of a particular domain application
(e.g. “TT”).

63
A deployment parameter type indicates a quantitative requirement that specifies a configuration of a system, such as the number of clients connected to the system, or the number of measurement points in a control system.

Once defined, deployment parameters are used to specify the different deployment configuration types of the domain application. For example, the deployment configuration for the domain application “TT” could be specified as using 30 clients and 100 measurement points; or 3000 clients and 5000 measurement points; or 30,000 clients and 100,000 measurement points. Each deployment configuration is defined by one or more deployment parameters.

The meta-model in figure 3-16 indicates that load condition types are specified using compound units. Deployment parameter types could, in principle, be specified using a simple and a compound unit, but usually only simple units are used. While each quantitative NFR type can have more than one unit defined, only one unit is used when specifying a quantitative NFR.

Unit conversions help standardized quantitative NFR types expressed in different units. Standardizing quantitative NFRs across different domains and deployment configurations is particularly important in organizations that aim to create reusable software system frameworks or assets. Figure 3-16 shows concepts to define quantitative NFRs, to unify their specification across different domain applications.
and deployment configurations, and to standardize them for the development of common software assets.

**Linking quantitative to qualitative NFRs**: scenario models serve as intermediary to linking the analysis of qualitative and quantitative NFRs. A “scenario model” represents an architectural design solution in terms of high-level architectural structures, processes and processing steps. To support integrating the analysis of quantitative and qualitative NFRs scenario models are on one hand parameterized by quantitative NFRs, and are on the other hand constructed and/or embellished by selections of design task (or operationalizing softgoals) in goal graphs.

The integrated analysis of quantitative and qualitative NFRs thus broadly involves the following steps:

1. The definition of a baseline scenario model parameterized with quantitative NFRs
2. The calculation of baseline processing capacity needs using the baseline scenario model bound to actual load conditions and deployment parameters (quantitative NFRs).
3. Selection of one or more design tasks (or operationalizing softgoals) to address qualitative NFRs
4. Embellishing of the baseline scenario with additional processes and process steps; each selected design task has scenario templates defined that are used to embellish the scenario models
5. Recalculation of processing capacity needs using the embellished scenario models, again bound to the same actual load conditions and deployment parameters.
6. Evaluation of the calculated processing capacity needs usually in terms of hardware/software and infrastructure costs. If costs are too high, then some quantitative and/or qualitative NFRs must be adjusted to reduce the projected processing capacity needs of the system.
   a. Reduce load conditions or reduce deployment parameters – thereby positioning the system at a lower end in the market place, in terms of processing ability.
   b. Select operationalizing softgoals that require less overhead processing (i.e. less embellishing of the baseline scenario models) – thereby reducing the processing capacity needed to meet the specified quantitative NFRs

Figure 3-17 illustrates a sample scenario model (Chapter 8 includes a more detailed explanation of this model). The curved line represents a process and the “x’s” on the curved line represent process steps that perform some computation. Rectangles represent architectural system structures. The filled dot on the left edge of the process represents the starting point.
The starting point is associated with a load condition, such as number of value changes per second. Selected processing steps can be associated with data processing activities, which are in turn parameterized by deployment parameters. For example, the “Notify tag value changed” process step in figure 3-17 is performed for each client known to the system. The number of clients known to a deployed system is specified as a deployment parameter.

By analyzing the quantitative requirement specifications of load conditions and the deployment parameters, a scenario model can be produced to calculate the number of processing steps required, as well as the processing capacity needed to meet the stated quantitative requirements. In other words, the purpose of a scenario model is to estimate the throughput capacity required of a system to meet the stated load conditions, given a set of deployment parameters. This estimate allows calculating the cost of the system in terms of hardware capacity requirements.
Using a scenario model helps link the analysis of qualitative and quantitative NFRs linked as follows: the link between qualitative NFRs analysis and scenario model analysis is defined through “design tasks”. A design task is, on the one hand, chosen to address qualitative NFRs. On the other hand, it introduces additional scenario elements into a scenario model. Figure 3-18 illustrates how a design task is linked to a scenario model via an annotation. The “WS” (web services) annotation is added to a process step to indicate that the process step needs to be expanded to include additional scenario elements. For example, in figure 3-18 the PS (process step) was previously annotated as WS, meaning that before the PS process step is performed, it is expanded, not unlike a macro-expansion, into additional architectural structures and process steps.
Each design task in a qualitative goal model has its own scenario “fragment” defined. By selecting different design tasks, different scenario fragments are incorporated in a baseline scenario model, generating new scenario model variations. At the same time, each selection of design tasks also selects qualitative NFR tradeoffs.

Figure 3-19 shows in a conceptual manner this interplay between goal graphs and scenario modeling. The relationship is defined via “operationalizing softgoals”, a concept drawn from the NFR Framework.

**Operationalizing Softgoal** corresponds to a design task and is composed of **Operationalizing Type** and a **Topic**. An “operationalizing type” is the name of a design approach, such as “web services”, while a “topic” corresponds to either a concrete artifact within a model, or an artifact class that classifies one or more concrete artifacts over which the design approach is applied. For example, the operationalizing softgoal “web services [notify tag value changed]” comprises the operationalizing type “web services” and the topic, a concrete artifact, “notify tag value changed”.

The operationalizing type is in turn associated with a scenario model template (a scenario model with an isTemplate attribute set to “true”), which indicates how the topic can be expanded with additional...
process steps and structures.

**A combined qualitative and quantitative analysis method:** Figure 3-20 provides an overview of the modeling and analysis process steps using an IDEF0\(^{12}\) process representation. The series of steps conveyed in the diagram represents the flow of input and output data among method steps. The process model is not meant to prescribe a specific order of method steps. A fuller description of the analysis steps is provided in Chapter 8.

![Figure 3-20: IDEF0 diagram of Analysis method steps overview](image)

The first step is about standardizing terminology and metrics for specifying quantitative and qualitative NFRs. Only when NFRs specifications are comparable across different domains and application it is possible to perform goal graph modeling and goal-and-scenario analysis.

Several levels of capacity analysis are outlined below.

**Basic capacity analysis:** On this level, scenario-oriented analysis assumes the same architectural structures and processes across the control system applications and the application platform. This level of analysis thus simplifies the structure and processes of individual domain applications, and disregards platform specific processing overhead. This level of analysis offers first-cut estimations of processing capacity needs for the application platform.

**Intermediate capacity and platform overhead analysis:** This level of analysis still assumes uniform architectural structures and processes across all the domain applications, but allows for additional overhead structures and processes for the application platform. Although assumptions about structure and processes of individual domain applications are simplified, results are somewhat adjusted to include platform-specific overhead introduced by platform-specific NFRs. This level of analysis offers more

---

\(^{12}\) IDEF0 stands for Integration Definition for Function Modeling
accurate capacity estimations.

**Detailed capacity and overhead analysis:** At this level of analysis process and goal models are developed for each domain application. Architectural structures and processes are further justified by NFR goal graphs, which establish the qualitative NFR tradeoffs for each domain application.

The process and goal models developed for each domain application are consolidated via systematic migration steps to develop a global process and goal model for the application platform. During migration steps NFR are tradeoffs to accommodate NFRs of the platform. Once the platform processes and goal models are completed, processing capacity analysis can be performed by discounting or transferring domain-specific architectural processes and structures to platform-specific overhead structures.

This level of analysis is the most time consuming. However, it enables the most accurate capacity estimations when structures and processes of individual domain applications exhibit significant differences both in processes and structures and in of specified NFRs.

**Variability analysis for the runtime:** This analysis focuses on identifying platform structures and processes (including platform services), which can be added, removed or exchanged during the system runtime to support dealing with varying deployment and load conditions. For example, under normal load conditions all platform services could be activated, while under burst load conditions, some platform services could be deactivated so that additional processing capacity is made available. This would however incur specific qualitative NFR tradeoffs that need to be acceptable, given the circumstances.

### 3.4.8. Extensions to Deal with Terminology Ambiguity

All modeling and analysis involves the use of specialized terminology. Often, terminology is defined informally, leaving much room for ambiguity. Often also, the modeling itself is considered a means to clarifying the meaning of terms; however, relying on modeling to specify terms risks making specification commitments that unnecessarily constrain the meaning of terms (K. Liu, 2000; Stamper, 2006).

This section overviews an approach to specify terminology that complements modeling approaches such as UML (Fowler & Scott, 2000). The approach borrows from works in organizational semiotics. The key concepts taken are the semiotic agent and ontological dependence schemas, which are adapted to support specifying the meaning of terminology used to describe design approaches. A fuller description is included in Chapter 9.
Figure 3-21 illustrates meta-model elements to support semiotic-agent modeling and to linking design tasks to semiotic-agent modeling elements. A “semiotic agent” is classified into substantive and symbolic semiotic agents, and a modified semiotic agent. A “substantive semiotic agent” refers to a human who interprets semiotic affordances. The “symbolic semiotic agent” refers to a symbolic computational entity.

A “modified semiotic agent” is recursively defined as the composition of a semiotic agent and one or more affordances. The recursive definition of a modified agent supports composing modified agents from other modified agents. The “modifies” association between the Substantive Agent and the Semiotic Agent represents the “modifies” links between agents, which supports selectively composing higher-level semiotic agents from lower level ones.

“Affordances” are classified into entity affordances, which refers to physical or informational entities; role affordances which refer to the role subsequent affordances play with respect to semiotic agents; and action and operation affordances which refer to the actions and operations that a substantive agent and a symbolic agent, respectively, are enabled to perform given all the preceding affordances. Finally, design tasks are associated with semiotic agents. Design tasks that refer to entities, actions and operations are linked to the respective modified agents. For example, to clarify the meaning of the design task “publish to specific provider”, the design task is associated with Modified Agent 7 in table 3-2.
Table 3-2: Semiotic composition structure to define operational term

<table>
<thead>
<tr>
<th>Semiotic agent</th>
<th>Ontological Composition structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent 1</td>
<td>Physical point-to-point integration designer</td>
</tr>
<tr>
<td>ModifiedAgent1</td>
<td>Agent1, Data</td>
</tr>
<tr>
<td>ModifiedAgent2</td>
<td>Agent1, Physical Connection</td>
</tr>
<tr>
<td>ModifiedAgent3</td>
<td>{ModifiedAgent2, Agent1}, Communication Device</td>
</tr>
<tr>
<td>ModifiedAgent4</td>
<td>ModifiedAgent3, [Sender]</td>
</tr>
<tr>
<td>ModifiedAgent5</td>
<td>ModifiedAgent3, [Recipient]</td>
</tr>
<tr>
<td>ModifiedAgent6</td>
<td>{Agent1, ModifiedAgent3}, Physical Device Address</td>
</tr>
<tr>
<td>ModifiedAgent7</td>
<td>{ModifiedAgent1, ModifiedAgent6, ModifiedAgent4, ModifiedAgent5}, Transmit data from Physical Sender Address to Physical Recipient Address</td>
</tr>
</tbody>
</table>

A comma represents the composition of a semiotic agent with an affordance and defines a modified agent. The curled brackets indicate two or more semiotic agents that, together, are composed with an affordance.

The Modified Agent 7 definition in table 3-2 indicates the operational meaning of “publish to specific provider”, by using ontological dependences to preceding affordances. Meaning is captured by the operation “Transmit data from Physical Sender Address to Physical Recipient Address”, and the semiotic compositional context ({ModifiedAgent1, ModifiedAgent6, ModifiedAgent4, ModifiedAgent5}) that determines the necessary affordances required to defining the operation.

Analyzing and linking affordances using ontological dependences helps identify the essential elements that define specialized terminology. One modeling approach is to start with the main design theme indicated by a term and to “assign” a substantive semiotic agent to that theme. Then main operations associated with this theme are identified, together with affordances that are considered ontologically necessary to exist for the operations to be defined.

3.5. Conclusion

One key objective of this thesis research was to identify limitations, if any, of a core IAL to model and
analyze architectural decision-making in organizational settings. This chapter presented extensions to
the core IAL that were identified during case studies, to support dealing with specific decision-making
needs, for which core IAL was considered insufficient.

Using meta-model extensions to the core IAL meta model, this chapter presented in detail the
proposed IAL extensions and explained why each extension was necessary. Given the small number of
case studies, the proposed extensions are considered first-cut. Additional case studies are needed to
more firmly establish the utility of the proposed extensions to architectural decision-makers in
organizational settings.
4. IAL Applied to the KWIC System Literature Case Study

4.1. Introduction

In this chapter reviews the architectural design of the Key Word in Context (KWIC) system (Chung, 1993; D. L. Parnas, 1976) and illustrates a hypothetical organizational setting in which a KWIC component is designed to meet business goals. The main modeling and analysis features illustrated in this chapter include:

- Representing stakeholders who are the source for business goals that guide architectural design and decision processes;
- Representing the propagation of business goals as requirements (in particular non-functional requirements) from source stakeholders, through organizational stakeholders and designers, and towards KWIC component architectural decision makers;
- Representing the reasoning of architectural designers, during the design of components, when handling conflicting non-functional requirements that originate from different stakeholders and designers in the development organization.

Section 4.2 introduces the design discussion related to alternative architectural design approaches reported in the literature, and section 4.3 illustrates how these are placed in a broader hypothetical organizational context. Section 4.4. responds to the research questions in light of this illustrative case study, and section 4.5. discusses the modeling method. Finally, section 4.6 concludes this chapter.

This chapter revises and extends the original publication (Chung et al., 1999) in several ways. First, the organizational modeling included in the original publication used intentional actors to represent organizational stakeholders and designers, in this chapter intentional collectives and intentional roles are used to represent organizational units, making the actor models more precise. Second, instantiation links are used between intentional collectives to illustrate generalized organizational responsibilities. Finally, the organizational models are expanded with additional intentional collectives and intentional dependencies. References are also updated.

---

4.2. Description of Case Study

To illustrate the benefits of information hiding as a modularization criterion, Parnas and collaborators proposed a software architecture for the Keyword in Context (KWIC) system. The architecture design decisions that are likely to change hides in modules (D.L. Parnas, 1972). Equating modules to the “Abstract Data Type architectural style,” Garlan and Shaw (1994) compared and contrasted this style with three additional architectural styles—Shared Data, Implicit Invocation, and Pipe-and-Filter—and presented the benefits and liabilities for each architectural style (in terms of non-functional requirements) in table form (table 4-1).

Using a goal- and process-oriented modeling and analysis approach, Chung et al. illustrated a more systematic and fine-grained approach to developing and analyzing the NFRs and solution alternatives (Chung et al., 1995). The goal-graph in figure 4-1 illustrates how an initial set of non-functional requirements such as **Modifiability**, **Extensibility** and **Performance** were put up-front as design goals, and then refined into more detailed design goals until specific solution approaches that address these design goals were identified.

<table>
<thead>
<tr>
<th>Table 4-1: KWIC Comparison of Solutions Approaches (Garlan &amp; Shaw, 1994)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Data</td>
</tr>
<tr>
<td>Change in Algorithm</td>
</tr>
<tr>
<td>Change in Data Representation</td>
</tr>
<tr>
<td>Change in Function</td>
</tr>
<tr>
<td>Performance</td>
</tr>
<tr>
<td>Reuse</td>
</tr>
</tbody>
</table>

For example, **Modifiability (System)** is identified as a high-level goal. It is then decomposed into **Modifiability (Process)**, **Modifiability (Data Representation)** and **Modifiability (Function)**.
Modifiability (Function) is then further refined into Extensibility (Function), Updateability (Function) and Deletability (Function). Once these refinements are identified, they are linked to the different architectural solution alternatives.

The knowledge captured in the goal-graph is generic and not specific to the KWIC component. This is indicated by the “general” labels used in representation of NFRs, such as System, Process, Data Representation. However, by binding the generic labels with the data, function, and process elements of a particular KWIC component, the design reasoning is applied to the design of a concrete KWIC system or component.

Figure 4-1: Goal graph to support selecting among solution alternatives (Chung et al., 1995)

The knowledge captured in the goal-graph is generic and not specific to the KWIC component. This is indicated by the “general” labels used in representation of NFRs, such as System, Process, Data Representation. However, by binding the generic labels with the data, function, and process elements of a particular KWIC component, the design reasoning is applied to the design of a concrete KWIC system or component.
4.3. Placing Architectural Design Discussions into an Organizational Setting

Figure 4-1 shows several design goals such as Performance and Reusability that have been given priority, as indicated by the exclamation mark. An architect usually selects a design approach that best addresses prioritized design goals. In a business organizational setting, the prioritization of design goals is generally derived from the priorities of the business goals the organization is pursuing. An analysis of business goals leads to design goals, and the prioritization of business goals therefore justifies the prioritization of the design goals, and subsequent design choices (P. Clements & Bass, 2010; R Kazman & Bass, 2005).

To ensure that the architectural design of the KWIC component meets business goals, we turn to the stakeholders whose goals need to be met, and to the stakeholders and designers in the organization who responsibility is to meeting the goals. The diagram in figure 4-2 depicts design and decision responsibilities in a hypothetical organizational setting using different types of intentional actors, including one with a design responsibility for designing the KWIC component. An organizational responsibility is characterized by the goals, tasks, and resources that the person assuming this responsibility is required to achieve, perform, and make available, respectively.

Several actor types are included in the diagram. A circle with a curved line on the bottom is an intentional role representing an abstract organizational responsibility, usually to perform some organizational function. The clover-leaf shaped symbol with the “<collective>” label is an intentional collective representing a collective organizational responsibility.

Dependency links between the different intentional actor types indicate the work and deliverables that the dependee actor expects the dependant actor to fulfill or deliver. Dependency links are directional with the little “D” indicating the direction, the back of the “D” indicating the dependee and the front of the “D” the dependant.

According to figure 4-2, the e-shopping vendor depends on the e-App development business for the development of an e-catalogue application. The design task dependency, depicted by a hexagon-shaped dependum, indicates the request of the vendor to the developer. Since the request is represented by a design-task dependum, there is an implication of some hard constraint, in this example, that the catalogue must be web based (hence an e-catalogue), rather than any other kind of catalogue implementation. The request by the e-shopping vendor to have an e-catalogue application delivered that offers good value for money paid is indicated by the Good value for money softgoal. The softgoal indicates that “good value” is essentially subjective, and open to negotiation, but not
necessarily optional.

Figure 4-2: Organizational context for an e-catalog application and the KWIC architect/development

Turning to the intentional actors that are part of the development organization, the e-Apps development business expects the e-catalogue architect to develop an e-catalogue application architecture, and expects the e-catalogue development to deliver the e-catalogue application. A goal dependency, represented by a rounded rectangle between the e-Apps development business actor to the e-catalogue architect, indicates that the business organization essentially gives the e-catalogue architect freedom in the design of the architecture. This request is indicated by a hardgoal dependency link indicating that the request does not specify any specific approach or constraint. However, this freedom is somewhat qualified by the softgoal dependency defined in parallel that requests Conformance with Vendor requirements. As mentioned earlier, since the conformance request is represented using a softgoal, it is an indication that conformance is negotiated and not absolute.

The resource dependency from e-Apps development business to e-catalogue development indicates the request to deliver the e-catalogue application, and the design task dependency indicates the request to develop the application according to the architectural specification.

The e-shopping vendor has specified several business goals to the e-catalogue architect. These are
shown as softgoal dependencies between the e-shopping vendor actor and the e-catalog architect actor. The e-shopping vendor has further indicated which of the goals are of higher priority. These are visually indicated by softgoal criticalities: single or double exclamation marks.

Continuing in figure 4-2, the e-catalogue architect decides how to address the e-shopping vendor’s business goals. While the “internal” decision-making of the e-catalogue architect is not shown, some resulting decisions can be gleaned from the dependency links leading from the e-catalogue architect to the KWIC architect. The fact that the e-catalogue architect decided to rely on the development of a KWIC component indicates that indexing is an important feature in addressing the e-shopping vendors’ goals – likely due to performance requirements for search functionality. The e-catalogue architect further translated the business requirement for ease of use into interactivity, the need to support multiple vendors into modifiability, and cost efficient into the requirement for performance.

Since the internal design reasoning of the e-catalogue architect is not shown, it is not indicated whether the e-catalogue architect decided to introduce a KWIC component into the design space, or decided to direct new design requirements to an already existed component. In the former case, the e-catalogue architect would not only have decided on using the KWIC component to address some relevant design goals, but would also have introduced the responsibility to design a KWIC component into the development organization. This new organizational design responsibility is then represented by the KWIC development intentional collective.

In figure 4-2, the is-Part-of links syntactically indicate that the e-catalogue architect, the e-catalogue development and the KWIC development actors are organizationally part of the e-Apps development business. The specific meaning of the is-Part-of relationship can only be derived from the intentional dependencies defined between the composite parent actor and the component children actors (E. Yu, 1994).

Figure 4-2 illustrates a rather straightforward organizational setting, linking business goals articulated by the e-shopping vendor to the non-functional goals allocated to KWIC development. However, the web of interdependencies between stakeholders and designers is usually much more complex. Figure 4-3 illustrates a more complex organizational setting. In figure 4-3, NFRs originate from the e-shopping vendor, from the Man\textsuperscript{14} Admin intentional collective, as well as from the Reuse Manager role.

\textsuperscript{14} Man, standing for Unix Manual Pages system
Figure 4-3: Organizational context for advancing a reuse requirement
Furthermore, a Help file system architect mediates between the business goals articulated by the Man Admin and KWIC development. To make reusability enforceable, the organization requires that all architects and designers obtain approval for reusable design from the reuse manager. This is indicated by the Reuse Manager role who expects reusability, and giving reusability high criticality, and who can enforce this expectation by providing or not providing Approval to different organizational stakeholders.

According to figure 4-3, the KWIC architect evaluates the alternative design choices in light of the NFR design goals that originate from the different organizational stakeholders and architects. The architectural reasoning of KWIC development has thus become more complex. In figure 4-4 is a goal graph illustrating the design reasoning of the KWIC development actor. The goal graph includes the design goals from the different higher level designers, the different origins of which are indicated by the “h”, “e” and “r” letters, denoting the help file system architect, the e-catalogue architect, and the reuse manager, respectively.

By studying figure 4-4, it may seem that the pipe-and-filter architectural style might have been a
better choice for the help file system architect, or that the abstract data type would have been a better choice for the e-catalogue architect. However, due to the criticality of the reusability requirements, the implicit invocation architectural style appears the most justified choice for KWIC development.

Returning to figure 4-3, the model further illustrates two organizational development projects—a help file system development project, and an e-catalogue development project—both of which are included in the same organization. Figure 4-3 shows that both are instantiations of a generalized System Development responsibility. This is indicated by including and linking the generalized System Development intentional collective via INS links to the e-catalogue development and help file system development intentional collectives. That System Development is a generalized responsibility is indicated by the “g” in the <g, collective> label.

By using such organizational structuring, the general organizational expectations for development projects in general can be indicated in an actor model a generalized manner, which can then be funneled down to define part of the expectations over particular projects.

4.4. Responding to Research Questions

This section discusses the modeling and analysis illustrated in this chapter in light of the research questions presented in section three. Since the case study is a hypothetical example based on architectural decision-making reported in the literature, the answers to the research questions are based on observations of the models produced. In particular, the manner IAL modeling constructs were put to use to illustrate the hypothetical example.

Q1.1. What role do business goals play during architectural decision-making?

This case study illustrated how business goals can drive architectural decision-making in development organizations. Stakeholders and designers, such as the e-catalogue and help file system architect, in the development organization interpret, translate and refine business goals into non-functional requirements, which are then further interpreted, refined and addressed by the designers of a component. Business goals serve as the ultimate justification for choosing among alternative design options. Negotiating architectural approaches ultimately requires negotiating the business goals put forward by clients and the development organization.

For example, the KWIC component designer appealed to the interactivity NFR to argue for the
implicit invocation architectural approach. This NFR is in turn derived from the ease of use softgoal put forward by the e-shopping vendor, which, according to the e-catalogue interpretive design work, justifies the existence of the interactivity NFR. If architectural choices require trading of interactivity with other NFRs, then the e-catalogue architect may need to renegotiate the ease of use softgoal with the client.

This case study also illustrates how the prioritization of business goals leads to the prioritization of NFRs, which in turn guides the architectural decision-making task. Prioritizations of goals are propagated, via criticalities defined on softgoals, through the network of dependencies between stakeholders and designers in the organizations and guide decision-making.

This case study further illustrated that business goals do not only originate from client organizations, but also from the development organization itself, such as the reusability goal illustrated in this case study. These are similarly interpreted and translated by organizational participants to guide design decision-making. Architectural designers often need to trade-off organizational goals with the goals of clients, and aim to find a balance between keeping clients satisfied and ensuring the viability of the development organization as a business.

**Q.2.1. What modeling constructs are appropriate for making business goals and the role they play during architectural decision-making explicit?**

Softgoals appeared to be appropriate for representing business goals. Intentional dependencies from client stakeholders to stakeholders in the development organization supported having business goals serve as a source for deriving NFRs. Criticality attributes associated with softgoals indicated the priority client stakeholders attributed to business goals. These criticalities were then propagated in the organization across NFR softgoals, which in turn guided the evaluation of tradeoffs during architectural decision-making.

Intentional dependencies that link business goals via intentional actors to NFRs help indicate the social nature of goal achievement in the development organizational setting, pointing to focal points of negotiation between clients and stakeholders, and stakeholders and designers in the development organization.

Given the social nature of the relationship between stakeholders and designers, business goals may not sufficiently be addressed or not addressed at all during architectural design. The degree of achievement is part of the direct and indirect negotiation processes between the clients and the
organizational stakeholders and designers in the development organization.

**Q.1.2. What role do qualitative NFRs play during architectural decision-making to meet business goals?**

NFRs serve as a link between business goals and architectural design deliberations. Architects often find that NFRs can be synergistic or in conflict with each other (Chung et al., 1999; Chung et al., 1995). This is usually the case when NFRs originate from different organizational stakeholders, but also when they originate from a single stakeholder. NFRs when traced back to business goals provide the basis for negotiating architecturally relevant requirements. This supports arriving at solution approaches that address important business relevant NFRs.

**Q2.2. What modeling constructs are appropriate for making qualitative NFRs and the role they play during the architectural decision-making to meet business goals explicit?**

Softgoals in conjunction with the modeling constructs offered by the NFR framework supported dealing with qualitative NFRs during architectural decision-making to help meet business goals. Key decision-making tasks supported include a) clarifying the meaning of NFRs b) prioritizing NFRs, c) dealing with NFR tradeoffs during the exploration of architectural design choices, d) placing NFR tradeoff knowledge in context of specific domain knowledge e) linking NFRs to business goals.

Most importantly, this case study illustrated that dealing with NFRs is a decision-process in a broader organizational context, in which different organizational participants deal with and contribute to addressing of business goals.

**Q1.3. What role does the organizational setting play during architectural decision-making to meet business goals?**

This case study illustrated some of the roles that an organization plays during architectural design to meet business goals. Organizations set up organizational responsibilities to negotiate business goals with different clients, and translate them to non-functional requirements. Organizational roles are also set up to mediate the negotiations between clients and developers in the development organization. Organizations also set up organizational roles to deal with the organizations own business goals and
negotiate these with developers. These illustrations concur with studies reported in the literature, such as the observation that in organizations knowledge and decision-making is distributed (B. Curtis et al., 1988), and that organizations are established so that stakeholders and designers can utilize their specialized knowledge to deal with different knowledge intensive tasks required to achieve the organization’s goals (Rasmussen et al., 1990).

Evaluations of opportunities and vulnerabilities also play a role when considering architectural design in an organizational setting. While organizational participants seek others to help contribute to the achievement of goals, stakeholders and designers also evaluate whether delegated work will be achieved. This includes determining the motivations and interests of contributing stakeholders and designers to see if they align with their own, as well as monitoring work in progress, to ensure favorable outcomes (Farenhorst & van Vliet, 2009).

For example, the help file system architect may consider whether to depend on KWIC development to deliver the KWIC component is really an opportunity, for example, when design goals such as UNIX compliance are not forthcoming. If this requirement is important enough, the architect may consider developing alternative version of the KWIC component in spite of the reuse requirement.

Ultimately, the organization’s clients evaluate and constantly monitor whether their important business goals are going to be met through the system that is developed by the development organization. The opportunity and vulnerability evaluation performed by depender actors thus applies to organizational participants in the development organization and to the organization as a whole.

**Q2.3. What modeling constructs are appropriate for making the organizational setting and the role the organizational setting plays during the architectural decision-making to meet business goals explicit?**

This illustration utilized intentional actors and intentional dependencies to represent organizational responsibilities and the allocated of responsibilities to organizational stakeholders and designers. Different actor abstractions, such as intentional collective, intentional role and position, represent organizational responsibilities at different levels of abstraction.

Intentional dependencies support representing expectations intentional actor have on each other to achieve goals, perform design tasks or provide or make available resources and deliverables. Incoming dependencies help explain the design goals that intentional actors address, as well as understand the origins of conflicting demands put on organizational decision-makers. Outgoing dependencies help
indicate what a depender actor expects others to achieve.

Intentional actors support representing client organizations, development organizations, as well as organizational participants such as management, architectural designers and development teams and individuals. Intentional actors together with intentional dependencies therefore supports representing a rich network of interrelated interests and demands amongst stakeholders and designers that serve as context and provides guidance during architectural design and decision-making.

Q1.4. What role do quantitative NFRs play during architectural decision-making to meet business goals? Q2.4. What modeling constructs are appropriate for making the role quantitative NFRs play during the architectural decision-making to meet business goals explicit?

This literature case study did not deal with quantitative NFRs.

4.5. Modeling Discussion

Capturing design reasoning: In this case study design reasoning was captured using modeling constructs drawn from the NFR Framework (Chung, 1993; Chung et al., 2000). An alternative approach to capturing design reasoning is to use the Strategic Rationale modeling approach offered by the i* framework (E. Yu, 1994; S. E. Yu, 2009). The modeling method presented in this thesis supports utilizing either of these approaches, depending on whether the modeler wants non-functional or functional requirements as the center of analysis.

This case study expands earlier work by Chung et al. in which the NFR framework modeling approach was used to analyze the design reasoning of the KWIC system (Chung et al., 1995). The NFR goal graph that Chung and colleagues developed to capture KWIC reasoning structures was adapted and put into an organizational setting.

Knowledge structuring and reuse: Using knowledge structuring such as ISA and INS links between intentional actors enables making the models more compact, as well as allows the reuse of model fragments during future modeling efforts. For example, the System development intentional collective Knowledge and its incoming dependencies could be reused in another model. Structuring that makes use of INS links reflects how organizational participants could be structuring organizational responsibilities. For example, the “System Development” collective in figure 4-3 abstractly represents the organizational responsibilities that the organization places on all development projects. This would
include development methods, quality assurance, reporting responsibilities, and the like. These expectations are then applied towards new development projects, and join the project goals and constraints in addition to project-specific design goals. Knowledge structured for reuse may thus also represent how stakeholders and designers structure the problem domain also.

Exposing actor-internal reasoning during modeling: In this case study, the only reasoning explored was that included in the KWIC development collective. A fuller analysis could explore the internal reasoning of other actors and capture these using goal graphs (or strategic rational models). Such broader analysis would help gain further insight on the effect of the decision-making behavior of the different actors on each other.

4.6. Conclusion

Achieving NFRs during architectural design is a key objective in architecture-based approaches to software engineering. However, it must be remembered that NFRs are developed and dealt with in a larger organizational context, and are derived from business goals through an organizational negotiation process. Stakeholders and designers in organizations must often mediate between stakeholders whose business goals are addressed, and architectural designers who are asked to address NFRs.

This chapter outlined a systematic approach for representing, analyzing and addressing quality requirements during architectural design in an organizational setting, using intentional actor and dependency concepts, along with a goal-oriented architectural design reasoning approach. Using this organizational modeling approach, an analysis of the KWIC component architecture in terms of NFRs when placed with a broader development organizational setting was illustrated.

This chapter further illustrated how the intentional collective concept can represent larger organizational units such as development teams or a development organization as a whole. During this case study, the intentional collective concept was used to model the organization as a whole and the KWIC development team was descriptive in nature, indicating an organizational responsibility that encompassed others. No particular reasoning, such as the need to deal with conflicting goals emerging from lower-level organizational stakeholders and designers, was attributed to the intentional collective.
5. IAL Applied to the “WML” Industrial Case Study

5.1. Introduction\textsuperscript{15}

This chapter reports on the architectural design of a commercial telephone switching system and the design reasoning of suggested architecture modifications during an architectural evolution effort. The modifications were discussed by a team of stakeholders and designers in the development organization tasked with enhancing the switching system to introduce internet browsing capabilities on the system’s telephone sets.

At the beginning of this case study, the thesis author learned that the team had been discussing architectural design options back and forth, as how to introduce needed functionality into the system, without coming to a clear resolution. This led to the suggestion to present to the team the design arguments using a modeling notation to facilitate the decision-making. This thesis researcher was given one week to produce these models.

During this week members of the WML team were interviewed to identify the key design arguments they brought in favor of and/or against architectural options. Interviews, which typically lasted one hour, were recorded and relevant portions were transcribed and modeled. Once models were produced, they were presented to the full team, who referred to the models to consolidate their discussions. The team members had no prior expose to the modeling notation used. At the end of the presentation, the team adopted one of the design approaches. Incidentally, the team did not share the decision adopted with the researcher, possibly due to the proprietary nature of the system.

This chapter replicates some of the modeling and analysis included in Chapter 4. The main modeling and analysis features illustrated in this chapter include:

- Representing stakeholders who are the source for business goals that guide architectural design and decision processes;
- Representing the propagation of business goals as requirements (in particular non-functional requirements): from source stakeholders, through organizational stakeholders and designers,

and towards architectural decision makers;

- Representing the reasoning of architectural designers during the design of a new telephone feature when handling conflicting non-functional requirements that originate from different stakeholders and designers in the development organization;
- Representing the establishing and evolution of organizational design responsibilities in response to architectural decision-making;
- Representing generalized organizational design responsibilities associated with reusable architectural structures and processes as well as the systematic application of such generalized and reusable structures to create particular architectural design responsibilities, structures and processes.

The next section introduces the case study. Sections 5.2.1 and 5.2.2 respectively illustrate how the source of business goals and NFRs as well as the reasoning put forward by the development team responsible for the new features are represented and analyzed. Section 5.2.3 illustrates generalized organizational design responsibilities and how these are applied to support architectural and organizational design. Sections 5.3. discusses the research questions in light of the case study finding, while sections 5.5 and 5.6 discuss methodical points and conclude the chapter.

This chapter revises and extends the original publication (Daniel Gross & Yu, 2001a). Instead of a generic intentional actors, this chapter utilizes the intentional collective to represent design responsibility, expands on the organizational context in which the WML team is placed, and revises and expands on intentional role modeling to support generalizing and applying architectural design in organizational settings.

5.2. **Description of Case Study**

Mitel, a multinational telecommunications company, initiated a project, called the “WML” project. The key objective of this project was to introduce internet browsing capabilities to telephone sets of a major business telephone exchange product (an enterprise PBX). The call control subsystem, key subsystem of the telephone exchange, was the focus of this development effort.

Traditional call control systems are designed using a centralized call control architectural style: telephone sets connected to the call control subsystem are “dumb” terminals and all call control

---

16 WML stands for Wireless Markup Language a markup language interpreted by browsers on mobile device.
functionality is performed and managed centrally by the call control subsystem.

Today’s trend in the telecom industry is to move to a decentralized call control architectural style, where telephone functionality is located in “intelligent” telephone devices. Indeed, the WML project took place during an enterprise-wide effort to evolve the telephone exchange and all related subsystems and auxiliary products (such as telephone sets) from a centralized to decentralized architecture.

Figure 5-1 shows the main architectural elements of a centralized call control subsystem. The diagram replicated here was hand drawn by the call control subsystem architect during an interview. The diagram shows that the call control subsystem is responsible for all aspects of a telephone session: establishing calls, call forwarding, call waiting, terminating calls, and other call-related functions. All such call functionalities are implemented by (instances of) the Phone process in the call control subsystem. In addition, setting up call features, such as configuring call forwarding on the telephone set, is implemented by the Setup process within call control.

The I/O handler in the User State manager subsystem interprets signal data sent from the telephone to check if the user entered telephone setup commands, or initiated or otherwise managed a telephone call, and activates and deactivates phone and setup processes accordingly. The Device activity map holds data about different types of telephone sets, which is used by the I/O handler to support its data interpretation processes to correctly interpret data coming from the different types of telephone sets supported by the switching system.

Figure 5-1: A telephone system architecture
The **Peripheral**, a dedicated hardware device, connects proprietary telephone sets to the switching system. The **Virtual peripheral** is a software application on standard PC hardware that emulates a peripheral device. The virtual peripheral connects next generation "intelligent" telephone sets to the switching system. Intelligent telephone sets are connected through a standard IP-based network to the virtual peripheral.

Turning to internet browsing on mobile devices, figure 5-2 shows key architectural elements implementing WML browsing capabilities on a client device. The client includes a WML browser, which connects via a WAP gateway to a Web Server.

According to figure 5-2, in order to offer internet browsing on telephone sets connected to the switching system requires adding WML browser software to the telephone sets. However, to be consistent with the centralized call control architectural style, the WML browser must be implemented as a phone process in the **user state processes** subsystem. On the other hand, following the enterprise-wide effort to move to a decentralized call control architectural style suggests placing the WML browser into the new IP telephone sets\(^\text{17}\). A third choice, a compromise between the two others would be to implement a WML browser process within the virtual peripheral\(^\text{18}\). Figure 5-3 illustrates the choices in a visual manner.

**Figure 5-2**: WML/WAP reference architecture

During this case study three types of models were created: a) an organizational model that depicted

\(^{17}\) It was decided early on that the regular telephone sets would not be upgraded to support WML browsing. This decision was not part of the design discussions during the case study and was not modeled and analyzed.

\(^{18}\) Since regular “dumb” phones were excluded from this project, the “regular” peripheral did not need to support the addition of browser software.
the participants in the WML project and their goals, b) a strategic rationale model that consolidated the collective design reasoning that members of the team put forward, when explaining the different options and tradeoffs, and c) an organizational architectural model that represented architectural design responsibilities corresponding to different architectural design choices.

Where to place the *Client* in the telephone system?

![Diagram of WML browser component placement](image)

**5.3. Placing Architectural Design Discussions into an Organizational Setting**

**5.3.1. Representing Stakeholders and Designers Who Are the Source of Architectural Design Goals**

During the case study the following organizational participants were interviewed: the architect of the call control subsystem, the architect of the IP telephone sets, the enterprise-wide architect responsible for strategic architectural directions, the manager responsible for customer products, the manager responsible for new business strategy, and the marketing stakeholder, as well as the industrial designer responsible for interface design of telephone sets. The thesis author recorded the interviews, and then transcribed relevant portions of the interview and constructed actor- and goal-oriented models. The
design options were then discussed with all of the aforementioned people during the presentation of the models to the WML team.

The model in figure 5-4 represents the WML team and its organizational participants. Also, included are the respective business and system goals that the organizational participants pursue. At the center is the WML team represented as an intentional collective to indicate that the WML team is an aggregated organizational responsibility composed of other responsibilities. Intentional roles represent the organizational participants assigned to the WML team to indicate the roles the participants played in the WML project. This assignment is represented by the Is-Assigned-To links from the intentional roles to the intentional collective.

The NFRs that the WML team needs to pursue are derived from these business and system goals. An initial assessment of these goals suggests some conflicts. For example, a main system goal of the call control architect is to maintain the architectural integrity of the centralized call control architecture, while the enterprise-wide architect’s objective is architectural evolution, which requires changing the architecture to exhibit new kinds of structures and processes. Such conflicting goals lead to conflicting demands on the WML project team.

Figure 5-5 expands on the organizational context and includes upper management and the Mitel organization as a whole as well as dependencies between different intentional actors. The Upper Management responsibility is represented by an intentional position since during this study upper management appeared as a coherent decision-maker. Also included are two intentional roles that are part of upper management: the Next Generation Systems and the Portalization of telephones. These represent decision-tasks that Upper management took on. The first relates to the design of next generation systems where upper management decided to develop IP-based systems, such as IP phones.

The figure also shows some related softgoal dependencies from the IP based systems design task to the IP phone architect. These include the open system, architectural evolution, and the reduce time to market softgoal dependencies. As indicated in the model in figure 5-4 these are the softgoals that the IP phone architect put forward to the WML team.

---

19 Organizational participants interviewed discussed business and system goals in relation to the WML project. Some of the goals mentioned were quite specific to the WML project, while others were more general. For example, maintaining the architectural integrity of a call control system is a system goal relevant to many other projects, while fast and easy access to internet services is specific to the WML project. In this case study, no distinction between goals were made to indicate their different scope. In future work, goals could be classified or allocated to intentional actors or roles representing different aggregate levels.
The softgoals included in the intentional roles in figure 5-4 are thus derived from “incoming” intentional dependencies that originate from upper management. Some of these softgoals are directly included while others are refinements of the “incoming” softgoal, and/or are additions that the participants suggested given their knowledge expertise in their respective area.

For example, Architecture strategy is expected to design open systems, that reduce time to market, and that make use of off-the-shelf components. The Architecture Strategy further identified the need to
work towards **architectural evolution**. To simplify, the model in figure 5-5 does not show all intentional dependencies originating from upper management to the different intentional roles, nor does it show all softgoals included in the intentional roles in figure 5-4.

Figure 5-5 further shows that the Mitel organization depends on upper management for achieving business objectives such as to develop **competitive products**, **reduce development time and cost**, and **increase sales**. To address the **increase sales** goal upper management, in its role to deal with the Portalization of telephones, decided to introduce **internet browsing on telephone sets**. According to figure 5-5, upper management hopes that this contributes to **increase customer base**, which in turn contributes to the **increase sales** goal. To implement this decision, upper management established the WML team, and directed via a design goal dependency the key design requirement **internet browsing**.
appliance and WML infrastructure to the WML team. Note that upper management appeared to have made the choice to make use of the WML/WAP technologies in establishing the WML team and stipulating the design goal constraint that required the use of a WML infrastructure. In figure 5-5 this design constraint is captured by the intentional design task dependency from upper management to the WML team. During this case study this particular choice was taken for granted and was not explained.

This request to design internet browsing capabilities and WML infrastructure together with the various goals expected from and derived by the organizational participants define the broader organizational context in which the WML team discusses its design approaches and makes its decisions.

5.3.2. Representing and Analyzing Architectural Design Solutions and Tradeoffs

Since the WML team was a coherent organizational unit, the design argumentation and goal reasoning mentioned by the team participants was consolidated into one strategic rationale model (figure 5-6). The top-level softgoals in the model are derived from the stakeholder and designer softgoals included in figure 5-4. The model also indicates the highest-level design task of the WML development team to design a WML based service infrastructure, represented by the design task, WML based service creation infrastructure. This design task is a refinement of the more general task internet browsing appliance and infrastructure requested by upper management and focuses the design discussions on a key design question at hand.

To indicate the alternative design approaches the WML team considered the design task is decomposed into the main sub-goal: service creation infrastructure be WML based. The WML team identified three alternative design approaches: a) a master-controlled WML based infrastructure, which is the centralized architectural design approach; b) a shared controller based WML infrastructure, which is more decentralized and shares some of the control responsibility with a centralized control process; and c) an appliance-based WML infrastructure, which is fully decentralized (three means-ends links in the bottom middle of figure 5-6).

Further analyzing the second design approach, the design team identified two additional design alternatives: a stateless shared and a stateful shared controller design approach. This additional design alternative is indicated by decomposing the design task shared controller based WML infrastructure into a design goal, and linking the two additional design approaches via means-end links to the new design goal (two additional means-ends links in the bottom of figure 5-6).

The stateless shared controller WML infrastructure design approach introduces new controlling
functionality related to WML browsing outside of the centralized call control subsystem, but keeps state management functionality within call control. The **stateful shared controller WML infrastructure** design approach reallocates all controlling functionality to outside of the call control subsystem, including state management.

The goal graph in figure 5-6 also shows how the alternative design approaches are linked to softgoals, indicating their contributions (positively, negative, and partially) to different non-functional design goals. These contribution links allowed the WML team to engage in a more systematic evaluation of the alternative design approaches. For example, contribution links indicate that placing the WML Browser within the virtual peripheral (the shared controller approach) contributes positively to the
architectural evolution goal (namely the ability to provide “evolved” state manager components to future switching systems). Adding a stream interpreter component (making the controller stateless) adversely affects the performance of telephone sets attached to the system, and increases the likelihood of processing errors. This is due to the difficulty of interpreting streamed data within the virtual peripheral since the full meaning of streamed data is only known to call control (this additional argumentation could have been added to figure 5-6 as a belief, that supports the negative contribution; to not further clutter the diagram it was omitted in figure 5-6).

5.3.3. Evolving Architectural Responsibilities Linked to Architectural Decision-making

Choices that stakeholder and designers in organizations make often require establishing new organizational responsibilities or changing existing ones. At Mitel, upper management created the WML team and its related design responsibilities after deciding to introduce internet browsing capabilities on telephone sets. The WML team in turn considers architectural choices for dealing with the design details, the design of which may introduce additional design responsibilities or change existing ones.

The organizational responsibilities modeled in this and the next section were not directly observed or discussed with stakeholders or designers during the case study, but were derived by extrapolating from the research data collected. Some of the responsibilities included here were presented to the WML team during the presentation and feedback session, while others were added while reviewing and extending the modeling included in the published paper (Daniel Gross & Yu, 2001a).

Establishing the WML team collective: Figure 5-7 illustrates some organizational responsibilities that are related to the introduction of the WML project. These include the Mitel intentional collective that is decomposed through “Is-Part-of” links to a) upper management, and b) a responsibility for the PBX lines of businesses. The PBX lines of business responsibility includes a collective responsibility for “PBX OT”, representing the Old Technology (legacy) PBX systems, as well as responsibilities for “PBX NT”, the New Technology PBX systems. The PBX OT intentional collective further includes responsibilities for the call control subsystem, while the PBX NT intentional collective includes responsibilities for the Virtual Peripheral component. Finally, figure 5-7 also includes the WML team collective.

The “new” link between Upper Management and the WML team responsibilities indicates that in response to a choice (as indicated by a design task included in Upper Management) Upper Management newly created the WML team responsibility. Other additions in figure 5-7 are the numeric annotations in some intentional collectives and roles. Numeric annotations can be added to intentional actors defined
to take design responsibility over the design of architectural elements. The numeric “1” annotation indicates that only one design time instance of the architectural element exists in the system, while a “+” indicates that more than one design time instances of the artifact can exists. For example, annotating Virtual Peripheral with “1” indicates that only one virtual peripheral component exists. The “+” label (shown in figure 5-11 in the User State processes intentional collective) indicates that the design responsibility represented by an intentional actor includes all “design time” instances of that artifact.

Figure 5-7: Establishing organizational responsibilities

Representing design responsibilities of capabilities included in a subsystem: Figure 5-7 further shows that the I/O Handler is an intentional collective that is part of the Call Control intentional collective. The I/O Handler intentional has three intentional roles assigned: Command Interpreter,
**State Manager** and **Signal redirector**. Each of these roles represents an organizational responsibility to design the corresponding capabilities. By representing organizational responsibilities as intentional roles and making them part of the I/O Handler intentional collective, it is indicated that at this stage of modeling, each of these responsibilities could be allocated to a different physical designer, without, however, restricting that two or all three intentional roles could be assigned to the same designer or collective of designers.

The model in figure 5-7 illustrates an organizational situation in terms of responsibilities usually carried out by more than one person (i.e. distributed responsibilities) that already suggests a non-trivial architectural decision-making environment for the WML team. The models in figure 5-8 to 5-11 add further details to include design responsibilities related to the architectural choices the WML team confronts.

**Figure 5-8: Creating new organizational responsibilities for stateless shared controller**

**Designing a stateless shared controller**: Figure 5-8 shows new organizational design responsibilities that the WML team establishes when selecting the **stateless shared controller based** design approach (to reduce clutter, the figure omits the higher-level organizational responsibilities). The WML team expects the collective design responsibility associated with the Virtual Peripheral to include design responsibilities for designing a WML browser, and a stateless WML controller. These are indicated by the design task dependencies, **include WML Browser in VP**, and **include Stateless WML controller**
in VP from the WML team’s “stateless shared controller based” design task to the Virtual Peripheral intentional collective.

The “Is-part-of” links between Integrated WML Browser, Stateless WML controller and Virtual Peripheral, do not indicate that the WML browser and the stateless WML controller components are necessarily part of the virtual peripheral component/subsystem. Rather, the link indicates decomposition of organizational responsibility, which could indicate a composite relationship among design artifacts also. To specify the composite relationship a separate design task dependency is included from the WML team intentional collective to the Virtual Peripheral collective, to indicate that the Integrated WML browser and the stateless WML controller components are to be made part of the virtual peripheral.

Figure 5-9: Creating new organizational responsibilities for stateless shared controller (continued)

Note, that if the WML team has full control over the decision to include these components into the virtual peripheral, then no intentional dependencies are necessary, only an intentional task within the WML browser intentional collective that indicates the inclusion.

Figure 5-8 includes additional responsibilities that the WML team concurrently established, including the collective design responsibilities WML signal redirector, WML state manager and WML
command Interpreter. The WML team further indicates that the WML signal redirector design responsibility is assigned to the Stateless WML controller design responsibility, and that WML State manager and the WML Command Interpreter are assigned to the State Manager and the Command Interpreter intentional collective, respectively. Since in figure 5-8 the State Manager and the Command Interpreter design responsibilities are assigned additional responsibilities, their symbol was changed from an intentional role, indicating one assignable organizational responsibility, to an intentional collective, indicating that this responsibility could be assigned in part to more than one designer persons or teams of designers.

Figure 5-10: Creating new organizational responsibilities for stateful shared controller

Designing a stateless shared controller, and intentional dependencies: Figure 5-9 adds to figure 5-8 to include intentional dependencies that the WML team defines between existing and new design responsibilities. For example, the integrated WML browser is dependent on the WML signal redirector to received redirected WML browsing signals. The WML team defines dependencies for the WML signal redirector and the WML Command Interpreter design responsibilities, indicating what each expects of the other so that their functionalities can be designed. Primary guiding forces during the
design of these responsibilities are non-functional requirements such as performance and reusability. For example, the **WML team** directs a performance NFR towards the **Call Control** responsibility. The performance NFR is propagated down to the **WML Command Interpreter** to guide its design. The **WML Command Interpreter** in turn propagates the performance NFR to the **WML signal redirector**, indicating that its performance property is (at least) partially dependent on the performance of the processing design decided on by person or persons responsible for the **WML signal redirector**.

![Figure 5-11: Creating new organizational responsibilities for master controlled WML infrastructure](image)

**Designing a stateful shared controller:** The diagram in figure 5-10 illustrates the organizational responsibilities created by the WML team when selecting a stateful WML controller approach. Instead of the **Stateless WML controller** in figure 5-9, a **Stateful WML controller** intentional collective is created. Furthermore, the responsibilities related to state are now made part of the **Stateful WML controller**. For example, the intentional roles, **WML Command Interpreter** and **WML State**
Manager, are now part of the Stateful WML controller responsibility. For this design approach, call control is not involved anymore, although a new intentional responsibility is created from the Command Interpreter, included in Call Control, to the WML signal redirector intentional role, included in the Stateful WML controller, to ensure that non-WML signal data continues to be redirected to the command interpreter.

**Designing a master controlled WML infrastructure:** Figure 5-11 illustrates the organizational design responsibilities that the WML team establishes when selecting a master controlled WML infrastructure. This design approach requires adding functionality to existing components. Figure 5-11 illustrates these extensions are also shown by exchanging intentional roles for intentional collectives and adding intentional roles as components. For example, the existing intentional role, Command Interpreter is now represented using an intentional collective and the new intentional role WML Command Interpreter is added and made part of the Command Interpreter intentional collective. In a similar manner, responsibilities for extending State Management and Signal redirector functionality are included.

![Figure 5-12: distributed design reasoning](image)

*Intentional actors creating intentional actors:* Figure 5-12 illustrates an example in which the WML
development team decided to form a new design team to design a new controller, which could either be stateful or stateless. This example is hypothetical since the formation of such a new design responsibility was not observed during the case study. Some of the design reasoning included in the goal graph in figure 5-6 is taken out and included in the new control actor. The model in figure 5-12 illustrates the WML team allocating the design of the new controller as a distinct organizational design responsibility and delegating to this new responsibility design goals that the designers of the new controller are expected to address.

5.3.4. From Generalized to Concrete Organizational Architectures

During the architectural discussions, the WML team referred to the existing call control architecture in an abstract manner, identifying functionality that needed to be redistributed. Figure 5-13 illustrates a generalized organizational architecture that abstracts away from the underlying concrete architectural alternatives discussed in the previous section.

In essence, when approaching the design of WML browsing capabilities the WML team considered the architectural design responsibilities as analogous to designing a shared device architecture. These included the design of the shared device, the applications that run on and share the device, and resource sharing functionality that enables running and switching between device applications. Furthermore, in their discussions only certain aspects of device sharing were considered, such as those that relate to the user actively switching between applications.

In figure 5-13 each modeling element includes the prefix “<g>” (introduced in Chapter 3, section
3.2.5) to indicate that these are generalized design responsibilities that are the basis for identifying and/or redistributing organizational responsibilities during architectural design and evolution. In effect, generalized responsibilities are organizational reference architectures.

![Diagram](image)

**Figure 5-14: Instantiating generic organizational architecture**

Figure 5-14 illustrates how organizational architecture for a stateless shared controller design approach includes intentional roles that are instantiations of the corresponding generic intentional role. More specifically, figure 5-14 illustrates (shown by the light brown arrow, not part of the modeling...
notation) how the generic organizational responsibility for the Command Interpreter intentional role (upper rectangle) is split into two intentional roles (lower rectangle): WML command interpreter and the Command Interpreter. Each intentional role is then assigned to a different intentional collective, indicating that, in principle, each could be assigned to different physical designers within a different organizational unit and/or hierarchy.

Figure 5-14 further illustrates that the Integrated WML Browser is an instantiation of the generic Device Application intentional collective. Once such instantiations are defined, intentional dependencies between generic intentional actors can also be allocated. For example, the intentional dependency, Redirect device data to active application, between the generic intentional collective, Device application, and the generic intentional role, Command Interpreter, is now included between the concrete intentional collective, Integrated WML Browser and the concrete intentional role, WML signal redirect. Note that splitting intentional collectives or roles (i.e., creating more than one instance) also suggests duplicating intentional dependencies accordingly.

By explicitly defining and reusing generic organizational architectures as reference organizational architectures, designers are guided in identifying the design elements and their associated organizational responsibilities that need to be included when designing and/or evolving concrete architectures.

5.4. Responding to Research Questions

Q1.1. What role do business goals play during the architectural decision-making?

During this case study, it was observed that business goals played several roles during architectural decision-making. First, business goals such as increase sales and increase customer base served as the motivation for developing internet browsing capabilities on telephone sets, and the establishing the WML project to design and develop these capabilities. Upper management saw an opportunity to increase the customer base by offering WML browsing on the company’s telephone sets and decided to direct resources towards establishing such capabilities. Business goals, such as having competitive products in the product portfolio, while reducing development time and cost, reduced development risk, also drove the strategic architectural effort within the organization as a whole. The need for open systems approach and architectural evolution was justified by upper management and architecture strategy by referring to such higher-level business goals.

Business goals also played a role during architectural decision-making performed by the WML team.
Non-technical members presented business goals as part of the criteria and justifications for choosing amongst architectural alternatives. In effect, these non-technical stakeholders represented upper management’s interests, as part of their organizational responsibility. For example, the Call Control architect’s concern for reduced maintenance cost as well as to reduce product risks are business goals that prompted the architect to advocate for maintaining architectural integrity, and hence, justified to the Call Control architect the design approach to including the WML browser in the Call Control subsystem.

Indeed, when discussing architectural alternatives WML team members did not clearly distinguish between business goals and derived design goals, and mentioned both types of goals while advocating for one design approach and discussing relevant tradeoffs. This observation was mirrored in IAL in softgoals can represent both business and design goals.

**Q1.2. What modeling constructs are appropriate for making business goals and the role they play during architectural decision-making explicit?**

It was appropriate during this case study to use softgoals to represent business goals. Softgoals helped indicate that business goals, such as to develop competitive products, to reduce development time and cost and to reduce development risk, are tentative and inherently in conflict with others. During the many architectural decision situations organizational participants must repeatedly renegotiate business goals and trade them off with each other.

In this case study it was also observed that the reasoning structures offered by the strategic rationale model helped indicate to decision-makers how architectural design options put forward contributed (directly or indirectly via intermediate softgoals) to the achievement of business goals, or the need to trade them off for others.

Intentional dependencies that link business goals amongst intentional actors further supported representing the fact that goal achievement must be negotiated amongst organizational participants, and that their achievement is not ensured, by merely delegating their achievements to organizational participants in the organization.

During this case study it was also observed that upper management appealed to business goals to justify the creation of the new WML development project. The IAL notation supported representing the creation of such intentional actors in response to the selection of design tasks that contributed to relevant business goals. For example, in figure 5-5 the selection of the design task internet browsing on
telephone sets that contributes positively to the business goals increase customer base, was the reason for creating the new intentional actors WML team.

Q2.1. What role do qualitative NFRs play during architectural decision-making to meet business goals?

During this case study it was observed that the main role NFRs played during architectural decision-making was to serve as criteria to discuss and evaluate architectural alternatives, and point to tradeoffs. For example, NFRs such as minimize processing error, maintain architectural integrity, reuse of commercial software code (see figure 5-6), were mentioned to compare the merit of the different architectural alternatives.

During discussions participants also referred to NFRs to explain other NFRs. For example, one participant explained that to reduce complexity of software code, a relevant NFR for the WML team, means to reduce the need to change legacy code, and to reduce the need of developing custom code. NFRs thus helped clarify the meaning of other NFRs. Another example was the explanation of what the architectural evolution goal entailed in the context of the WML project. The Architecture strategy team member explained that to support architecture evolution implies a WML infrastructure design that supports the evolution of components related to the controller(s) and the evolution of components included in the browser application.

Related to the previous point was the role NFRs played to abstract away details during design discussions. For example, the NFR architectural evolution was often mentioned during discussions, without elaborating on its more detailed meaning. Under some circumstances this a good way to making a point, such as to explain that a master controlled WML based infrastructure does not support the evolution goal without the need to delve into additional details. Higher-level NFRs thus compactly referred to more elaborate design implications.

NFRs also helped to explain how to address business goals or other NFRs. For example, it was pointed out that to reduce time to market it would be necessary to try to reuse commercial software code (i.e., off the shelf components), and try to reuse the existing software system infrastructure as much as possible, and to maintain the architectural integrity. In this sense NFRs served as bridges between business goals and architectural decisions.

NFRs also helped identify what solution elements of design approaches contribute to the approaches non-functional properties. For example, the question: “what part of the master controlled WML based
infrastructure design approach helps to maintain architectural integrity”, leads to identifying that it is the design of the Browser proxy as user state process that establishes this non-functional property. Similarly, having the Browser run externally as a Windows NT process (Browser on NT) is what establishes the reuse of commercial software code property. NFRs thus help in identifying the key solution elements of each alternative design approach, their respective non-functional properties, and hence explain why each solution approach is different in terms of non-functional properties from the others.

During this case study it was also observed that organizational participants did not make a clear-cut distinction between business goals and non-functional requirements, and treated both similarly during architectural decision-making. For example, during architectural decision-making the Call Control Architect was concerned with reducing the risk of new products, reduce time to market, both business goals, as well as maintaining architectural integrity, a non-functional requirement.

In general, it was observed that technical stakeholders, such as the different architects, and the industrial designer, tended to emphasize trade-offs related to NFRs during architectural decision-making, while non-technical stakeholders, such as Marketing and New Business Strategy, tended to emphasize business goals.

Q2.2. What modeling constructs are appropriate for making qualitative NFRs and the role they play during the architectural decision-making to meet business goals explicit?

In this case study it was observed that the strategic rationale modeling approach was appropriate for capturing many of the roles NFR played during architectural decision-making. Softgoals supported capturing NFRs, thereby indicating that their meaning is often first-cut and tentative, and in need of elaboration. For example, that architectural evolution is likely in need of further elaboration when exploring the architectural alternatives. Softgoals also supported indicating the qualitative nature of the architectural design discussions. The WML team participants did not quantify the mentioned NFRs but used adjectives such as “better”, “worse”, “supports”, “harms” and the like to describe in qualitative terms how architectural approaches relate to, or are justified by, NFRs.

Contribution and correlation links, together with contribution types (positive, negative, sufficient, etc.), helped indicate how NFR mentioned during discussions relate to each other (and to goals). For example, the reuse of commercial software code NFR contributes positively (but not sufficiently on its
own), to the **reduce time to market of software development** goal. This helped explain how the WML team considered addressing, in part, the reduced time to market goal. Contribution and correlation links also supported clarifying the meaning of NFRs, as well as indicating how the alternative design approaches contributed to stated NFRs.

NFRs in conjunction with design task decomposition also helps identifying the elements of design approaches that contribute to NFRs, which in turn helps to clarify how design solutions differ. For example, by decomposing the master controlled WML based infrastructure into several sub design tasks, **Browser proxy as user state process, browse state in user state manager** and **browser on NT**, it is possible to indicate how the components of the solution approach contributes positively or negatively to the different relevant NFRs (see figure 5-6).

**Q.3.1. What role does the organizational setting play during architectural decision-making to meet business goals?**

While WML team members prominently mentioned business goals and NFRs (albeit often only when comparing alternative solution approaches), they said little about the organizational setting in which they worked, and how it influenced architectural decision-making to achieve business goals. The call control architect mentioned that upper management had specifically established the WML project as a tactical measure, since they currently perceived an opportunity to increase the customer base by offering internet browsing on phones. The Architectural strategy architect explained that upper management was much interested in architectural evolution as a means to ensure competitiveness.

Other details related to the organizational setting could be gleaned from observing the larger organizational context in which the WML team made its architectural decisions. One observation for example was that all members of the WML team held responsibilities in other design and development efforts at Mitel, some of which had overlapping design concerns with the WML project. The IP phone architect for instance leads the architectural design of a new IP phone product line designed to work with a novel IP PBX system. The architect also designed the IP phones to work, via the Virtual Peripheral, with the existing centralized PBX (it was not mentioned during the case study whose design responsibility the Virtual Peripheral was, whether that of the IP phone architect, the Call control architect, or some third person). In relation to the WML project, the IP phone architects thus articulated design objectives that supported an evolution path for IP phones system components from the legacy PBX to the new IP PBX system, thereby aiming to increase the synergy between development work done
for the WML project, and future development needed for the novel IP PBX system.

Another observation was that the WML team was a coherent decision-making unit tasked by upper management to design internet browsing infrastructure and capabilities for telephone sets. This included, if necessary, the making of tradeoffs, to advance the WML projects’ specific goals, such as to ensure that the opportunity to increase the customer base was not missed. All while at the same time not making unreasonable current and future demands on different parts of the development organization.

It was also observed that the goals articulated by team members, when further probed why they were pursued, revealed broader organizational and business purposes and contexts. For example, the reason why the Architectural Strategy architect pursued architectural evolution is to making the current proprietary switching system more open (i.e., an open systems approach). This in turn allows increased use of off-the-shelf components, which in turn helps increase reuse, reduce time to market, and reduces the risk during new systems development efforts.

Probing the reasons also revealed higher-level stakeholders in the organization whose mandate it was to ensure the meeting of those goals. For example, it is upper management that establishes project and teams to achieving those organizations goals.

Yet, another observation was the effect on stakeholder and designers at different levels in the organization when different development projects needed to deal with conflicting demands, in particular demands that required trading off short-term costs for longer termed gains. For example, the WML team was essentially required to choose between a centralized approach to introducing WML browsing, that offered immediate benefits, and various degrees of decentralization, that carried an immediate cost (more complex integration issues), but carried future benefits (easier evolution). Such choices have repercussions that go beyond the boundaries of the WML team, and affect the higher-level goals of the organization in different ways. Considering that other development teams are confronted with similar kinds of tradeoffs and choices, the manner how higher level goals in the organization are achieved depends on complex negotiation processes between upper management and various organizational participant and teams.

Although upper management can articulate design objectives for development projects and teams, whether and to what extent these would be meet is uncertain and contingent on many factors not directly controllable by upper management or the development teams themselves.

A case in point is the following observation:

With the somewhat ad-hoc creation of the WML team, to pursue a tactical opportunity, and with
WML team members having parallel organizational responsibilities in other development projects, it was not clear how the team would resolve the different tradeoffs that each architectural approach engendered. Given the ad-hoc nature of the project, no obvious team leader existed who could consolidate and make a final choice. On the contrary, WML team members were not only equally consulted for their opinions, but appeared to have an equal say in the decision-making also. Nevertheless, at the conclusion of the case study, after models were presented to the team, several team members spontaneously huddled together, reviewed the team’s discussions during the presentation, and arrived at a decision. It would be difficult for upper management to a-priori predict which of the tradeoffs the WML team would eventually make, and consequently, which of its (higher level) objectives would be addressed, and which traded-off.

Finally, another observation related to the organizational relationship between the decision done by the WML team and developers further downstream in the development process. It was clear to the WML team members that once a decision was made, as to where the WML browser would be located, the resulting architecture could be given to developers, who would implement the WML browsing capabilities, accordingly. The WML team thus understood which decisions it was required to make and which decisions it could leave to developers.

Conversely, the WML team understood when decision-making was not completed, and a hand-off to developers was not yet desirable. For example, once the WML team considered to include the WML browser in the Virtual Peripheral (indicated by the shared controller based WML infrastructure design task in figure 5-6), two additional choices presented themselves—a stateless and a stateful shared controller design approach. The WML team identified that these choices need to be explored by the WML team and included these in their discussion and decision-making, instead of leaving the choice to developers further downstream.

The WML team thus exhibited a sense for the boundaries of architectural decision-making that was required by them to be made, and decisions that could be left to others in the development organization.

**Q.2.3. What modeling constructs are appropriate for making the organizational setting and the role the organizational setting plays during the architectural decision-making to meet business goals explicit?**

**Constructs representing WML team members.** The WML team members were represented using
intentional roles thus indicating that each team member played a distinct role in the WML team. Each team member articulated individual design objectives for the WML project which came about in light of his/her own role in this and other development projects in the organization. These design objectives were represented as softgoals within each intentional role. The role names, such as **IP phone architect**, **New Business Strategy**, etc., indicate that the organizational responsibilities assigned to the WML team exist independently of the creation of the team. For example, the role **IP phone architect** is part of the IP phone system project (not shown in the figures), which was established before the WML project was conceived. The intentional roles assigned to the team thus represent organizational responsibilities that already existed in the organization and that upper management considered relevant to be included in the WML development effort. Upper management could, of course, have created additional roles to deal with specific concerns related to the WML project, however, this was not observed during this case study.

The intentional collective construct was used to representing the WML team as a whole, indicating that the team is a collective of organizational units that includes team members with differently assigned responsibilities.

While intentional roles helped to individually indicate the team members’ goals, one all-encompassing strategic rationale graph consolidated the individual design arguments elicited during interviews. The reason for consolidating individual design reasoning was because a key purpose of this case study was to present a summary of the team members’ design arguments. This is in contrast to a modeling approach taken in the industrial case study presented in Chapter 6, where distinct goals and design reasoning of designers discussing a shared design issue at hand were represented (Daniel Gross & Yu, 2010b). In that case, an intentional viewpoint construct was used to encapsulate how each designer differently rationalized the same design issue at hand. Intentional viewpoints were used since the designers had overlapping responsibilities over a design artifact. However, this approach was not appropriate for the case study discussed in this chapter.

**Constructs representing upper management.** In this study, upper management was represented using the intentional position construct since upper management appeared as a coherent decision-maker whose goals and decision-making was not fragmented amongst different organizational decision-makers. However, when it was observed that upper management dealt with different decision responsibilities at different points in time, these focal points of interests were captured individually as intentional roles that are part of the intentional position. For example, at one point in time, upper management focused on decision-making related to the development of next generation systems, and
at another point, focused on portalization of telephone sets in relation to the creation of the WML project management. These focal points were captured as the **Next Generation Systems** and the **Portalization of telephones** intentional roles in figure 5-5. Since upper management is represented using an intentional position, the model includes the assumption that both roles were occupied by the same upper management person or team of persons in the organization.

The Mitel organization as a whole was represented as an intentional collective, and the expectations to addressing business goals that Mitel has on upper management is indicated by directing business goals via intentional dependency links to upper management. Upper management roles then address some of these business goals through decision-making and/or by delegating goals further downstream to the other organizational participants, including the WML team.

Intentional dependencies from upper management’s intentional roles to the intentional roles representing WML team members’ responsibilities in the WML project, help explain the origins of design goals each WML team member articulated. For example, goals such as reduce maintenance costs, reduce time to market and product quality, associated with upper management, help explain why the call control architect is interested in maintaining architectural integrity, since each of these goals is helped when architectural integrity is maintained. Representing team members using intentional roles, and linking these via intentional dependencies to depender actors, thus indicates part of their responsibility to interpreted “incoming” design objectives, and articulated design objectives for the WML team effort accordingly.

** Constructs indicating opportunities and vulnerabilities:** The meaning of intentional dependency goes beyond simply indicating a trace between the goals of a depender actor and those of a dependee actor; even more so, intentional dependencies indicate the organizational opportunities that actors in organizations seek out and the vulnerabilities that they are exposed to (E. Yu, 1994). For example, the intentional dependency to reduce maintenance costs from upper management to the call control architect, on one hand, indicates that upper management is taking advantage of the knowledge and expertise of the call control architect to design the call control subsystem (and hence reduces maintenance costs—a sought-out opportunity). However, at the same time, the call control architect may not be able to stop certain design choices that the WML team within the call control subsystem make and that may adversely affect the reduce maintenance cost goal. Thus, the call control architect is exposing upper management’s goal, reduce maintenance costs, to some vulnerability. More specifically: the call control architect may not be able to avoid the design approach which places the WML browser within the Virtual Peripheral, thus trading reduced maintenance costs for architectural evolution goals.
It is noteworthy that by using intentional dependencies between the roles of upper management and those of the WML team members, the various opportunities and vulnerabilities that upper management may be exposed to depending on the different choice the organizational participants could make are identified. Given the goals they want to prioritize, the model in figure 5-5 thus could guide upper management in identifying the organizational relationships they need to focus on in order to increase the likelihood that those tradeoffs are made that suit the organization’s higher-level goals.

**Constructs indicating decision boundaries:** The models in figure 5-6 and 5-12 illustrate two approaches in delegating decision-making across intentional actors: the strategic rationale model in figure 5-6 shows the result of architectural analysis and identified choices that the WML team identified; figure 5-12 shows the analysis choices relating to the design of a new controller component, which was delegated to a new team to design it. The boundaries of intentional actors thus support indicating alternative ways that design analysis and decision-making can be delegated between intentional actors. The ability to link architectural choices to softgoals in strategic rationale models helps indicate which goals are already sufficiently addressed (or not) within a strategic rationale graph. If, for example, the WML team would have decided (by following the contribution links from design task to softgoals) that using the shared controller approach would sufficiently address all priority design goals of the team participants, then the team could have easily delegated the design for the new controller to the new, different team. The boundary concept of intentional actors thus supports representing and reasoning about decision-delegation in an organization setting.

**Q1.4. What role do quantitative NFRs play during architectural decision-making to meet business goals? Q2.4. What modeling constructs are appropriate for making quantitative NFRs and the role they play during the architectural decision-making to meet business goals explicit?**

This literature case study did not deal with quantitative NFRs.

5.5. **Modeling Discussion**

**Defining an organization as collective:** Why is the Mitel organization included in the model? Mitel is essentially defined by its constituent agents, roles, positions and collectives. Specifying Mitel essentially means to specify the constituting intentional actors, in particular their business and system goals. However, for modeling purposes representing Mitel as a distinct intentional actor has its benefits.
At one level of discourse, it is Mitel as a whole that depends on third parties, and vice-versa, third parties depend on Mitel. The Mitel intentional actor can thus serve as a focal point for Mitel’s intentional dependencies with third parties in the market place, such as customers, suppliers and the like. A model that includes Mitel as an intentional actor can thus represent the strategic relationships that guide Mitel in its dealing with the outside world. Representing an organizational as a whole and its relationship to clients is illustrated in Chapter 6.

The Mitel actor can also serve as a focal point for establishing who in the organization has been made responsible for “incoming” design goals. This usage was illustrated in this chapter, with intentional dependencies originating from the Mitel actor and directed towards the upper management position.

It is important to emphasize that such a modeling approach is an abstraction of much more complex organizational delegation and decision processes. However, focusing on the intentional analysis facilitates focusing on specific areas of interest, such as the WML project and team decision-making, rather than on the high-level managerial processes that established upper management positions and roles.

**Design by Abstraction and Synthesis:** This chapter illustrates a design by abstraction and synthesis approach. During the abstraction steps, a generalized organizational architecture model is created. One or more generalized organizational architectural models can then be synthesized to design a new concrete organizational architecture solution. Often an abstraction step occurs during architectural evolution efforts, when some aspects of an existing architecture are emphasized while others are ignored. For example, during the architectural evolution of the call control system, two relevant design themes were identified that guided the abstraction step: one for a sharing device, and the other for integrating new capabilities in a legacy system. Each design theme can be represented using generalized design responsibilities. During the architectural evolution, both themes were synthesized and included into already existing design responsibilities of the call control system.

5.6. **Conclusion**

Using a real-life project in an organizational setting, this chapter presented the use of actor and goal modeling to represent and communicate design reasoning amongst stakeholders and designers during the architectural design and evolution effort. The models presented in this chapter are somewhat expanded version of the models that were presented to the WML development team and other management stakeholders during an hour-long presentation. During the presentation, the WML team made use of the diagrams to review the main discussion points. After the presentation, the team
proceeded to choose one alternative over all the others. Team members felt that the diagrams adequately captured the design discussions during the previous weeks, and helped to consolidate the discussions and to arrive at a decision.

It is interesting to note that the WML team had not been exposed to actor and goal modeling before, and the approach of representing architectural design reasoning using a goal and actor-oriented notation was new to them; the practitioner’s usual approach to architectural design involved generating different solution approaches in an ad-hoc manner, and informally discussing them amongst each other. Modeling and analyzing alternative design approaches by use of a strategic rationale model that represented alternatives design approaches and their respective contributions to design goals significantly departed from their usual approach to expressing and dealing with design alternatives. In addition, the agent-oriented approach to illustrate goal propagation in the organization was even further removed from the way these practitioners thought about architectural decision-making.

The use the practitioners made of these models to consolidate and drive decision-making was thus important feedback. However, while the models presented were seen as helpful, more research is needed to determine if such an approach could become part of the practitioner’s standard repertoire during architectural decision-making, and how the modeling and analysis should be performed – whether by the architects themselves of dedicated facilitators who capture and consolidate ongoing design discussion and present these to the decision-makers.
6. IAL Applied to the “Phoenix” Industrial Case Study

6.1. Introduction

This chapter reports on a case study in the information technology department of The Phoenix insurance company. At the time of the case study the CTO of the organization and an enterprise-wide SOA architect were advocating for, and implementing, an enterprise-wide effort to evolve the current enterprise application landscape to a service-oriented enterprise architecture. The CTO and SOA architects had to explain to upper management the reasons for an SOA evolution strategy while also deal with the day-to-day implementation of SOA design approaches in the organization.

The study modeled and analyzed an architectural design discussion, reported by an SOA architect, which took place between the SOA architect and a component designer for an Enterprise Application. The SOA architect argued for one design approach, while the component designer for a different approach.

Prior to this study the author of this thesis was given by the CTO two hours to present the modeling approach, explored during this study, to convince him of the utility of the approach to his IT development organization. The CTO explained that this initial presentation was an essential first step to gain further access to his organization to study architectural decision-making.

During this first presentation, during which the CTO also included the enterprise-wide SOA architect, the modeling notation was presented and the motivation for an intentional actor and goal-oriented modeling notation was explained. Both, the CTO and SOA architect, quickly identified some value such a modeling notation could offer to them. The CTO for example pointed to the ability of the notation to link architectural decision-making to higher-level organizational goals, a feature the CTO saw as an important contribution to better dealing with Governance processes in the organization. The enterprise-wide SOA architect pointed to the ability of the notation to make explicit alternative design options and their respective tradeoffs, a feature the SOA architect identified as particularly useful in his work.

At the conclusion of the presentation, the CTO agreed to a case study wherein this author would be studying a design decision, related to the architectural design of a messaging feature, that he and the SOA architect were currently discussing with a component designer. Both the CTO and SOA architect felt

---

20 This chapter is based on (Daniel Gross & Yu, 2010b)
that the presented modeling approach could offer some benefit in communicating the key design arguments each side was putting forward during this discussion. The CTO and SOA architect suggested that I further interview the SOA architect to elicit the key arguments of this design discussion.

The thesis author subsequently interviewed the SOA architect for about an hour eliciting the key argumentation points for and against two alternative design approaches offered by the SOA architect and component designer. The interview was recorded, and the discussion summarized in writing. The summary was then presented to the SOA architect to verify that all relevant decision points were elicited. The summary was then modeled and analyzed and the results presented to the CTO and SOA architect, who provided feedback on the models produced.

Based on the analysis of this design discussion reported by the SOA architect this chapter presents main modeling features including:

- Representing and analyzing the design reasoning and decision-making of designers and stakeholders from different contrasting viewpoints, such as from a component designer’s point of view and the enterprise architect’s point of view;
- Identifying higher-level management stakeholders and decision-makers whose goals, priorities, and choices influence the goal- and design-reasoning and decision-making of designers and stakeholders at a lower hierarchical level in the organization;
- Systematically analyzing goals and design approaches of lower-level stakeholders and designers in light of goals, prioritizations, and tradeoff making of higher-level stakeholders and designers in an organization.

This chapter expands on the published paper by including a detailed description of the case study, the models produced during the study, as well as discussing the findings in relation to the thesis research questions presented in Chapter 2.

6.2. Description of Case Study

During the study the SOA architect and the CTO of the company were interviewed, during which they reported on an ongoing design debate between the SOA architect and a designer of an enterprise application component: the SOA architect wanted component designers to use asynchronous messaging to access service providers whereas the consumer-component designer wanted to use synchronous messaging.

This design discussion is an example of “design time” interaction that typically occurs between SOA architects and those who must comply with their design guidelines, and is noteworthy because a debate
The design discussion occurred in the following design context: A consumer component handles purchase requests by potential new insurance customers for new insurance policies. Once all policy purchase data is entered the consumer component sends the new policy request to a provider system to create the new insurance policy. An Enterprise Service Bus (ESB) subsystem forwards messages from the consumer component to the providers and vice versa (Josuttis, 2007). However, as stated above, to implement the transmission of policy data from the consumer to the provider, the SOA architect wanted the consumer component designer to use asynchronous messaging, while the consumer component designer preferred to use synchronous messaging. Both messaging approaches are, in principle, offered by the ESB.

**Advantages of synchronous messaging.** The consumer component designer preferred synchronous messaging for the following reasons.

- **a) Simplification of the consumer component:** Synchronous messaging simplifies consumer design and code. It simplifies sending new policy requests to a specific provider, and simplifies the handling of the responses received from the provider. “Simplification” is an important design goal since it helps make code more understandable, which contributes to the reduction of the consumer’s maintenance costs. A simpler design is easier and faster to code, reducing the potential for bugs and reducing development costs—less time and money is spent on coding synchronous messaging than on asynchronous messaging.

- **b) Speed:** Synchronous messaging returns an immediate response. The ESB can immediately fulfill a synchronous request, avoiding the delays that can occur during asynchronous messaging. This improves response time for the consumer component and contributes to the customer’s perception of quality of service.

- **c) Improved design accountability:** Having a response returned immediately improves the control that the designer of the consumer component has over the overall policy submission process. Synchronous messaging allows the consumer component designer to design directly for various contingencies, such as if the process never ends or takes too long; asynchronous messaging does not offer such direct control since responses are not immediate and are also uncertain [therefore also called “fire and forget” messaging (Josuttis, 2007)]. Having control over the process is important to allow accountability for the design quality to higher-level Product Management.
Advantages of asynchronous messaging. The SOA architect preferred asynchronous messaging for the following reasons.

a) Better load balancing: Asynchronous messaging is a more natural approach to support load balancing at the backend: a service request is queued and can be taken by any physical provider available (such as when several physical provider servers establish one logical service provider). Load balancing contributes to the system’s scalability.

b) Improved resource efficiency: Asynchronous messaging requires fewer SOA infrastructure resources (such as session information), and ESB resources can be allocated for shorter periods. This
increases the number of consumers that can concurrently send messages over the ESB infrastructure per SOA infrastructure dollar. Reducing the time that ESB resources must be allocated also significantly contributes to avoiding infrastructure bottlenecks, which is key to achieving scalability.

**c) Improved extensibility:** Synchronous messaging requires identifying a specific message destination, while asynchronous messaging does not. Asynchronous messaging therefore supports forwarding policy data to other relevant providers, without needing the consumer to know a-priori about additional destinations. This supports adding and removing providers dynamically during runtime. Being able to easily add provider destinations also helps when requirements require a consumer component to submit additional policy data, which new providers extract and process. By using asynchronous messaging new data can be submitted by the consumer component, and new providers added could process the new data, with no need to introduce design changes to the consumer component.

**d) Simpler “exception handling”:** When more than one provider is involved in the processing of a new policy, synchronous messaging requires the consumer component to wait until all the providers successfully complete their processing. A situation may occur where one or more providers return error codes while others complete processing successfully. How a consumer component should be designed to deal with such a case is not obvious, and complicates the design of the consumer component. Since in asynchronous messaging such issues are delegated to the ESB the consumer component’s design becomes much simpler.

**e) Simplification of multiple-provider feedback processing:** By simplifying support for multiple feedback messages returned by one or more providers, and simplifying routing feedback messages to several interested consumers, asynchronous messaging supports the implementation of multi-channel consumer access. In the event, for example, that an insurance client submits a new policy request via the internet, and then want SMS notifications about the status of the request’s progress, asynchronous messaging can directly support such decoupling whereas synchronous messaging will require more design work.

### 6.3. Representing and Analyzing Design Reasoning from Contrasting Points of View

**Consumer designer’s point of view:** Figure 6-1 illustrates the strategic rationale model included in the consumer’s intentional viewpoint, which captures the design reasoning of the consumer designer. At the bottom of the diagram is the overall design goal **Publish data to provider.** Two alternative design tasks
were identified that achieve this design goal: Publish data synchronously and Publish data asynchronously. Given the preference of the consumer designer for a synchronous messaging design approach, this design task is further analyzed. Two sub-design tasks are identified that are relevant to the design argument of the consumer component designer: that the synchronous design requires Wait and handle response, and Publish to a specific Provider. Both of these design approaches (captured by design tasks) are considered appealing to the consumer designer, as explained below.

![Figure 6-2: intentional viewpoint of the SOA architect](image)

The softgoals that are linked to these design tasks — Accountability of Process, Quality of Service, Response time—indicate why the consumer designer considers these approaches useful: they offer better accountability for the designed software behavior, they perform better, and they simplify the design. By repeatedly asking “Why?”, the higher-level design goals that the consumer designer pursues, such as quality of service perception of the component user, and reduced development and maintenance costs, are revealed.
SOA architect’s intentional viewpoint: In contrast, figure 6-2 illustrates the design reasoning of the SOA architect. The design responsibility of the SOA architect is included as the intentional collective “SOA evolution”, which is assigned the Publish data mediated by ESB viewpoint. Starting from the same overall design goal Publish data to providers, the same two alternatives, Publish data synchronously and Publish data asynchronously are identified. Softgoals capture the SOA architect’s arguments, such as reduced resources needs per request, reduced time needed for resource to be allocated, as well as others. Also included are arguments that suggest precluding synchronous messaging, such as the negative effect on minimizing change of consumer components when requirements change. Repeatedly asking “Why?” reveals the higher-level design goals of the SOA
architect, which include scalability of the system, optimal use of system resources, and the need to reduce consumer maintenance.

**Placing viewpoints side-by-side:** The diagram in figure 6-3 illustrates the design reasoning of each participant placed side by side. The SOA architect can use this model to explain to the consumer component designer both sides of the design argument.

### 6.4. Identifying Higher-Level Stakeholders and Decision Makers

Besides showing the arguments of each designer, figure 6-3 also illustrates how the top-level design goals of the SOA architect and the consumer component designer could be derived from a higher-level production management stakeholder. During the interview a product management and a Business Strategy stakeholder were not specifically mentioned, however, the feedback obtained from the SOA architect and the CTO indicated that such higher-level responsibility can be identified in the organization, and including these in the diagram is thus appropriate.

The product management stakeholder requests the reduction of development and maintenance costs from the consumer component designer, and a design that meets future scalability, usability, and maintainability needs from the SOA architect.

Figure 6-4 includes a **Business Strategy** stakeholder and the **Insurance Company** as a whole to represent how architectural design goals are linked back to the needs of the Consumer component user – the customer of the insurance company. According to figure 6-4, the **Consumer User** depends on the **Insurance Company** for reduced premium payments, on one hand, as well as **quality of service** and **customer satisfaction** on the other. These goals are directed towards the insurance company as a whole.

According to figure 6-4, the insurance company decides that to minimize premium payments, it is best to work on minimizing administrative costs while keeping an eye on quality of service; the company then decides to direct these new requirements to business strategy in the organization, who is then tasked with identifying appropriate approaches to that goal. This is captured by the **Minimize Administrative cost** dependency link from the insurance agent intentional collective representing the insurance company to the **Business Strategy** role. **Business Strategy**, in turn, decides that to minimize administrative costs, the cost of software should be reduced (**Reduce Software Cost**), and decides to direct this new requirement to **Product management**. Finally, **Product management** decides that to reduce software costs, the cost of development and maintenance must be reduced, and directs these to the **Consumer component**.
The design goals involved in reducing maintenance costs that are directed from **Product management** to the **Consumer component** is a relevant design goal because different decision-making stakeholders in the organization have gradually, successively, and collectively identified that this is a good way to reduce the policy premium cost in the organization, and thus to attract new customers and
keep existing customers satisfied with their insurance provider. The intentional dependencies linked across organizational and technical intentional agents convey that, in the final analysis, the architectural decision whether to use synchronous or asynchronous messaging is linked to the business goal of acquiring and retaining paying insurance customers.

Depending on the organization, business goals and organizational structures may well be different, leading to different requirements and eventually different design tradeoffs. Josuttis, for example, reports on an enterprise organization where the SOA architect specifically demanded that component designers use synchronous messaging (Josuttis, 2007), mainly because of the increased complexity in debugging asynchronous processes in enterprise systems.

6.5. Analyzing Viewpoint Reasoning in Relation to Higher-level Goals, Prioritizations, and Tradeoff-making

Presenting the intentional viewpoints side-by-side helps consumer component designers and SOA architects discuss their respective design approaches. Systematically presenting the reasoning of each side is a first step in justifying why one design approach might take precedence over the other. However, to gather additional support designers may need to appeal to the design reasoning and goal prioritization of higher-level stakeholders in the development organization. When needed, the prioritization of higher-level stakeholders can then guide which design goals to trade off and which design approach to adopt.

Figure 6-5 illustrates this point and shows the internal goal graph of the Product Management agent. This goal graph captures how Product management interprets and refines their top-level goals: Reduce software cost and Quality of products. It can be seen that, according to production management, reducing software costs is addressed by reducing both consumer component costs and infrastructure costs, whereas addressing product quality is done by addressing service quality. Product management further indicates that the reduction of infrastructure cost is of priority—shown by a red exclamation mark beside the softgoal—while reducing the cost of the consumer component is not. Product management also indicates that another way to help reduce infrastructure cost is to make the infrastructure scalable. Scalability is therefore also given priority. On the other hand, while reducing consumer-component maintenance costs is considered a valid method of reducing software cost, it is not prioritized.
Figure 6-5: Higher-level prioritization

By interpreting and prioritizing design goals, product management indicates to lower level stakeholders and designers what tradeoffs they prefer, such as to trade off **Reduce maintenance cost of Consumer component** in favor of **Scalability of System**. By appealing to such higher level goals and prioritizations, lower level stakeholders and designers are able to resolve disagreements and support design choices that help achieve higher-level business and system goals.

**Feedback from study participants**: The case study identified that distributed reasoning and decision-making is part of architecture evolution processes at The Phoenix insurance. The CTO and the SOA architect both reported to struggling to convince decision-making stakeholders and designers in the organization for the need to adopt SOA design principles and approaches.

The CTO and SOA architect confirmed that intentional viewpoint modeling provides a useful communication tool to help document side-by-side the argumentation of different stakeholders and designers during design discussions. They both also anticipated that such models could also help reduce
the need for face-to-face discussions between the SOA architect and enterprise application designers, as an efficient means for documenting the design reasoning underlying design decision-making, and for providing to designers and stakeholders in the organization with quick reminders, why one design choice was favored over others.

The CTO and SOA architect explained that this was a key problem. Since they needed to repeatedly spend much face-to-face time to explain again and again why the proposed SOA design principles were important and the trade-offs necessary. They felt that this was much wasted time both for them and for others. The CTO and SOA architect therefore saw much value in the models produced, which help document rationales in a manner that was not possible by methods and tools they used so far. This was apparently despite the upfront effort to produce such models, since both were interested in having such models produced to help with the ongoing discussions.

Finally, the CTO and SOA architect also perceived as helpful the inclusion in the models of management level stakeholders, their goals and goal reasoning. This helped put the ongoing design discussion into the broader organization decision-making context, which was seen as a contribution to IT Governance – the need to justify architectural decision-making in light of organizational strategic goals and directions.

**Simplifying diagrams:** One item of feedback received during this case study was the need to keep agent and goal reasoning structures simple. This refers to the number of reasoning elements (agents, goals, and solution elements) to include in agent and goal models as well as the degree of formality the agent and goal-oriented notation should offer. Both the SOA architect and the company CTO suggested the importance of simplifying intentional agents and viewpoints to support representing and communicating key discussion points of/to different relevant stakeholders in the organization (such as intentional viewpoints for maintainers, security, deployment, and the like). The CTO felt that such simplifications would in fact be key for the successful adoption and use of the proposed technique in “the real world” in an enterprise organization.

However, a more detailed and higher degree of formalization offers benefit for computer assisted analyses such as to systematically compare and contrast reasoning captured in intentional viewpoints when taking into account the different scope and level of abstraction represented. We believe that each organization applying an agent and goal-oriented modeling approach needs to identify a balance between modeling details, formality, effort and the benefit it obtains.
6.6. Responding to Research Questions

Q1.1. What role do business goals play during the architectural decision-making?

Business goals played several roles during architectural decision-making. First, upper management articulated business goals and required that decision-making in the organization, including architectural decision-making be justified in terms of those goals. This ensured that decision-making in the organization was aligned with the goals of the business. In this case study it was therefore observed that the CTO and SOA architect were keenly aware of the organization’s business goals during architectural decision-making. Both the CTO and SOA architect felt compelled to justifying SOA decision-making to upper management.

Business goals also helped in explaining tradeoffs and choosing between architectural choices. More specifically, the prioritizing of business goals and in particular whether to emphasize current operational or future strategic goals had an effect on architectural choices. For example, choosing whether to adopt a synchronous or an asynchronous messaging approach essentially related to whether current operational goals such as cost effective and timely implementation should be given priority over future goals such as extensible, flexible and scalable but more costly solutions. In this case study future capabilities were preferred over reducing current development costs.

Business goals also justify, and explain the existence, of non-functional requirements. For example, in this case study it was found that the NFR to reduce software cost was justified by the business goal to reduce administrative costs.

Finally, business goals also helped explain architectural choices to upper management. The CTO and SOA architect pointed to the difficulty of explaining to upper management, and in particular non-technical management stakeholders, architectural design approaches. For example, upper management stakeholders were not easily able to understand the meaning of asynchronous or synchronous messaging. However, elaborating on the implication of a design approach on a stakeholder’s business goals, helped clarify the architectural choice in a manner relevant to those stakeholders.

Q2.1. What modeling constructs are appropriate for making business goals and the role they play during architectural decision-making explicit?

In this case study it was appropriate to represent business goals using softgoals to help indicate that business goals are tentative, often in conflict with each other, and negotiated. For example, in this case
study the business goal to minimize administrative costs had to be negotiated and traded-off with the costs of SOA evolution.

Intentional dependencies between organizational actors, such as between the Insurance Company, Business Strategy, Product Management and the technical actors, Consumer Component and SOA evolution, together with strategic rational graphs included in intentional actors, support representing how business goals, which originate at the business level are linked to NFRs discussed by stakeholders and designers during architectural decision-making.

Setting and propagating the priority attribute across softgoals supports indicating the priorities stakeholders and designers give the business goals and NFRs that guide the selection between alternative design choices. The intentional actors support indicating how organizational participants collectively propagate and directly or indirectly address business goals in the organization. Contribution links between softgoals indicate how stakeholders and designers refine business goals into NFRs, and NFRs into lower level NFRs towards architectural design solutions.

**Q1.2. What role do qualitative NFRs play during architectural decision-making to meet business goals?**

In this case study the SOA architect referred to NFRs when explaining the software engineering properties of each design approach. For example, the SOA architect explained that asynchronous messaging is more resource efficient and results in an extensible solution.

NFRs also played a role in revealing deeper reasons for design preferences. For example, asking why “simplifying design” is an important NFR to the consumer component designer lead to identifying the “reduce complex work” NFR, which in turn helped identify the “reduce development cost” NFR as the deeper reason for preferring synchronous messaging. Continuing “upwards” with why questions eventually helps reveal the business goals that ultimately justify stated NFRs and consequently designers design preferences.

In this case study NFRs also played a role in comparing and contrasting the design argumentation put forward by different designers during architectural design reasoning. The NFRs articulated by the component designer were evaluated against the NFRs mentioned by the SOA architect. By discussing NFRs of each design solution side-by-side the SOA architect hoped to convince the component designer to agree to an asynchronous design approach.
Q2.2. What modeling constructs are appropriate for making qualitative NFRs and the role they play during architectural decision-making to meet business goals explicit?

In this case study the strategic rationale graphs supported capturing the different NFRs mentioned by the component designer and the SOA architect. This helped make explicit the reason for their respective preferences, as well as reveal and represent the higher-level design goals that underpin and justify their NFRs.

In this case study the strategic rationale graph in each intentional viewpoint represented the reasons for choosing one design approach over the other. For example, the intentional viewpoint of the SOA architect mainly represented the design reasoning for preferring asynchronous messaging, while the intentional viewpoint of the consumer component designer represented the design reasoning for preferring synchronous messaging. To reveal to a fuller extent the design tradeoffs one design approach makes needs consulting the other intentional viewpoint. For example, the reasons for choosing asynchronous messaging are traded off when choosing synchronous messaging.

In this case study focusing each intentional viewpoint on the reasons for choosing a design approach helped simplify the diagram and make it easier to follow, and helped sharpen to comparison between the two approaches.

Q1.3. What role does the organizational setting play during architectural decision-making to meet business goals?

During this case study the organizational setting played several roles. First, different organizational participants were assigned different decision-making responsibilities in the organization each contributing in a distinct way to achieving business goals. Product management was for example tasked with translating higher-level business goals into requirements for products and infrastructures. To support meeting future business goals, the SOA architect was made responsible for evolving the current enterprise architecture towards a service-oriented architecture. Component designers on the other hand were responsible for the design and implementation of capabilities currently needed in the organization.

Some organizational participants also served as arbitrators in deciding on which design goals should be given priority. For example, product management decided whether the design goals achieved by
asynchronous messaging should be given higher priority than the design goals of the component designers. Finally, different organizational participants advocated for different architectural choices, which helped explore the design reasoning and arguments during architectural decision-making, increasing the quality of decision-making in the organization.

Q2.3. What modeling constructs are appropriate for making organizational setting and the role the organizational setting plays during architectural decision-making to meet business goals explicit?

In this case study the intentional actor constructs supported representing and delineating different decision-making responsibilities in the organizational setting as well as different reasoned vantage points designers took in relation to design decision-making. The Insurance Company collective represented the organizational responsibility of the company as a whole vis-à-vis its external customers, and internal stakeholders and designers. The Business Strategy role indicated the responsibility of formulating a business strategy that addressed and prioritized the important business goals of the organization, while product management’s responsibility indicated how business goals are further refined and translated into features and NFRs for software development. The intentional viewpoint construct supported indicating the different vantage points the consumer component designer and the SOA architect took on the synchronous vs. asynchronous design discussion.

Structuring design reasoning and decision-making along intentional actors also supported indicating where architectural conflicts arose in the organization, and who in the organization was responsible for arbitrating the conflict based on the higher-level goals articulated of the organization.

Q1.4. What role do quantitative NFRs play during architectural decision-making to meet business goals? Q2.4. What modeling constructs are appropriate for making quantitative NFRs and the role they play during the architectural decision-making explicit?

This case study did not include quantitative NFRs.

6.7. Modeling Discussion

Roles vs. Viewpoints: During architectural evolution, enterprise application designers and enterprise-wide architects often have overlapping organizational design responsibilities. For example, the
consumer component designer is responsible for the design of a component that communicates with another system in the enterprise, at the same time that the enterprise-wide architect’s responsibilities include dealing with the communication infrastructure within the enterprise. Due to this overlap of responsibility, there is no distribution of decision-making, but rather distribution of decision authority, with a need to find design solutions cooperatively and jointly. Figure 6-3 illustrates how the consumer component designer and an enterprise-wide architect reasoned differently about the design issue. Intentional viewpoints represent such distinct but overlapping design responsibilities.

In contrast, intentional roles represent decision-making that is non-overlapping, both in terms of responsibilities and authority. For example, the SOA architect’s design decisions relate to the messaging system itself, such as how to design synchronous or asynchronous messaging, is a distinct responsibility that does not overlap with the responsibilities of enterprise application designers. Such non-overlapping responsibilities could then be represented by other intentional actor concepts, including intentional roles.

The intentional role concept is appropriate for representing design responsibilities related to the messaging system, even though the design properties of the messaging system have a direct effect on the non-functional properties of the enterprise applications that make use of the messaging system, (such as performance, reliability, security, maintainability, and extensibility of enterprise applications). These effects can then be captured by intentional dependencies between different intentional actors, since these require dealing with distributed decision-making with non-overlapping responsibilities.

**Bottom up intentional viewpoint analysis:** A bottom up analysis approach was used to construct the intentional viewpoints. First, the alternative design approaches were identified, along with the designers who argued for each of the approaches and the designers’ respective design responsibilities in the organization. Next, “why” questions were asked, to elicit their reasons for favoring one design approach over the other. Once top-level design goals were identified, it was investigated from where and whom in the organization these goals originated. Repeatedly following this analysis process helped identify the higher-level stakeholders, their goals and their reasoning, including the highest-level business goals of the organization relevant to the design question at hand. Once these were identified, questions were asked to identify why those goals were important to the organization. This line of questioning led to the identification of a typical organization customer, and the customer’s expectations of the organization.

**Trade-off modeling in intentional viewpoints:** The design reasoning included in each intentional viewpoint mainly captures arguments in favor of a specific design approach. Tradeoffs can then be intuited by putting the viewpoints side-by-side. However, intentional viewpoints can also include
tradeoffs. For example, the SOA evolution intentional viewpoint includes both positive and negative contributions. The negative contributions come to strengthen the argument of the SOA architect, indicating where the alternative has negative implications.

**Common goals across intentional viewpoints:** Different intentional agents may discuss reasoning using terminology at different levels of abstraction. Identifying overlapping hardgoals may therefore require some effort. For example, an enterprise application designer may define a concrete hardgoal “transmitting new insurance data to new policy processing subsystems”, while an enterprise-wide architect may define the hardgoal as “messaging between Consumer components and Provider components”. Although these hardgoal are at different levels of abstraction, from a system’s user perspective both refer to the same intended functionality.

Softgoals can also be used to define organizational responsibilities. For example, security of messaging and performance of messaging are responsibilities allocatable to architects and designers. However, softgoals are used to qualify hardgoals, such as the “messaging” hardgoal. Overlaps are therefore defined based on the associated hardgoals.

### 6.8. Conclusion

The agent and goal-oriented modeling and analysis technique was applied while the CTO and the SOA architect of the insurance company were involved in explaining to relevant organizational stakeholders and component designers (including the Consumer component designers) the merits of using asynchronous messaging over synchronous. Both the CTO and SOA architect had found that it was challenging to discuss the different SOA related design intents informally, and why design goals of different application designers are better traded-off to achieve longer termed design objectives. Furthermore, articulating how such technical design discussions relate to, and possibly impact, objectives of higher-level stakeholders in the organization was also found to be challenging. Both, therefore, perceived the value of agent and goal-oriented models produced, and their ability to systematically capture, compare, and explain key discussion points, and relate those discussions to the broader organizational decision-making context of middle and upper managements.

While the modeling approach can assist in documenting and communicating key design discussion points amongst stakeholders and designers, the CTO and SOA architect identified tool support as being crucial for adopting such a modeling and analysis approach within an enterprise setting.
7. IAL Applied to the “TeleLarm AB” Design Pattern Literature Case Study

7.1. Introduction

Software system designers often use design terminology that succinctly refers to design decisions and their known non-functional properties. Discussing design solutions in such a “packaged” manner helps raise the level of abstraction during design discussions and decision-making, which improves the efficiency of decision-making processes. “Design patterns” are a typical example of packaged design solutions (Gamma et al., 1995).

During case studies, the need to consider coarser-grained constructs and patterns when dealing with non-functional requirements during design became apparent. In particular, designers often thought in terms of solutions that addressed a variety of NFRs concurrently, rather than in terms of finer-grained qualitative goal refinements, or finer grained solution approaches that addresses single NFRs.

In the case study reported in this chapter, NFR goal graphs were applied to capture and analyze design reasoning described in design patterns, and when patterns are successively applied during software design. The design patterns analyzed were reported in the literature (Molin & Ohlsson, 1998). The patterns reported were documented after a major architectural redesign effort at the Swedish TeleLarm AB security company, during which the architectural design of existing alarm systems were evolved to use a newly developed object-oriented framework. The design patterns were created in an attempt to capture the key architectural design decisions and related rationales that made the object-oriented framework a successful basis for developing and evolving the company’s alarm systems.

The analysis approach recognizes that the “forces” that shape the designers’ understanding of the problem, the solution, and the solution rationale documented in the captured design patterns are, in fact, non-functional requirements that can be represented and analyzed. Using goal graphs makes the reasoning structure behind a design pattern explicit and amenable to systematic analysis. This helps focus on the non-functional requirements that the pattern addresses, the tradeoffs that are involved,

---

21 This chapter is based on (Daniel Gross & Yu, 2001d). The second author suggested the idea to analyze design patterns using concepts derived from the NFR framework. The first author identified which patterns in the pattern literature to study, and worked out the modeling and analysis presented in this chapter. The paper was written by the first author, and edited by the second.
and how requirements are met when the pattern is applied. More specifically, this case study illustrates how NFR goal graphs help to achieve the following:

- **Clarify the role of NFRs:** In a design pattern, the NFRs are explicitly represented. This makes them first-class entities, which enables referring to and analyzing the role they play in the argumentation structure of that particular design pattern.

- **Provide structure for characterizing each pattern:** By relating the NFRs to each other and to the alternative solutions discussed in the patterns, NFRs become the criteria for evaluating whether to accept or reject a solution. Making such relationships explicit allows characterizing pattern solutions in terms of their NFR-related properties.

- **Systematically support applying patterns during design:** Since NFRs need to be systematically addressed during design, by characterizing patterns with respect to NFRs, it becomes easier to address system-wide NFRs when patterns are retrieved, selected, and then successively applied to the current design situation.

- **Support forward engineering and traceability:** NFRs become focal points to be addressed by patterns. Designers are guided by addressing NFRs when searching, retrieving and selecting patterns. By keeping track of how NFRs drive the pattern application during design, the evolution of the system structures are retained which thus facilitates NFR-related traceability.

This chapter revises the goal graphs presented in (Daniel Gross & Yu, 2001d), extending them with additional modeling elements. The chapter also presents an explanation how the goal graphs for design patterns were derived from the pattern description presented by Molin and Ohlsson. Finally, this chapter simplifies the global goal graphs constructed to represent the NFRs and the achievement of NFRs when applying design patterns during architectural design.

During this case study, the focus was on using NFR goal graphs to represent and analyze design patterns and the application of patterns during architectural design. Actor-oriented representation and analysis of the design patterns have been left for future work.

### 7.2. Design Patterns

Christopher Alexander, the renowned (building) architect is widely acknowledged to be the originator of the pattern idea. He explains that “each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution ... as an element in the world, each
pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly
in that context, and a certain spatial configuration which allows these forces to resolve themselves”
(Alexander, 1979).

Jim Coplien, a noted researcher and practitioner in the software patterns community, explains that a
pattern is a “piece of literature that describes a design problem and a general solution for the problem
in a particular context”, and elaborates that “if we understand the forces in a pattern, then we
understand the problem (because we understand the trade-offs) and the solution (because we know
how it balances the forces), and we can intuit much of the rationale. For this reason forces are the focus
of a pattern.” (Coplien, 1996).

For example, in discussing software design using pattern-oriented terminology, a designer may refer
to applying an “Observer Pattern” to solve a particular software design problem, thereby compactly
referring to a design approach that exhibits several beneficiary NFRs and several known tradeoffs. The
designer would also be implicitly rejecting several possible alternative design approaches. Design
patterns, as “packaged solutions”, are often the subject of design discussions and are represented as
design tasks in larger goal graphs and actor models.

7.3. Description of Case Study

Molin and Ohlsson identified and documented several design patterns
during the redesign of a commercial real-time alarm system (Molin & Ohlsson,
1998, pp. 3,4). The patterns addressed a
variety of requirements and design
issues, including performance
optimization, optimal utilization of
limited memory, reliability (such as fault
tolerance), maintainability, and
portability.

Figure 7-1 shows a schematic diagram
of a typical alarm system layout,
illustrating the control nodes, sensors,
output devices, and communication lines between them. The control nodes are autonomous processors, with each control node having a set of input sensors and/or output devices connected to it through communication lines. Control nodes are also interconnected between themselves through communication lines.

Figure 7-2 shows an object diagram that describes the design of a control node in the above system. The diagram shows that an **AlarmHandler** is responsible for surveying the environment. The **DevicePoller** retrieves input status of all input devices connected to the control node, and stores all inputs as **SystemStatus** objects. An **AlarmList** keeps track of all the alarms that the locally attached output devices are responsible for, by managing a mapping between the locally attached output devices and the subset of **SystemStatus** objects that are needed to check for the occurrence of an alarm. The **AlarmList** is also responsible for checking, on behalf of the **AlarmHandler**, the system status with respect to each defined alarm. Finally, the **AlarmList** communicates with **OutputDevice** objects, so that they may write to the locally attached output devices either to report the current system state or to activate actuators in the environment to perform emergency actions. Note that **DevicePoller** and **AlarmHandler** work concurrently within the control node.

Two principal problems arise with the above design: a) how to store system status data that is distributed across many control units, and b) the fact that input and output devices exhibit great variability in the type of sensors, their detection algorithms, access protocols and physical packaging. The “Deviation Pattern” addresses the first design problem of distribution, while the “Point Pattern” addresses the second problem of device variability.

### 7.4. Understanding, Analyzing, and Applying the Deviation Design Pattern

Dealing with computation units distributed within the monitored environment means that an output device may be dependent on a set of input devices regardless of where the input devices are physically...
located. An input device can be connected to the same control unit as the output device, or it can be connected to remote control unit. The ‘mapping’ between outputs devices and system status items defined in the **AlarmList** (for all output devices locally available) may need to refer to **SystemStatus** objects available locally and to **SystemStatus** objects that reside on other control nodes and that need to be fetched remotely. The “Deviation pattern” explains the design issue as follow (italics added):

The logical behaviour of the system is completely independent of its distributed nature; an actuator may depend on some particular input regardless of whether it is connected to the same control unit or some other control unit in the system. Furthermore, *the number of inputs in the system can be considerable*, and the number of control units can also be large.

*If every control unit were to store the current status of every input sensor, it would place heavy demands on the memory capacity of each control unit*. An alternative would be for each control unit to have only a proxy for remote inputs. Each proxy would consist of a system wide reference, and **any request would be forwarded to the control unit where the input actually existed**. However, even storing as little as one reference per input in each control unit requires a great deal of memory space (Molin & Ohlsson, 1998, pp. 3-4)

The deviation pattern first considers two design approaches: a) to replicate and store in each control unit all input status data available in the system, or b) to only store a proxy for remote inputs in each control unit and then forward and fetch status request on demand.

Analyzing the first design approach – to store in each control unit all input status data – in more detail, we observe that: a) the design approach requires duplicating all input status data on all control units; b) Memory capacity needs of each control unit is of concern; and c) an approach that duplicates input status data across all control units makes memory demands on the control units, in because there is a large number of inputs across the system.

A design goal approach is sought that reduces, or, if possible avoids, duplication of input status data within the system. The
The diagram in figure 7-3 captures the design reasoning of the first design approach.

Turning to the second design approach – to utilize a proxy in each control unit for each remote input status data in the system, we observe that: a) this approach indeed helps reduce duplication, since all input status data is stored only once in the system. However, even storing a reference only to all remote input status data in the system, still requires a great deal of memory. Beside the need to reduce data duplication in the system, there is also a need to reduce, and, if possible eliminate, the number of “on-demand” data reference items in the system. Figure 7-4 adds this reasoning to the goal graph.

Finally, the deviation pattern turns to the proposed solution:

On the other hand, the ratio of alarms, faults and disturbances to the total number of inputs is very low. Most of the sensors are in a normal state most of the time. All inputs need not therefore be known at each control unit since an unknown input can be assumed to be in a normal state. Only information about deviations from the normal state must therefore be globally accessible from each control unit.

Represent each detected deviation from the normal state as a Deviation object. Use Deviation subclasses, such as Alarm, Fault and Disturbance to represent different kinds of deviations. Let deviations be the unit of distribution in the system in the sense that all deviations are replicated to all
control units. Since the set of deviations defines the complete system state, this is immediately available on all nodes. (Molin & Ohlsson, 1998, p. 4)

The third design approach, which is the one adopted by the deviation pattern, is to store deviation objects in control units only. The pattern text argues that memory usage is much reduced because much of the input data status can be assumed normal for most of the time. The deviation design approach thus achieves optimal memory utilization by not storing data whose state can be assumed known. Figure 7-5 shows the third design approach and related design reasoning.

![Figure 7-5: Deviation pattern, third design solution and reasoning](image)

The pattern text above further indicates that the deviation pattern has an advantage that the system state is “immediately available on all nodes”. This suggests another design goal that the system designers had in mind, namely the access time of system state in each control node. At a higher level of abstraction, this concern relates to the performance of the system. The different design approaches differ with respect to access time: the first and the third have all the system state available locally, while, the proxy approach requires additional access to request and receive remote input status data. Figure 7-6 includes the performance related design goals and includes additional contribution and correlation links.

The goal graph supports identifying additional design reasoning details, not mentioned in the design pattern text. For example, to store deviation in each control node duplicates deviation data in each
control node, thereby trading off the optimal utilization of memory in control units for optimal access time of input status. This additional design tradeoff is indicated by a correlation link with a negative contribution from store deviation in each control unit design task to the reduce data duplication [control units.memory] softgoal. The correlation link with a positive contribution, from store deviation in each control unit design task, to optimize access time [input status], indicates the performance gained.

Figure 7-6: Deviation pattern approaches and reasoning structures

Given this tradeoff, the goal graph helps further explore the design space such as asking: how can one reduce the duplication of deviation data. One approach could be to store remote deviation data using a proxy. This positively contributes to reduced memory utilization, while trading off the access time.

Finally, the goal graph includes an additional crucial design tradeoff that the storing of deviation objects potentially makes. Namely, the potential trading off of the reliability for reduced memory capacity needs. Storing deviations only helps reduce memory capacity needs is based on the assumption that the number of deviant input status occurrences is very low relative to the number of input sensors. However, if this assumption fails, and the number of deviant occurrences is very large, then a very large number of deviant objects need to be stored in each control unit – which could lead to an “memory full”
exception effectively stopping the alarm processing during an emergency. In such unusual circumstances, the system would thus not work reliably.

In figure 7-6 this tradeoff is indicated by the correlation link from “store deviation in each control unit” design task to “Reliability [alarm processes]” softgoal. To indicate that this correlation is conditional on a large number of failures occurring at once, the belief node “number of deviant input status is very high” is added, and linked via a “make” contribution link to the correlation link.

**Applying deviation pattern during architectural design:** Let us now turn to applying the pattern during design. Figure 7-7 shows non-functional requirements mentioned in the paper’s introduction (denoted by the softgoal “clouds” on the left-hand side) together with a first elaboration of functionality the system should be capable of performing. The functional elaboration is represented using intentional design task, each indicating the functionality in need for design. These include surveying the physical environment by detecting out-of-the-ordinary occurrences and responding to those occurrences by reporting them on output devices and, if necessary, activating actuators. The functionality elaborated are the essential functional capabilities of an alarm system.

![Figure 7-7: Functional and non-functional elaboration of alarm system processes](image)

At this stage of the design, the designer knows that:

a) each control unit participates in surveying the environment, and to evaluate whether an out-of-ordinary occurrence happened, the surveying processes need to know the sensor status of various sensors in the system, and that;
b) some sensors are connected locally to the control unit while others can be connected to remote control units; therefore,
c) some input sensor data must be made available locally.

One obvious design choice is to replicate all input sensor data to all control units. However, the designer immediately realizes that this would require a lot of memory capacity, in particular given the large number of sensors in the system. The design question therefore is: how to represent the system state in a compact manner, that is to reduce the memory capacity needs for control units.

As discussed earlier, the deviation pattern answers this design question. Figure 7-8 shows how the deviation pattern is applied to the memory capacity NFR as well as contributions of the pattern on other NFRs. The figure also shows how the deviation pattern links to, and helps expand, the additional elements in functional elaboration structure.

![Figure 7-8: Applying deviation pattern during design](image)

On the left-hand side is the **use deviation pattern** design task—the internal reasoning structure and alternative design approaches are hidden. Also shown is a contribution link with a positive contribution from the **use deviation pattern** design task to the **reduce capacity needs [control units.memory]** softgoal, as well as a belief element indicating the assumption — “# of faults are very minimal relative to the number of input sensors” — that supports the positive contribution. The use deviation pattern design
goal also includes a positive correlation link to the **optimize access time [input status]** softgoal—another top-level softgoal included in the deviation pattern goal graph. The **optimize access time [input status]** softgoal contributes positively via the **performance [control units. Processes]** to the **Performance [alarm system]** top-level softgoal/non-functional requirement.

Figure 7-8 also shows how the reliability NFRs are affected by use of the deviation pattern: the negative correlation link from the **use deviation pattern** softgoal to the **Reliability [alarm processes]** softgoal, which in turn positively contributes to the **Reliability [alarm system]** softgoal. The belief **number of faults very high** further indicates the assumption that must hold for reliability to suffer. In usual cases this belief is not satisficed, and hence the negative reliability is not supported.

Finally, figure 7-8 shows how the **use deviation pattern** design task is linked to the functional elaboration structure on the right-hand side. To indicate alternative design approaches the functional elaboration structure is extended with hardgoals and alternative design tasks. For example, the **store input status data** design task is decomposed in the **input status data stored** hardgoal. Two design approaches for storing input status data are included: to **store and disseminate deviation objects** or to **store and disseminate regular status objects**. The first approach is then linked to the use deviation pattern design goal via the **accepted** link – indicating that the first approach is included after accepting the design solution proposed by the deviation pattern. The other design approach could be linked via a **rejected** link, to indicate these design approaches were rejected by the deviation pattern. To reduce clutter, these links are not included in the diagram.

Figure 7-9 shows the solution structure after applying the Deviation Pattern. Instead of storing every input status as a “system status” object, only “deviation” objects (those that represent input status...
information that exceeds certain threshold limits) are stored. **ReplicationHandler** was also added to replicate a copy of the “deviation” objects to all control units. The **Alarm List** object is now aware that it refers to deviation objects only rather than to all system status objects. It is also aware that system states that do *not* point to a deviation object are *not* exceeding a threshold are thus in a normal state.

### 7.5. Understanding, Analyzing, and Applying the Point Pattern

This section illustrates NFR reasoning performed when more than one pattern is successively applied to a system under design. This section also illustrates the inclusion of a design pattern in the goal graph capturing intents and choices of another pattern.

The Point Pattern addresses the problem of variability amongst I/O devices – different sensor types, detection algorithms, access protocols, physical packaging -and the means by which sensors, actuators, and other output devices are connected to the system. Despite the variations in the make-up of these various devices (sensors, actuators and output devices), the logical behavior, such as requesting status or activating actuators, is similar. The Point Pattern problem statement therefore is: How can the logical behavior of the system be separated from the variation among input sensors and output actuators? In other words, how to standardized a device access interface which separates the logical behavior from device variations?

The Point Pattern provides two alternative strategies for achieving a standardized service interface (bold italics added):

One possibility would be to have an **abstract base class** which defines this standard interface and uses subclasses to implement the variations among different devices. But a sensor is often the source of two kinds of **Deviation**, both **Alarms** and **Faults**. Furthermore, a device may contain more than one sensor or actuator. Using inheritance only would thus *not prevent the variations in packaging from affecting the interface*. (Molin & Ohlsson, 1998, p. 5)

Figure 7-10 represents the first design approach, and the reasoning against this approach. At the bottom is the **use abstract base class** design approach captured by a design task. This design approach, in principle contributes positively to **standardize the I/O device access**. However, a belief element is added to indicate that **I/O device variability is not fully shielded by use of inheritance only**. This belief breaks the aforementioned positive contribution – indicating that using an abstract base class, in this design domain, is not sufficiently contributing to standardization.
Although not mentioned in the pattern text above, figure 7-10 includes the additional softgoals extensibility [IO devices] and modifiability [IO device access] to further indicate that having a standardized interface for IO device access contributes positively to these higher level design properties.

A second, and preferred, approach is then suggested:

Define the interface between the part of the system representing the logical behaviour and the part implementing different kinds of devices as a set of Points, either InputPoints or OutputPoints. A Point has a binary state, either active or inactive, without references to implementation details of how this state is represented or maintained.

Use the Bridge pattern to connect Points to separate Device objects that implement actual I/O-behaviour of the Points. Devices encapsulate access protocols and other details of how they perform actual I/O from the Points as well as whether they are polled or interrupt driven. Let the number and kind of Points connected to the same Device be determined by the characteristics of the physical devices.

The second design approach included in figure 7-11 is represented by the use point abstraction design task. As indicated in figure 7-11 the design approach is composed of two more detailed approaches: an abstract logical base class design task and a bridge pattern to connect logical class to implementation design task.

Figure 7-11 shows that the Point Pattern does, in fact address two distinct issues. First, it addresses the additional variability of devices by suggesting a logical rather than device-oriented abstract service interface. This is addressed by the abstract logical base class design task. Second, the point pattern also addresses the issue of separating implementation code from interface code through the Bridge Pattern. This is a related but different concern, and is shown by the bridge pattern to connect logical class to implementation design task.

The Bridge Pattern (Gamma et al., 1995) decouples the “point abstraction” from the “point
implementation code”. This addresses extensibility issues, allowing for new types of devices to be added without affecting the point “client” code. The Bridge Pattern introduces another layer of indirection (polymorphic invocation) between the clients’ interface and the implementation code. This has some negative effect on the performance of the system when accessing I/O devices (such as accessing sensor-state information); this is not mentioned in the pattern text description but can be identified when analyzing the bridge pattern for its NFRs and tradeoffs.

Figure 7-11: Point pattern reasoning structure

Figure 7-12 also illustrates how applying the Point Pattern may have an impact on non-functional requirements (“forces”) already addressed by having previously applied the Deviation Pattern. While design choices for optimizing performance have been made (by duplicating deviations, rather than, say, accessing them via proxies), there is no guarantee that performance would not be adversely affected during subsequent design. That is, the use that the Point Pattern makes of the Bridge Pattern may still, to some extent, adversely affect the overall system performance. It is thus crucial that the designer, when applying subsequent patterns, analyzes whether and in what way already-met non-functional
requirements may be negatively impacted. An NFR goal graph such as in figure 7-12 may help in making such analyses more systematic.

In this case, the softgoal topic component helps indicate what part of aspect of the system is causing these effects. The deviation pattern positively optimizes the access time of input status, while the point pattern negatively affects the access time of devices. When evaluating such conflicting contributions, the designers thus need to determine, which of the effects are more pronounced to affect the overall performance quality of the alarm system.

Figure 7-12 also shows how applying the point concurrently links to the several design tasks in the functional elaboration. Applying design patterns thus often crosscut functionality in systems. Figure 7-12 shows that the use point pattern design task is linked via an accepted link to the access sensors via point abstraction design task alternative, and the access actuators via point abstraction design task. To reduce clutter in the diagram the access output devices is not elaborated in similar respective alternative design tasks. The rejected links to the respective rejected design task alternative are again omitted.

Figure 7-12: Applying the Point Pattern

Figure 7-13 illustrates the solution structure after applying the Point Pattern. The Point object is used to access both input and output devices, and serves as the interface through which DevicePoller and
**AlarmList** access I/O devices respectively, while the **PointImp** object implements the different protocols to access I/O devices. There is a zero-to-many relationship between the devices and the **PointImp** object, which means that one input device may have several points defined if that input device offers several sensor services to the control node.

![System solution structure after applying the Point Pattern](image)

**Figure 7-13: System solution structure after applying the Point Pattern**

### 7.6. Responding to Research Questions

**Q1.1. What role do business goals play during the architectural decision-making?**

Although the alarm system discussed is a commercial product, Molin and Ohlsson mainly focused on non-functional properties of the alarm system architecture. Only a few business goals are mentioned, such as the need to comply with national and international standards, and the flexibility of system to comply with such different national standards. The authors do make it clear that non-functional requirements play an essential role in the commercial success of the alarm system. What exactly the business goals are, how these guided the identification, specification and prioritization of NFRs is not specifically discussed.

**Q1.2. What modeling constructs are appropriate for making business goals and**
the role they play during architectural decision-making explicit?

Since business goal and link specific link to NFRs were not explicitly discussed in the paper, they were also omitted from the modeling and analysis.

**Q2.1. What role do qualitative NFRs play during architectural decision-making to meet business goals?**

NFRs played a crucial role in explaining architectural design choices included in patterns. For example, a key driving force for the deviation pattern was to optimize memory utilization of control units, and to quickly access input status data during out-of-the-ordinary occurrences. Similarly, key drivers for the point pattern were the definition of standardized interfaces while dealing with much variability in the physical construction and access of input and output devices.

NFRs also helped explain the design tradeoffs (“forces in pattern terminology”) discussed in design patterns. For example, that the deviation pattern traded off memory utilization for access time to deviation objects, and that the point pattern traded off performance for flexibility in dealing with different device implementation needs.

This case study further demonstrated that NFRs played a role when applying patterns during architectural design. NFRs guided the selection of applicable patterns. For example, the NFR to reduce memory utilization needs of control units, prompted the designer to consider the deviation pattern. Likewise, during architectural design, the need to deal with much variability in I/O devices, while constructing a maintainable system prompted the need for a point pattern approach.

**Q2.2. What modeling constructs are appropriate for making qualitative NFRs and the role they play during architectural decision-making explicit?**

This case study illustrated that the softgoal concept together with goal graph modeling supported describing the reasoning structures presented in pattern texts. Softgoals capturing NFRs indicate the forces in the description of design patterns. Constructing a goal graph for the deviation and the point pattern helps clarify the role NFRs play to support key design arguments included in design pattern, and why alternative approaches were considered less optimal.

Softgoals, such performance, reliability, maintainability, included in a “global” goal graph, help indicate the higher-level design goals during architectural design of the alarm system. The global goal
graph further helped indicate how the application of design pattern help address global NFRs, as well as the effect on NFRs when multiple design patterns are applied during the architectural design process.

Q3.1. What role does the organizational setting play during architectural decision-making to meet business goals? Q3.2. What modeling constructs are appropriate for making the organizational setting, and the role the organizational setting plays during architectural decision-making to meet business goals explicit?

This case study did not deal with representing the organization in which the alarm system architecture was developed.

Q4.1. What role do quantitative NFRs play during architectural decision-making to meet business goals? Q4.2. What modeling constructs are appropriate for making quantitative NFRs and the role they play during the architectural decision-making explicitly?

This case study did not include quantitative NFRs.

7.7. Modeling Discussion

NFR discussions and NFR goal graph modeling: This case study illustrated that goal graphs support a more systematic analysis of the reasoning argumentation included in design patterns. For example, during the construction of a goal graph for the deviation pattern, it was possible to identify that storing deviation objects in fact duplicates deviant status data across different control units – something not explicitly mentioned in the pattern text. This was made possible by explicitly representing the reduce data duplication softgoal in the goal graph, and exploring the contribution each design approach made, if any, to this softgoal.

Capturing assumptions as Beliefs also helped explore what happens when assumptions fail. During the analysis of the deviation pattern, exploring failure of assumptions lead to identifying that the deviation pattern in fact trades of reliability for optimal memory utilization – a circumstance not mentioned in the pattern text.

Capturing packaged solution and levels of abstraction: Molin and Ohlsson (1998) discuss what the level of generality ought to be for documenting their patterns. They explain that they documented
patterns to be specific to fire alarm systems, but acknowledge that most of their patterns could be rewritten for a more generalized system to making the patterns relevant to a wider audience.

However, the authors point out that generalizing the patterns makes them more difficult to understand and to apply during design. From a modeling point of view, the level of generality of packaged solutions can be adjusted by the level of detail chosen for defining hardgoals and topics in softgoals. For example, in this case study the hardgoal sensor accessed or the softgoal topic “alarm processes” makes the goal graph specific to alarm system.

7.8. Conclusion

Designers discuss and reason about software design while referring to packaged design solutions. The software pattern approach illustrates such design thinking, and offers some thinking tools to further help deal with packaged solutions during system design. In this section we illustrated a more systematic approach to dealing with design patterns while documenting, analyzing, and understanding design patterns and also when applying the design patterns during system design.

By indicating that packaged solutions are coarse-grained design tasks in themselves the first step towards integrating packaged solutions with intentional actors was taken. However, a finer grained integration may be necessary that takes into account how packaged solutions establish design structures that involve more than one focal point of decision-making, thus requiring the introduction of several intentional actor into the design space.

For example, introducing the deviation patterns requires the collaboration of local and remote control units in an alarm system. However, if different control unit types are designed by different organizational units, or even different vendors, this may affect the feasibility of applying the deviation pattern across all units. Such design circumstances would call for an intentional analysis of distributed intents and decision-making.
8. IAL Applied to the Modeling and Analysis of Qualitative and Quantitative NFRs During an Industrial Project

8.1. Introduction

This chapter presents a goal- and scenario-oriented modeling and analysis technique that supports specifying qualitative and quantitative NFRs, in an integrated manner, during the requirements analysis of an application platform for industrial control systems. During this case study, a method, named the Platform Non-functional Development (PND) method (Song, Hwong, & Ros, 2009), which was developed at Siemens Corporate Research (SCR), was enhanced with a goal and scenario-oriented modeling technique. The proposed method was named: GS-PND – for Goal and Scenario-oriented PND.

The PND method was constructed based on experience gathered during an ongoing requirements analysis effort during which quantitative non-functional requirements were specified for a service-oriented platform. The PND method offers guidelines to support analyzing the quantitative NFRs of existing control system applications, each defined for a different application domain, to derive the quantitative NFRs for the service-oriented application platform.

The objective of this case study was to develop a model based method that offered a more systematic approach to deriving quantitative NFRs from existing domain application for the application platform, and to support integrating in the analysis both quantitative and qualitative NFRs.

During this case study a requirements document for the application platform was analyzed, as well as several excel spreadsheet documents that included quantitative non-functional requirements gathered from several stakeholders belonging to different Siemens daughter companies, each responsible for the requirements of a different industrial control system. Each control system for which qualitative requirements were gathered was developed by a daughter company for a specific target market, such as transportation, security and the like. The requirements document further included a high-level architecture of the application platform. This architecture was presented in form of a detailed block and arrow diagram. The spreadsheet documents included several hardware architectures of the deployed

---

This chapter appears in Developing Non-Functional Requirements for a Service-Oriented Application Platform – a Goal and Scenario-Oriented approach authored by Gross, D., Yu, E., & Song, X. (2011), in N. Milanovic (Ed.), Non-Functional Properties in Service Oriented Architecture: Requirements, Models and Methods (pp. 24-47). Hershey, New York: Information Science Reference, Copyright 2011, IGI Global, www.igi-global.com. Reprinted by permission of the publisher. This chapter is largely based on the aforementioned publication. The first author was the principal author of this study. The paper was edited and commented on by the second and third authors.
control systems, such as hardware architectures for a control system deployed as an embedded system, as moderate sized client/server PC-based system, as well as a multi server, multi site distributed system.

During this case study these documents were analyzed, and the GS-PND modeling approach that offers an integrated approach to analyzing quantitative and qualitative non-functional requirements was developed. At the conclusion of this case study the modeling and analysis approach and derived non-functional requirements for the application platform were presented in front the researchers and practitioners involved in the development of the application platform.

Beside two researchers (one who sponsored this research, and another one who had prior exposure to this modeling notation), none of the attending researchers and practitioners had prior exposure to the modeling notation employed. During the presentation positive feedback on the utility of the modeling and analysis method were received, as well as suggests for further improvements made.

This chapter presents the resulting GS-PND method. The main modeling and analysis features illustrated in this chapter include:

- Specifying quantitative NFRs to support dealing with different terminology and metrics used to specify quantitative NFRs in different application domains
- Consolidating quantitative NFRs to identify quantitative NFRs for the application platform
- Identifying architectural design approaches to dealing with qualitative NFRs.
- Specifying and integrating the high-level architectural structures and processes associated with architectural design approaches
- Analyzing the effect of different high-level architectural design configurations on quantitative platform NFRs

8.2. Description of Case Study

8.2.1. Difficulty in specifying quantitative NFRs for an application platform

Quantitative NFRs are NFRs such as the number of connected clients a system needs to support concurrently, or the time in which a software process must be completed. Such quantitative NFRs play an important role during architectural design. They often must also be considered together with qualitative NFRs in an integrated manner. This is particularly the case during the design of an application platform that aims to support a large number of domain-specific applications in meeting functional and non-functional requirements.

The PND method was developed to address several main challenges that were identified while
analyzing and specifying the quantitative NFRs for an application platform for control systems (Song et al., 2009). These challenges include:

**Varying domain-specific needs:** Different application domains give rise to different NFRs. For example, a solution system that supports automation in manufacturing, which often requires meeting tight real-time constraints, and a solution system that provides building security in factories, where real time requirements are much less demanding, have quite distinct NFRs, even if both systems involve much of the same control functionalities.

**Varying deployment configurations and load conditions:** Solution systems can be deployed and operated in different configurations and under different load conditions. For example, the application platform needs to support an application that is deployed and operated as an embedded standalone system responding to several hundred events per minute, and an application deployed as an integrated multi-site system of dedicated servers responding to tens of thousands of events per second. What non-functional platform requirements should be specified so that, once implemented, the platform can be deployed on an embedded standalone system or on dedicated servers, and respond well for configurations?

**Terminology and metrics mismatch:** During the development of NFRs for the platform, requirements analysts must deal with a wide range of concepts, terminology, and metrics used in different application domains. For example, in one industrial domain, a performance requirement could be specified in events per second, while in another, in alarms per minute. Platform developers must translate such differences in terminology and metrics into common platform terminology and metrics before compatible platform requirements can be specified.

**Dealing with NFR tradeoffs:** Developing an application platform in general and adopting service orientation in particular requires the implementation of specific design principles, such as modularity, loose coupling, service statelessness, etc. (Erl, 2007), which help achieve non-functional benefits such as reusability, interoperability, consistency, and the like that are associated with application platforms and service orientation. However implementing such design principles comes at a price and requires developers to trade off other relevant NFRs, such as performance and/or reasonable upfront development costs. Platform developers must evaluate the importance of each NFR to the success of the domain application. Such evaluation determines whether the NFRs are inconsistent with the service-oriented design principles. If so, the platform developers, when establishing a service-oriented application platform, or when adopting the platform for domain applications, must determine what tradeoffs must be made to platform NFRs, or domain application NFRs, respectively.
8.2.2. **Industrial control systems**

The purpose of an industrial control system is to control processes within an environment (Speck, 2003; Sperling & Lutz, 1997). Essentially, a control system continuously reads input data from a number of input sensors, and feeds the data into control algorithms, which calculate some output control data. Output control data is then sent to actuators, which effect appropriate physical changes to the controlled process. These changes are in turn reflected in new input data that is read from input sensors (see Figure 8-1). Control systems also support displaying to users the data captured and calculated from sensors, and support users in configuring the control system and its various components.

Industrial control systems usually have a large number of input sensors and actuators connected, and continuously input, process, and output large volumes of data. A key non-functional requirement of a control system is that input data must be processed and outputted under real time constraints: the control system must be designed so that it offers sufficient processing throughput of input data.

For example, in one domain application (let’s call it the “TT” application23) a key throughput requirement is that the system is capable of detecting, processing and responding to 3000 changes of input value on its input sensors per second, while in another domain application (the “AT” application) the throughput requirement is to process 30,000 changes of values per second.

![Figure 8-1: Context diagram of control systems](image)

Apart from providing sufficient processing throughput, the commercial success of an industrial control system also depends on meeting other important NFRs. Developers of industrial control systems

---

23 “TT”, “AT” and the like are the code names given to applications during this research
must address NFRs such as cost of ownership, usability of the operators user interface, availability, reliability and interoperability of the control systems, as well as reusability, scalability and flexibility to changing controlling environments and customer needs. Industrial control systems are also often implemented on system hardware with limited capabilities. A key concern when addressing such aforementioned NFRs is whether hard real-time constraints can be met, if all else is kept equal, and if not, what tradeoffs amongst NFRs to make.

Figure 8-2 schematically illustrates a typical control system and several characterizing measures using the Use Case Maps (UCM) notation (Buhr & Casselman, 1996)\textsuperscript{24}. In the UCM notation, a wiggly line represents a process and an “x” on a wiggly line a computational responsibility, such as a control process step. Boxes represent architectural elements, such as applications, operating system subsystems, components, and the like. Together, the wiggly lines and boxes in figure 8-2 define a UCM. The purpose of a UCM is not to provide a complete description of a computational process, but to support capturing some essential computational structures and responsibilities at a higher level.

Control systems are usually client/server systems, where servers are connected to input sensors and output actuators, and clients provide user interfaces to the system’s operators. Clients are connected to the servers via a communication system. Standalone control systems usually combine client and server functionality in one physical device.

Important parameters that characterize control systems include the number of input sensors, the frequency these must be read and processed, the number of clients to which each server is connected, and so forth. Figure 8-2 shows a typical control process that reads data from an input sensor, performs some processing, then outputs control data and status information to clients. Internally, control systems maintain a data model, which stores data items such as data read, alarms identified and processed, as well as information about connected clients. Figure 8-2 schematically illustrates some modularization and layering structures which help address relevant non-functional requirements such as reusability, maintainability, and extensibility (Speck, 2003).

Specifying NFRs for an industrial control system typically involves a number of tradeoffs. The number of data points and hard real time requirements, such as detecting and processing 3000 changes of input values per second, establish baseline processing throughput needs for processing input data. Other

\textsuperscript{24} UCM, together with the Goal Requirements Language (GRL), a variant of i*, was approved as an international standard Z.151 for a User Requirements Notation (URN) by the ITU.T in November 2008.
relevant NFRs, such as usability, scalability, and interoperability, require additional system and software processes and structures, which add processing overhead to the baseline. Supporting additional throughput requirements requires specifying additional and/or more powerful processors, and more memory, increasing the cost of ownership. Therefore, to reduce cost of ownership, developers must either sacrifice some relevant NFRs, or reduce the system's throughput needs by reducing the number of input data processing needs—the latter can, for example, be achieved by positioning the product in a less demanding market niche.

Figure 8-2: Anatomy of a typical control system

8.2.3. NFRs of Platform, Platform Application, and Domain Application

Application platforms offer common runtime facilities and programming interfaces to application program developers. Successful application platforms are usually developed by analyzing already-existing domain applications to identify common reusable functionality and software assets. A platform NFR specification should thus be derived from NFR specifications of already existing domain applications.

To specify platform NFRs, an important distinction can be made between a “deployed application
system” and an “application” (see figure 8-3). A deployed application system, or “deployed application” in short, is an application deployed according to one of its predefined deployment configurations. For example, consider a building security control application that can be deployed as a small standalone embedded system for small homes or as a larger distributed client/server system for larger office buildings. In this example, the building security control application has two deployment configurations: an embedded configuration and a client/server configuration. The building security application can therefore be deployed as either of two types of systems: an embedded system, or a client server system.

Figure 8-3: Defining NFR specifications, an overview

Making this distinction is important, since some NFRs (such as load conditions—alarms per second that must be processed) are only meaningfully defined for deployed application systems. For example, the building security application can deal with a much greater number alarms per second when deployed as a client/server system, than when it is deployed as an embedded system.

According to this distinction, platform NFR development involves the development of two types of NFRs: a) NFRs that are defined independently of a deployment configuration, and b) NFRs that are defined for one or more predefined deployment configurations. In this chapter, unless indicated otherwise, “platform NFR” will refer to NFRs of either type.

Figure 8-3 further shows that a deployment configuration is defined by one or more deployment...
parameters, such as number and types of servers, number of clients, number of input sensors, number of configurable alarms, and so forth. Deployment parameters which characterize deployment configurations are NFRs specified for an application.

Another important distinction included in figure 8-3 is between quantitative and qualitative NFRs. Distinguishing between them enables us to use different modeling and analysis techniques during the development of platform NFRs. Deployment parameters and load conditions are both quantitative NFRs, i.e., NFRs specified in terms of countable quantities (R Kazman et al., 2000; Keller, Kahn, & Panara, 1990). In contrast, NFRs such as usability, security, interoperability, which are hard or impossible to formalize or count, are qualitative NFRs are (Chung, 1993; Chung et al., 2000; Mylopoulos et al., 1992). However, in a subsequent section we will further see that in the context of a qualitative NFR analysis technique, a quantitative NFR, such as processing throughput, can also be treated as a qualitative NFR.

Chung (1993) suggests sorting qualitative NFRs into “type” and “topic”. A type, such as scalability, is applied to a topic, such as defining the qualitative NFR “Scalability of System”. In figure 8-3, Topic captures the fact that Qualitative NFRs are defined over Applications or Deployed Application systems (type is not separately shown). The subsequent sections further clarify the difference between Quantitative and Qualitative NFRs, and elaborate on other type and topic distinctions.

Figure 8-3 indicates that the platform defines services. Platform Application is an application that uses the platform’s services. We distinguish between two types of services: application services and implementation services (Song et al., 2009). Application services are those that can directly be used by applications to add required functionality. Implementation services, on the other hand, are similar to programming libraries, and support implementing functionality of an application. Finally, Figure 8-3 also shows that the case of migrating Domain Application(s) to the platform also requires migrating functionality of the Domain Application to the platform services.

The structural relationships between the different types of applications allow us to point out an important relationship between NFR specifications:

1. NFRs of platform applications are dependent on NFRs of the platform, since platform applications make use of platform services.

2. When developing NFRs for the platform, a key success factor is that important NFRs of domain applications should be preserved as much as possible when a selected domain application functionality is migrated to make use of platform services.
Developing platform NFRs in such a way that domain applications migrated to the platform retain their important NFR properties will always require certain tradeoffs, since platforms, and in particular service-oriented platforms, have distinct architectural features. The best that can usually be achieved is to retain some of the important NFRs by sacrificing less important ones. Such tradeoffs could, for example, involve the development of more than one platform, each resulting from a different type of NFR tradeoff, and each offering different subsets of platform services to the platform applications. Figure 8-3 captures such an approach by defining an association link between Platform NFR Specification and the services provided by the platform.

8.2.4. Analyzing Processing Throughput and Processing Capacity

A control system usually has many types of throughput requirements specified. A key NFR for industrial control systems is to achieve sufficient processing throughput of input data so that all hard real-time performance requirements are fully met. Processing throughput is a quantitative requirement that specifies the volume of data a deployed control system must process within a unit of time under a specified loading condition. For example, a throughput requirement could be specified as the processing of 3000 Change of Input Values (COVs) per second during normal system operation or 50,000 COVs during peak load conditions. An additional throughput requirement could be alarms per second to process during normal or during maximum-load conditions.

“Processing capacity” was defined as the number of instructions a deployed system can process per second. Processing capacity is directly related to the system’s hardware capabilities. The more hardware or the more powerful the hardware, the higher the processing capacity. Intuitively, the more processing throughput required, the more processing capacity is needed. This intuition leads us to the key analysis questions underlying the development of platform NFRs:

- What factors impact total processing throughput of a deployed system?
- How much processing capacity is needed to achieve the required processing throughput requirements of different deployed systems, given the factors that impact each system’s processing throughput?

To systematically address these questions, GS-PND offers a both a goal-oriented analysis technique and a scenario-oriented one.

Goal-oriented analysis technique: This helps analyze the throughput NFRs and qualitative trade-offs
in order to address other important qualitative NFRs. An NFR goal graph is used to help clarify the operational meaning of the qualitative NFRs, and to justify design choices and tradeoffs (Chung, 1993; Chung et al., 2000).

**Scenario-oriented analysis technique:** This technique helps analyze the processing capacity needed to meet different load conditions of deployed applications. Scenarios are utilized to capture and analyze processing capacity needs and processing overheads when throughput NFRs are traded off to address other NFRs. Scenarios are captured using the UCM notation (Amyot & Mussbacher, 2003), which supports representing and analyzing high-level system processes and structures.

### 8.3. Goal-oriented Analysis Technique

Figure 8-4 illustrates an NFR goal graph that captures a goal-oriented analysis of processing throughput NFRs. Although processing throughput can be measured quantitatively (e.g., number of inputs processed per second), during goal-oriented analysis, the factors that affect processing throughput are analyzed qualitatively.

Figure 8-4 indicates that the first step of goal analysis to identify factors affecting throughput is to first state the goal “Maximize Throughput of Input data processing”. In the modeler’s opinion, this goal is sufficiently addressed by three subgoals: Maximize speed of software control process steps, Minimize Latency of System, and Minimize Failure rate of System. The “and” contribution link specifies that we expect those three subgoals, once addressed, to sufficiently address their parent goal. No other sibling goal needs to be identified.
Each subgoal is in turn analyzed to identify additional subgoals, until one identifies particular requirements or design techniques that address the lowest level goals. For example the Maximize speed of software control process steps NFR goal is further decomposed into Maximize speed of instruction processing, Maximize number of real (i.e. not virtual) parallel processing, Minimize number of bytes per process step processed, Minimize number of invocations of subroutines, and Minimize number of process steps interruptions.

The “Help” contribution link between the subgoals and the parent goal indicates that each subgoal, to some extent, contributes independently (“helps”) to achieving the parent goal, although other contributing sibling goals can further be identified. The child softgoals are then decomposed into known design techniques, captured as operationalizing softgoals: Use powerful processor helps Maximize the speed of instruction processing, and Use several CPUs or Systems helps Maximize number of real parallel processing. In the goal models, design approaches under consideration are expressed as operationalizations.

Figure 8-4 further illustrates that introducing Support Multitasking, which is brought in to help Minimize device access delays (which in turn contributes Minimize Latency of System), has a negative side effect on Minimize process step interruptions, as is indicated by the dotted “Hurt” correlation link. Hence, introducing multitasking is a tradeoff between latency and speed of processing.
of the control software. Other tradeoffs, not shown in figure 8-4 to avoid cluttering the illustration, can also be identified. For example, the use of several CPUs will “Hurt” Minimize total cost of ownership. Finally, figure 8-4 shows that Simple processing (keeping processing algorithms and data structures simple) helps Minimize number of Bytes per process step processed, as well as Minimize number of invocations of subroutines. In other words, simplifying data and system structures and processes will help reduce computational overhead.

Figure 8-5 expands the NFR goal graph to include additional NFR goals such as Usability, and Cost of Ownership, as well as Interoperability, a key design goal that motivates the use of service-orientation, which are analyzed alongside Processing Throughput. Figure 8-5 schematically illustrates how some of the design techniques (captured as operationalizing softgoals (indicated by the bold softgoals enclosed by rectangles – the rectangles are not part of the notation) are factors that negatively impact the MaximizeThroughputOfInputDataProcessing NFR. The Column chart at the bottom of figure 8-5 (which is not part of the NFR goal graph notation) shows that with the selection of different softgoal operationalizations (informally indicated by the arrows from operationalizing softgoals to columns in the column chart), processing overhead is incurred on top of a baseline level of processing needs. For example, the inclusion of a high level object model (an operationalizing softgoal) in addition to a basic control data model into the control system, helps improve usability (since it supports defining control data at an abstraction level closer to the system’s users); however, it also incurs a processing overhead. Similarly, the use web services, which helps achieve Interoperability incurs additional processing overhead. In any event, the NFR goal graph in figure 8-5 does help answer the first question: What factors impact processing throughput of a deployed system?

---

25 Figure 8-5 is a schematic overview to indicate the relationship between operationalizing softgoals in goal graphs and processing overhead. The goal graph is not intended to be read in detail, hence the larger goal graph is included although the size of the goal graph renders the labels in the softgoal not legible.
One aim of a goal graph is to assist in capturing domain knowledge. The validity of the goal graph must continuously be reviewed to identify whether additional information must be added to make the requirements and design information and knowledge more accurate and complete. Furthermore, capturing requirements and design knowledge with goal graphs also supports developing knowledge-based approaches, where existing goal graph templates can be used to validate and further complete other goal graphs under development (Chung, 1993; Chung et al., 2000; Mylopoulos et al., 1992; E. Yu & Mylopoulos, 1996).

8.4. Scenario-oriented Analysis Technique

To answer the second question—How much processing capacity is needed to achieve required processing throughput requirements of different deployed systems, given the factors that impact each system’s processing throughput?—we turn to a scenario-oriented analysis technique.

Goal modeling and scenario modeling complement each other. Goal modeling assists in identifying a selection of operationalizations that help address important qualitative NFRs, while making acceptable qualitative NFR tradeoffs. Scenario-oriented analysis assists in estimating capacity requirements for each.
of the related operationalizations, thus allowing a quantitative estimate of the total cost of ownership. If
the total cost of ownership is found to exceed acceptable limits, one returns to the goal models to seek
different quality NFR tradeoffs that have reduced overhead footprints.

During scenario-oriented analysis, an abstract processing model of an industrial control system is
constructed (see figure 8-6) which is used to estimate processing capacity needs. The model extends the
UCM notation (Buhr & Casselman, 1996) to capture a high-level description of control processes in
terms of processing steps and high-level structural elements.

Figure 8-6 captures baseline processing steps for two key control processes: **Change of Value**
(COV), and **Alarm**. **COV** is invoked whenever a change is detected in an input data point. The changed
value is then read from the input sensor and sent to the system’s subscribing clients. **Alarm** is invoked
by **Change of Value** to check if the changed value exceeds a threshold. If the threshold is exceeded,
alarm data is sent to the subscribing clients. The model also captures key system structures such as
**Server**, and **Client**, as well as **Communication System**. Scenario analysis starts with determining
baseline processing needs: where no overhead structures and processes are included (figure 8-6). Thus,
the process model in figure 8-6 corresponds to selecting the **Simple Processing** operationalization.

Each control process is further associated with key supporting data structures. For example, to check
if a changed value has exceeded a threshold, a lookup data structure is used that maps between input
data points and alarm entities that store respective threshold values for input data points. Figure 8-6
also shows two “looping” data structures that support the **COV** and **Alarm** processes in sending
messages to all subscribed clients. These data structures capture COV and Alarm subscribers.

Finally, the control process model is annotated with relevant quantitative variables derived from
deployment parameters and loading-condition NFRs. For example, a start point is associated the
number of Change of Values per second (a load condition), that the control system is expected to
process and a data structure is associated the number of configurable alarms (a deployment parameter).
These latter parameter determines the number of mappings required (from data points to alarms),
which in turn also determines the processing needed to look up one mapping – which is usually the base
2 logarithm of the size of the mapping table.
Once a processing model is established and relevant quantitative variables are linked to the control processes, a mathematical function is derived for calculating processing capacity needs. This function is then used to estimate the processing capacity needs of the different deployment configurations of existing domain applications.

Deriving the processing capacity function involves three steps:

1. Capture processing capacity needs for each processing step.
2. Capture processing capacity needs for single invocations of each control process.
3. Capture processing capacity needs for multiple invocations of each control process per unit of time.

Table 8-1 summarizes the first analysis step for the processing model in figure 8-6. Each process step is analyzed in terms of processing capacity needs, which are identified as follows:

- A processing step that has no associated quantitative variable requires a constant factor of high level instructions.
- A processing step that has a loop mapping table requires a constant factor times the maximum possible number of times the loop can be performed.
- A processing step that has a lookup mapping table requires a constant factor times the base two
log of the size of the lookup table to perform one lookup.

Note that processing steps that are shared amongst two control processes are scored separately for each control process because they execute separately.

Once the processing capacity of each processing step is identified, an aggregate formula for each control process can be derived. According to figure 8-6 and table 8-1 the COV control process includes processing steps P1, P2, P3 and P6, while the Alarm control process includes steps P4, P5 and P7.

Finally, the associated maximum load condition to capture processing capacity for multiple invocations of each control process per unit of time is identified. For example, the combined requirement value of maximum COV values per second and maximum number of alarms per second that an individual industrial controller must process would be calculated; then the total processing capacity for the COV process is calculated by the sum total of the individual processing steps for one control process invocation, times the number of times the control process is invoked per second. This yields the following formula: \((P1+P2+P3+P6) \times \text{COV}\), for the total capacity requirement of the Change of Value.
process, and \((P4 + P5 + P7) \times \text{Alarm}\), for the Alarm process.

By substituting the capacity variables (P1, P2, etc.) with the processing capacity formula that they designate, the total capacity formula (according to the processing model in figure 8-6) is derived as follows (tagged factors are sums of individual factors, such as \(F1' = F1+F2\))

Equation 1:

\[
Total\ Processing\ Capacity\ Need(\text{ActiveClient, ConfigurableAlarm, COV, ALARM}) = (F1' + F2' \times \text{ActiveClients}) \times COV
+ (F2' + \log_2\text{ConfigurableAlarms} + F2' \times \text{ActiveClients}) \times \text{ALARM}
\]

The above equation captures the total processing capacity needed as a function of deployment parameters (Active Clients and Configurable Alarms) and loading conditions (Change of Values per second and Alarms per second). Note that since the scenario model is a baseline model, the function constructed from the scenario model specifies baseline-processing needs without any processing overhead.

**Table 8-1: Processing capacity formula elements for single invocations of processing steps**

<table>
<thead>
<tr>
<th>Capacity Variable</th>
<th>Process Step</th>
<th>Processing Capacity Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Get data point value</td>
<td>(F1) a constant</td>
</tr>
<tr>
<td>P2</td>
<td>Create Tag record</td>
<td>(F2) a constant</td>
</tr>
<tr>
<td>P3</td>
<td>Notify Tag value changed</td>
<td>(F3 \times \text{ActiveClients})</td>
</tr>
<tr>
<td>P4</td>
<td>Check for Alarm threshold</td>
<td>(F4 \times \log_2) (ConfigurableAlarms)</td>
</tr>
<tr>
<td>P5</td>
<td>Notify Alarm for Data Point</td>
<td>(F5 \times \text{ActiveClients})</td>
</tr>
<tr>
<td>P6</td>
<td>Send Tag</td>
<td>(F6 \times \text{ActiveClients})</td>
</tr>
<tr>
<td>P7</td>
<td>Send Alarm</td>
<td>(F7 \times \text{ActiveClients})</td>
</tr>
</tbody>
</table>

8.5. Analyzing Service Orientation Processing Overhead

Figure 8-7 illustrates reasoning structures related to interoperability, a key service-oriented NFR. **Interoperability of the System** is decomposed into the subgoals of **Interoperability of Device Access** and **Interoperability of Messaging Access**, indicating that to achieve interoperability of the control system, interoperability for those processing steps that access external devices and those that access the messaging systems need to be addressed. To this end, these interoperability subgoals are further
decomposed into the subgoals of **Interoperability of GetDataPoint**, **Interoperability of Notify Data Point Change** and **Interoperability of NotifyAlarms**, which are ultimately further decomposed into one or more **UseWebServices** operationalizations to help achieve the respective interoperability NFRs.

Figure 8-7 further shows that the web service operationalization introduces both positive and negative effects on other quality NFRs. For example, the use of web services “helps” achieve **Standardization** and **Extensibility**, but “hurts” **Maximize Throughput** and **Reduce Cost of Ownership**. Choosing to adopt web services thus adds overhead to the baseline.

![Figure 8-7: Processing model annotated with web services indicators](image)

Once the web services operationalizations have been selected, it is possible to annotate the processing steps that should be web-service-enabled. Figure 8-8 shows the annotated process model. Annotations are shown below processing tasks as two parallel lines with “WS” initials. The process model now shows that the **Get Data Point Value**, the **Notify Tag Change Value**, as well as the **Notify Alarm for Tag** process steps utilize a web service based approach in its design. These annotations are used when determining processing capacity requirements for each processing step.

Figure 8-9 shows how web service annotations may be expanded into a processing model fragment that helps estimate processing capacity requirement for process steps implemented as web services. More specifically, each web service annotation specifies the execution of several additional processing steps, including encoding data as web service messages, sending web service messages, and decoding
web service messages to extract included data. From a capacity needs point of view of the control server, we find that each web-service-enabled processing step adds a constant factor ($F_{ws}$) of processing steps. Table 8-2 captures how the additional processing overhead is included in the processing capacity formulas when web-service-enabled processing steps are introduced.

It is noteworthy that the UCM notation supports a notion of UCM plug-ins, where a single responsibility can be expanded into another UCM. The expansion of an annotation is a generalization of the plug-in concept, which is useful when detailed processes and structures do not encapsulate well within single UCMs. Figure 8-9 illustrates an example where expansions occur both on the Client and on the Server.

Table 8-2: Processing capacity formulas for single invocations of processing steps with web service overhead

<table>
<thead>
<tr>
<th>Capacity Variable</th>
<th>Process Step</th>
<th>Processing Capacity Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Get data point value</td>
<td>$F_1 + F_{ws}$</td>
</tr>
<tr>
<td>P2</td>
<td>Create Tag record</td>
<td>$F_2$</td>
</tr>
<tr>
<td>P3</td>
<td>For each COV-subscribed Client notify Tag value changed</td>
<td>$(F_3 + F_{ws}) \times \text{ActiveClients}$</td>
</tr>
<tr>
<td>P4</td>
<td>Check for Alarm threshold</td>
<td>$F_4 \times \log_2 (\text{ConfigurableAlarms})$</td>
</tr>
<tr>
<td>P5</td>
<td>For each alarm-subscribed Client notify Alarm for Tag</td>
<td>$(F_5 + F_{ws}) \times \text{ActiveClients}$</td>
</tr>
<tr>
<td>P6</td>
<td>Send Tag</td>
<td>$F_6 \times \text{ActiveClients}$</td>
</tr>
<tr>
<td>P7</td>
<td>Send Alarm</td>
<td>$F_7 \times \text{ActiveClients}$</td>
</tr>
</tbody>
</table>

Based on table 8-2 we the Total Processing Capacity Needs equation is modified as follows:

Equation 2:

$$\text{Total Processing Capacity Need}_{WS}(\text{ActiveClient}, \text{ConfigurableAlarm}, \text{COV}, \text{ALARM}) = (F'_1 + F_{ws} + (F'_2 + F_{ws}) \times \text{ActiveClients}) \times \text{COV} + (F'_3 + \log_2 \text{ConfigurableAlarms} + (F'_4 + F_{ws}) \times \text{ActiveClients}) \times \text{ALARM}$$

To use the above equation, the constants included in the equation must be set to some specific numbers. For the purpose of our analysis, the number of high-level instructions that each step would require is estimated. Since different deployment configurations are to be compared, the equation may
be further simplified by just estimating the relative weight of each step to each other, as follows: Each processing step is weighted as “1”. If, however, a processing step is decomposed, then its weight is the total weight of its components. Based on that formula, we find that all constants are weighted with “1” while $F_{\text{WS}}$ is weighted with “5”, since it is expanded into 5 additional processing steps. Once numbers are substituted for variables the following formula is obtained:

Equation 3:

$$Total\ Processing\ Capacity\ Needs_{\text{WS}}(\text{Active\ Clients},\text{Configurable\ Alarm},\text{COV},\text{ALARM}) = (6 + 6 \times \text{Active\ Clients}) \times \text{COV} + (1 + \log_2 \text{Configurable\ Alarm} + 6 \times \text{Active\ Clients}) \times \text{ALARM}$$

This modified “Total Processing Capacity Needs” formula now captures the increased processing capacity that a control server requires, and, taken together with the NFR goal graph model in figure 8-7 and the processing model in figure 8-8, helps identify for what purpose and where within industrial control processes that the web service is introduced, and the kind of processing overhead that the introduction of web services would involve.

To summarize: The goal model supports identifying the design choices for achieving qualitative NFRs and helps to point out qualitative NFRs tradeoffs. As illustrated in this section, the introduction of operationalizations in turn introduce additional design elements in scenario models, thereby adding processing overhead. Stated differently, introduced processing overhead is justified by the achievement of stated qualitative NFRs. Analysts must therefore carefully select which of the stated NFRs to address, while trading off increased processing overhead and increased costs. While the goal-and scenario-oriented techniques do not support optimizing requirements and design choices, they do support identifying requirements and design choices that are adequate for the purposes specified. It is noteworthy that goal modeling and scenario modeling support the systematic modeling and analysis of processing overhead incurred for the selection of any kinds of quality NFRs, not only those relevant to service orientation.
8.6. Deriving Platform NFRs from NFRs of Existing Control Systems

For every distinct problem domain, there is specified a control application, each with three defined deployment configurations: small, medium and large. Each deployment configuration is characterized by distinct deployment parameters: the number of (active) clients that a particular server running the control application can concurrently handle, and the number of alarms that can be configured per server. Each deployment configuration is also associated with different loading conditions: the number of COVs per second and the number of alarms per second a server can process. The larger the deployment configuration, the more active clients and configurable alarms are supported. The analysis questions are as follows:

- What deployment configurations should be defined for a shared application platform?
- What deployment parameters and what loading conditions should each platform deployment configuration support?
- What effect does service-orientation have on defining deployment configurations for the shared application platform?

These questions can be answered using the mathematical functions derived earlier, and from the
particular input data (deployment parameters and loading conditions) provided for each deployment configuration of the existing control applications. Table 8-3 captures deployment configurations (top two tables) and loading conditions (bottom two tables) of three different control applications, for Automation Technology (AT), Building Technology (BT) and Transportation Technology (TT) domains. Each application has a small, medium and large deployment configuration defined. Table 8-3 shows the deployment parameters “Number of Active Clients [attached] per Server”, and “Number of Configurable Alarms per Server” that characterize the size of a deployment configuration, as well as the required loading conditions “Max possible COV/sec per Server” and “Max possible Alarms/sec per Server” for each deployment configuration.

<table>
<thead>
<tr>
<th></th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>1</td>
<td>8</td>
<td>50</td>
</tr>
<tr>
<td>BT</td>
<td>8</td>
<td>30</td>
<td>150</td>
</tr>
<tr>
<td>TT</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>12500</td>
<td>35000</td>
<td>91000</td>
</tr>
<tr>
<td>BT</td>
<td>50</td>
<td>250</td>
<td>340</td>
</tr>
<tr>
<td>TT</td>
<td>1450</td>
<td>3200</td>
<td>4800</td>
</tr>
</tbody>
</table>

Table 8-3: Deployment and loading conditions

Using the Total Processing Capacity Needs for Web Services formula (equation 3) we now calculate actual processing needs, including the processing overhead incurred by utilizing web services. Table 8-4 captures the total capacity needs when applying this formula for each deployment configuration of each domain application and is sorted according to capacity needs. The Power Increase column captures the ratio between two adjacent capacity needs, such as 33446 instructions per second, for BT/Small, divided by 3955 Instructions per second, for TT/Small, which equals to 8.46. The Power Increase column shows the amount of extra processing power that the hardware must offer in order to provide sufficient processing capacity.

The processing needs in table 8-4 help identify the cost that each deployment configuration type
incurs when supported by the platform. For example, we could define a small, cost effective deployment configuration which supports domain applications up to the processing capacity of BT/Small; or a medium deployment configuration that would support a processing capacity up to AT/Medium; a large one that can support a processing capacity up to AT/Medium; or, finally, a very large deployment configuration that supports processing capacities of up to AT/Large.

Table 8-4: Total processing capacity needs -- ranking

<table>
<thead>
<tr>
<th>Application</th>
<th>Deployment Configuration</th>
<th>Instructions Per Second</th>
<th>Power Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>Small</td>
<td>3955</td>
<td></td>
</tr>
<tr>
<td>TT</td>
<td>Small</td>
<td>33446</td>
<td>8.46</td>
</tr>
<tr>
<td>BT</td>
<td>Medium</td>
<td>58741</td>
<td>1.76</td>
</tr>
<tr>
<td>TT</td>
<td>Medium</td>
<td>99490</td>
<td>1.69</td>
</tr>
<tr>
<td>AT</td>
<td>Small</td>
<td>175147</td>
<td>1.76</td>
</tr>
<tr>
<td>TT</td>
<td>Large</td>
<td>254438</td>
<td>1.45</td>
</tr>
<tr>
<td>BT</td>
<td>Large</td>
<td>407371</td>
<td>1.60</td>
</tr>
<tr>
<td>AT</td>
<td>Medium</td>
<td>2205711</td>
<td>5.41</td>
</tr>
<tr>
<td>AT</td>
<td>Large</td>
<td>33776007</td>
<td>15.31</td>
</tr>
</tbody>
</table>

Table 8-5 compares the processing needs of service oriented and non-service oriented platforms. According to table 8-5, the use of web services increases processing capacity needs by a factor of 5 to 7 across all deployment configurations. Since processing capacity needs are proportional to cost of ownership, requirements analysis can thus predict that a lowest-end deployment configuration without service-orientation (such as BT/Small) costs about 6 times less than the same deployment configuration with service orientation. Cost of ownership considerations may suggest offering two low-cost deployment configurations: a) a non-web-service enabled configuration that offers a particularly small processing footprint with minimal capacity needs, but which, however, trades off interoperability and standardization; and b) a web-service enabled one, which does offers the benefits of service-orientation, however at a higher cost of ownership.

While the above analysis focused on qualitative NFRs related to the adoption of service orientation, other NFRs, such as modularity, extensibility and the like, can be analyzed in a similar manner. Requirements and/or design techniques addressing these NFRs can be identified and captured using goal graphs, and scenario models can be developed to identify and calculate relevant overhead.
Table 8-5: Comparing WS- and non-WS-enabled capacity needs

<table>
<thead>
<tr>
<th>Application</th>
<th>Deployment Configuration</th>
<th>With WS</th>
<th>No WS</th>
<th>Power Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>Small</td>
<td>3955</td>
<td>679</td>
<td>5.82</td>
</tr>
<tr>
<td>TT</td>
<td>Small</td>
<td>33446</td>
<td>5906</td>
<td>5.66</td>
</tr>
<tr>
<td>BT</td>
<td>Medium</td>
<td>58741</td>
<td>8641</td>
<td>6.80</td>
</tr>
<tr>
<td>TT</td>
<td>Medium</td>
<td>99490</td>
<td>17290</td>
<td>5.75</td>
</tr>
<tr>
<td>AT</td>
<td>Small</td>
<td>175147</td>
<td>25105</td>
<td>6.98</td>
</tr>
<tr>
<td>TT</td>
<td>Large</td>
<td>254438</td>
<td>40238</td>
<td>6.32</td>
</tr>
<tr>
<td>BT</td>
<td>Large</td>
<td>407371</td>
<td>58831</td>
<td>6.92</td>
</tr>
<tr>
<td>AT</td>
<td>Medium</td>
<td>220571</td>
<td>315231</td>
<td>7.00</td>
</tr>
<tr>
<td>AT</td>
<td>Large</td>
<td>33776007</td>
<td>4880007</td>
<td>6.92</td>
</tr>
</tbody>
</table>

Although not illustrated in this chapter, selected sets of operationalizations and related scenario models can be clustered together to support dealing with groups of related requirements and design decisions (ITU-T, 2008).

**Practitioner’s feedback:** the presented modeling and analysis technique was presented to practitioners involved in the development of the application platform. Practitioners provided positive response to the modeling and analysis technique, and found the quantitative analysis related to the utilization of web services in particular helpful. The analysis technique support types of analyses that went beyond an ad-hoc approach that was utilized during the project.

**Prototype implementation:** as part of the collaborative research project with Siemens, the author of this thesis developed a Microsoft Excel workbook plug-in prototype that demonstrated the analysis capabilities of the modeling technique. The prototype in particular focused on calculating the effect on computational capacity needs during the selections of different groups of operationalizations within an NFR goal graph, based on corresponding inclusions of template scenarios within a base line scenario model.

8.7. **Responding to Research Questions**

The focus of this case study was the development of a systematic modeling and analysis approach that supports specifying NFRs for an application platform. The platform NFRs need to be systematically
derived from predefined quantitative and qualitative NFRs of domain applications.

Since this research was about the specification of NFRs for the application platform, the data collected during this case study was drawn from requirement documents. Little architectural information was available in those documents, mainly a high-level block diagram of the platform architecture, and a collection of hardware configurations for different system deployment configurations. The data that was available for analysis was thus not sufficient to explicitly identify a link between business goals and architectural decision-making.

However, the modeling and analysis approach developed during this case study illustrated the need for considering architectural decision-making when specifying qualitative and quantitative NFRs in an integrated manner. The sample models and analyses were presented during a summary presentation of the research project to a group of about 15 researchers and practitioners at Siemens Corporate Research who were involved in the specification and development of the application platform, and positive feedback of the approach was obtained.

This was in particular interesting since it was our observation that at Siemens an explicit distinction between the requirements and the architectural design phase is made, with requirements engineers defining requirements, and architects working on addressing/achieving those requirements. The proposed approach illustrated to staff members that when dealing with NFRs, and in particular the specification of qualitative NFRs, some architectural design reasoning is needed, and that the distinction between requirements and architectural design is more fluid.

This section therefore answers the research questions in chapter 3 both in light of the documentation analyzed during the case study as well as feedback obtained from researchers and practitioners during the presentation of the GS-PND modeling and analysis approach.

**Q1.1. What role do business goals play during the architectural decision-making?**

The requirements documentation for the application platform included a section called *Business Drivers For Requirements*, enumerating several business goals that are considered relevant to the specification of NFRs. The NFRs enumerated are both qualitative and quantitative NFRs. Business drivers mentioned include: Benefit to customer (such as supports customers profitability, when using the product), Common UI, Shorten time to market, Openness, Low cost of ownership, reduce R&D cost and the like. Each of these business drivers is then related to NFRs including: Availability, Maintainability, Reliability, Configurability, interoperability, Usability, Security, Safety, Reusability, Localizability,
Performance, Scalability, and Extensibility.

The document reveals that Business goals are used to justify NFRs for the platform and for products derived from the platform. The document does not further indicate the role those business drivers play during architectural decision-making.

**Q2.1. What modeling constructs are appropriate for making business goals and the role they play during architectural decision-making explicit?**

Business goals were mainly represented as softgoals, such as the cost of ownership NFR in figure 8-7. A goal graph was utilized to link business goals to NFRs, and NFRs to architectural decision-making. However, the linkages to architectural decision-making were extrapolated from the research data and were not explicitly mentioned.

**Q1.2. What role do qualitative NFRs play during architectural decision-making to meet business goals?**

Qualitative NFRs such as interoperability and scalability played a role in driving and justifying the architectural design of the application platform. Indeed the requirements documentation presented a larger list of non-functional requirement, some of which were not quantified into scalars or ranges of numbers. However, while the requirement document indicated how the specification of NFRs help address business goals and deriviers, the requirement documentation did not reveal how these non-formalized qualitative NFRs informed architectural decision-making. This was considered out of the scope of the requirement document.

**Q2.2. What modeling constructs are appropriate for making qualitative NFRs and the role they play during the architectural decision-making to meet business goals explicit?**

Since the role of qualitative NFRs played during architectural design was not explicitly mentioned in the requirements documentation, no specific construct recommendation could be derived. However, the objective of the case study was to explore how a qualitative treatment of NFRs could be integrated with the definition of quantitative NFRs for the application platform.

This case study thus illustrated how qualitative NFRs can help in identifying and selecting
architectural design approaches, which in turn include different additional architectural processes and structures into scenario models. NFRs included in a goal graph thus guide in selecting different overhead processes and structures in scenario models, which in turn affects the cost of achieving quantitative NFRs.

Q1.3. What role does the organizational setting play during architectural decision-making to meet business goals? Q2.3. What modeling constructs are appropriate for making the organizational setting, and the role the organizational setting plays during architectural decision-making to meet business goals explicit?

During this case study no data regarding the organization was available. An organizational analysis was thus not performed.

Q1.4. What role do quantitative NFRs play during architectural decision-making to meet business goals?

The requirements documentation did not explicitly indicate the link between quantitative NFRs and architectural decision-making to meet business goals. However, at the requirements level quantitative NFRs helped characterized the deployment size and the maximal processing capacity needs of the deployed system. These parameters directly related to the target market of a system. For example, qualitative NFRs in the lower load and processing throughput range, required less-powerful hardware, such as embedded hardware components, thereby enable the offering of a low cost system for personal home use. Quantitative NFRs were thus closely linked to the kind of customer type and market segments. During architectural decision-making, the tradeoff between non-functional properties of the system, and processing capabilities geared to different market segments needed to be taken into account.

At the requirements level quantitative NFRs played another role. They were collected from different domain applications, such as the “TT”, and the “AT” domain application, to specify the quantitative NFRs the application platform needs to support, once the domain applications are migrated to the platform.

Q2.4. What modeling constructs are appropriate for making quantitative NFRs and the role they play during the architectural decision-making to meet business
goals explicit?

During this case study, to support specifying quantitative NFRs, a measurement meta-model was developed based on the work by Fowler (Fowler, 1997). The measurement meta-model supported specifying and translating between different terminology and metrics used for quantitative NFRs across different domain applications and deployment configurations. The measurement meta-model supported in standardizing terminology and metrics, an essential step to support consolidating quantitative NFRs collected from individual domain applications towards quantitative NFRs of the platform.

Scenario modeling supported determining the cost of specified quantitative NFRs. During this case study, instead of calculating actual cost (such as dollar amounts), cost was expressed in terms of atomic instructions per second a deployed system hardware/software needed to support, so that all specified quantitative NFRs were achieved.

Specifying different sets of quantitative NFRs each associated with a different deployment configuration thus supported calculating the overall cost of each deployed solution, and thus evaluate the feasibility of the quantitative NFRs for a particular market segment.

As explained earlier, these calculations were also “parameterized” by the selection of design approaches that addressed qualitative NFRs, which in turn introduced processing overheads – and thus increased the cost across deployment configurations.

8.8. Modeling Discussion

Related work: Other work has dealt with the integration of qualitative and quantitative NFRs. For example, Nixon examined performance NFR in a qualitative manner during information system design (Nixon, 1994). Nixon, however, did not deal with throughput requirements or with the integration of qualitative analysis and quantitative analysis of performance.

Cai and Yu (2002) outlined an approach for dealing with the performance NFR in conjunction with Use Case Maps (UCMs) (Buhr & Casselman, 1996). The modeling technique presented in this chapter is similar Cai and Yu’s work, in that both utilize the UCM notation in conjunction with NFR goal modeling. However, Cai and Yu focus on describing the results of architecture decision-making using Scenario modeling, while the method presented in this chapter integrates both modeling approaches to support an integrated qualitative and quantitative analysis of throughput requirements during the development of application platforms.

The work most closely related to GS-PND is by Pourshahid et al. (2007). Pourshahid et al. extend the
User Requirements Notation (URN) (ITU-T, 2008), a recent requirements analysis and specification standard that supports representing, capturing and analyzing qualitative NFRs, and then linking them to UCM scenario modeling. While Pourshahid et al. apply and extend URN to represent and evaluate business processes in organizations, the GS-PND can be seen as extending URN to specifically deal with the development of NFRs for a control systems’ application platform.

There has been much interest in recent years in dealing with NFRs in relation to service orientation [e.g., (NFPSLA-SOC'07) and (MNPSC'08)]. Most of the studies, however, relate to quantitative methods, models and techniques to support the specification, discovery, matching and selection of services, to best meet client’s quantifiable non-functional needs. Little work deals with Qualitative NFRs or with the integrated treatment of both Qualitative and Quantitative NFRs during the design of service-oriented software systems in general, or software system platform, in particular.

One notable exception is Liu et al. (Y. Liu, Zhu, Bass, Gorton, & Staples, 2008), which offers a method for modeling and evaluating the performance overhead when service-oriented business processes are deployed across different organizations. Liu et al. offers a quantitative performance modeling and analysis approach to evaluate how different deployment topologies affect process performance. However, no treatment of qualitative NFRs is provided. Furthermore, Liu’s et al. approach is applied in the business process domain and does not deal with the development of NFRs for an application platform.

Finally, the relationship between non-functional requirements and service orientation is informally explored by Erl (2007). While Erl provides important insights into the non-functional driving forces behind the adoption of service orientation, no systematic modeling and analysis approach is provided. Erl’s work is however a useful source for identifying relevant service-oriented NFRs during qualitative modeling and analysis.

8.9. Conclusion

Developing non-functional requirements for a service-oriented platform that supports industrial control systems in different domains requires dealing with various requirements challenges. Requirements analysts must deal with varying needs among the different domain applications both in terms of hard real time constraints and in terms of product viability in the market place.

This chapter presented an NFR platform development method that systematically supports dealing with such diverse requirements and offers analytical support in arriving at viable product NFR specifications. This approach was used to analyze the NFRs that were elicited from three control
systems, each designed for a different application domain. The approach supported developing platform NFRs. Positive feedback for the modeling and analysis approach was received from researchers and practitioners involved in the development of the application platform.

Dealing with throughput as a key qualitative and quantitative NFR that is traded off with other NFRs appears specific to industrial system’s real time domain. Also, the manner how deployment parameters and load conditions interrelate when determining throughput capacity as illustrated by scenario models also appear specific to the problem domain at hand. However, the goal-oriented modeling and analysis technique developed for dealing with these requirements is not specific to dealing with throughput. Other qualitative NFRs can be captured, developed, and different kinds of links between the goal models and process models can be represented and analyzed.

The method, derived from a real-life industrial problem, reflects complexities of requirements and architectural design problems found in actual practice. The method was reviewed by stakeholders in the platform development domain, who found the GS-PND method, in principle, appropriate to the problems at hand.

Future work can focus on developing and evaluating the more detailed capacity analyses and variability analysis further. Future work can also focus on extending the performance modeling approach and analysis approach in directions of SOA Governance similar to Liu et al. (Y. Liu et al., 2008) as well as dealing with performance measures that include probabilistic system behavior, such as probabilistic cache hit behavior, and maximum sustainable throughput in percentage of time. Additional work can focus on tool support for collecting qualitative and quantitative NFRs from product stakeholders, as well as on standardization of NFR terminology across different domain application.

Another area of future research is to extend GS-PND towards actor-oriented modeling and analysis (E. Yu, 1994, 2001b), to assist reasoning about functional and non-functional requirements and design decision-making that is performed by different development organization stakeholders (Daniel Gross & Yu, 2001a). Including actor-oriented analysis support into the GS-PND method appears particularly useful, since the success of the application platform’s design will depend on how well design decisions included in the platform facilitates the achievement of functional and non-functional design goals of the different business units responsible for developing platform based product offerings.
9. IAL Applied to Dealing with Terminological Ambiguity during Architectural Design

9.1. Introduction

For software architectural designers to effectively collaborate, it is fundamental that they share an understanding of the software design artifacts included in the proposed design solution. Often such understanding is based on a shared vision of how artifacts contribute to the execution of software processes (the artifact’s operational meaning). However, during the design process, artifact descriptions often lack sufficient detail to unequivocally establish operational meaning, usually because they are first-cut initially, and then successively refined to include additional design details until their operational meaning can be unequivocally demonstrated. Ensuring shared meaning in larger projects is particularly difficult because of the plethora of interrelated artifacts that architects and designers must deal with, and because the design details in descriptions of different artifacts can vary greatly.

This chapter presents a semiotic meaning analysis that supports clarifying the operational meaning of artifacts during software design, and that helps in identifying whether and in what way artifact descriptions must be further elaborated. This chapter further illustrates how clarifying the operational meaning of artifacts is closely intertwined with design decision-making, and how agent- and goal-oriented modeling and analysis can be supported by a semiotic agent modeling and analysis approach to support capturing the evolving operational meaning of artifacts. While the semiotic modeling and analysis discussion presented in this chapter does not directly address the research questions presented in chapter 2, the need to address terminology issues became apparent while analyzing interview data from the case study at the insurance company (Chapter 6). Although, and perhaps, despite that, experts in the field make use of specific terminology and think that what they say is well understood by other experts, misunderstandings, as observed during case studies and as will be demonstrated in this chapter, often arise. By using semiotic agent modeling the design terminology used during design discussions and solution modeling is made much more precise.

© SciTePress Digital Library. Reprinted with permission from Daniel Gross & Eric Yu. Resolving Artifact Description Ambiguity During Software Design Using Semiotic Agent Modeling. Proceedings of the Twelfth International Conference on Informatics and Semiotics in Organizations. This chapter is largely based on (Daniel Gross & Yu, 2010a). This paper was researched and written by the first author and edited by the second.
9.2. Why a Semiotic Terminology Analysis?

Software designers often use specialized terminology and/or conceptual models to support communication and collaboration during software design [see, e.g., (Fowler & Scott, 2000; Gamma et al., 1995)]. The assumption is that these terms and models efficiently and unequivocally communicate design information amongst designers, and help prevent misunderstandings. But is this assumption correct?

During the case study at the insurance company, an SOA\textsuperscript{27} enterprise architect reported on a design discussion that occurred. The enterprise architect was responsible for developing the overall service-oriented enterprise architecture for all enterprise systems, with a number of designers responsible for the design of individual system applications and components. The enterprise architect asked a consumer component designer to utilize an asynchronous messaging approach for sending insurance policy data from a consumer to a provider component, while the consumer component designer argued for a synchronous messaging approach.

“Consumer”, “Provider” and “Messaging” are terms taken from the service-oriented architecture (SOA) design style (Erl, 2007; Josuttis, 2007), which, broadly speaking, is a distributed system design approach, whereby consumers request computational services from (usually remote) service providers through a messaging infrastructure, (Erl, 2007; Josuttis, 2007). Figure 9-1 depicts a simplified schematic illustration showing how SOA is usually applied in business enterprises, with the components shown using rectangles. “ESB” is the messaging infrastructure component, often called an “enterprise service bus”. The double sided arrows between components indicate the two-way path (sending and receiving) of the messages.

![Figure 9-1: High level service-oriented architecture in business enterprises](image)

The enterprise architect and system designers at the insurance company have a good understanding of these SOA concepts, including the terms “synchronous” and “asynchronous” messaging, before

\textsuperscript{27}SOA – Service oriented architecture
embarking on this project. Yet, despite their expert understanding, while reviewing the design discussion it became clear to the researcher that each attributed a different operational meaning to these terms.

A synopsis of the design discussion is as follows: The consumer designer prefers to request a necessary service directly from a specific provider, and to receive an immediate response. This defines a synchronous style of messaging. From the consumer designer’s point of view, this is the simplest design to accomplish the requirements. However, the enterprise architect argued that requesting a service directly from a provider harms future extensibility of the consumer component, because it limits the consumer to that specific provider to process its service requests. If, in the future, other providers need to process the consumer’s service request, changes in the consumer component would be necessary. The enterprise architect therefore advocates asynchronous messaging, which would result in a loosely coupled design, in contrast to the consumer designer’s approach (referred to as point-to-point integration) that results in tight coupling.

Reflecting on this example, the following observations can be made:

1. The terms “synchronous messaging” and “asynchronous messaging” compactly refer to the use of a number of design artifacts, which respectively implement different approaches for integrating enterprise systems.
2. The collective software behavior of these artifacts is the intended “operational” meaning of these terms during the design discussion.
3. For the designers to clearly understand the meaning of these terms, there must be agreement on the operational meaning.
4. The design discussion between the enterprise architect and the system designers occurs at a particular level of abstraction. This influences the kind and number of artifacts that are chosen during the discussion that describes operational meaning.
5. To achieve effective communication, the enterprise architect and system designer need to understand each other’s design situation, particularly, the different demands that each of them faces. These demands may well be in conflict. A design situation includes objectives, constraints, solution alternative, artifacts, evaluations, and the like.

Consider the first three observations. A specialized term, such as “synchronous messaging”, translates in the enterprise architect’s mind into a number of artifacts that collectively exhibit some software system
behavior. Usually, such a translation is not mechanically derivable, but involves interpretation and decision-making. For example, the enterprise architect explains that during synchronous messaging the consumer knows and makes use of the physical address of the provider to send a service request to the provider. However, as we will demonstrate, the term synchronous messaging allows for an alternative operational meaning, and does not necessarily involve such physical point-to-point integration. During the design discussion, the consumer designer may have this other alternative operational meaning in mind. Shared operational meaning of the terms synchronous and asynchronous messaging is thus problematic.

Considering the fourth observation: The enterprise architect considers the point-to-point integration style at a particular level of abstraction. For example, the enterprise architect may have thought of the address artifact of a second system as a physical address, such as a static IP address, that unequivocally identifies a first and a second system in a network. Alternatively, the enterprise architect may indeed have considered a logical address artifact / artifact address that allows for some routing decisions within the ESB, thereby “loosening” the operational meaning of point-to-point integration style, but still objecting to this approach. Clarifying the level of abstraction at which a design discussion takes place is thus crucial for understanding operational meaning and for clarifying design intents.

Considering the fifth observation: During a design discussion both the enterprise architect and the system designers are actively involved in the design of enterprise systems, but from different vantage points. The system designer is responsible for the design and goal achievement of a single system, while the enterprise architect is responsible for enterprise-wide goals. This also includes dealing with single systems, however from a systemic point of view, such as in relation to other systems. Each designer therefore comes to the design situation with different, sometimes conflicting, objectives in mind; different design intents may lead designers to interpret design approaches quite differently.

In this example, the design intent of the enterprise architect to avoid a point-to-point integration style relates to the additional development effort needed to support a service request from the consumer to be processed in the future by additional providers. This intent leads the enterprise architect to focus on ensuring that the provider address is not directly known to the consumer component when sending the service request message. However, suppose the consumer designers understand instead that the main concern of the enterprise architect is actually to avoid the (symbolic) inclusion of the provider service interfaces within the consumer code. (This could create a costly syntactic and semantic dependence between a provider and its consumer components, should the provider’s interface be changed in the future. Note that “interface” has a broad meaning, including
knowledge of a messaging protocol between a consumer and provider. The operational (and structural) meaning that the consumer designer then attributes to the point-to-point integration term, would focus on artifacts embodying interface knowledge within the consumer, rather than on a physical or logical provider address. Knowing each other’s design demands may thus help clarify what artifacts and behaviors are involved in the operational meaning of terms.

The core problem is the need to clarify what designers mean when they use specialized terminology during discussions or in conceptual models, while acknowledging that meaning is constructed from interpreting the operational meaning of existing artifacts in the design space (e.g., consumer, provider, etc.), and from interpreting the operational meaning of new artifacts that the designers envision included in the design space (e.g., consumers’ and providers’ messaging routines).

Considering approaches to dealing with the meaning of design artifacts, naturally leads to considering the study of semiotics in general, and the semiotic agent modeling approach developed by Stamper, Liu and colleagues for dealing with the meaning of symbols during information system requirements and design in particular (K. Liu, 2000; K. Liu, Sun, Dix, & Narasipuram, 2001; Stamper, 1973, 2006; Stamper et al., 2003).

More specifically, the semiotic meaning analysis presented in this chapter adopts Stamper’s actualism ontology that takes as its premise that the process of “knowing” depends on a “knowing (semiotic) agent,” and that the agent’s knowledge depends on the actions afforded by the agent’s perception (Gibson, 1977; Michaels & Carello, 1981; Stamper, 2006). This approach further adopts Stamper’s ontological dependence schemas which apply these philosophical notions in the form of a semantic modeling and analysis approach using semiotic agents (K. Liu, 2000; Stamper, 2006).

This thesis extends Stamper and Liu’s ontological dependence schemas to deal with software design by introducing the following features:

- Defining a new logical relationship between semiotic agents (a “modifies” link), to allow the definition of new ontological dependence schemas by extending existing ones;

---

28 The term “agent” appears in the semiotic domain (semiotic agent) as well in intentional modeling and analysis in organizations domain (intentional agent). These agents have different meanings and application, which will be discussed in this chapter, and should not be confused.
• Distinguishing between “substantive” semiotic agents, which denote humans who interpret semiotic knowledge, and “symbolic” semiotic agents, which denote computationally manipulated “symbol knowledge”;  
• Refining the notion of “affordances” (the “things” and “actions” that agents perceive and do) to distinguish between substantive (application and design domain specific) and symbolic affordances (such as software operations).  
• Linking ontological dependence schemas to agent- and goal-oriented analysis models that support capturing and reasoning about intents and decision-making in development organizations (Daniel Gross & Yu, 2001a, 2001d, 2010b; E. Yu, 1994).

9.3. Affordance and Semiotic Agent Modeling of Terms

“Affordance” is a central concept of semiotic agent modeling. Generally speaking, one “thing” (or concept) “affords” another if the first helps the second in some way. For example, an ink pen affords writing. Gibson elaborates that affordance must be understood in relation to an agent, or more precisely, to the perceptive capability of the agent (Gibson, 1977; Michaels & Carello, 1981). It is the writer who can perceive the affordance, and thus can understand the meaning of an ink pen for writing. Stated differently, the meaning of the concept “ink pen” is operational (for “writing”), and is a perceived affordance, and its perceptibility is dependent on the writer’s writing capability. In short, perceived affordance is a perceived operational meaning: A person who cannot write, nor has seen writing before, would not perceive the pen’s affordance for writing; such a person would fail to understand the ink pen’s meaning, that the pen is for writing.

An agent’s ability to perceive affordances are “preconditioned” by what the agent is, in principle, capable of sensing and doing, and by learned experiences of past encountered affordances in the agent’s living environment. In Gibson’s theory of affordance, the environment and the perceptive capacity of an agent are interrelated—the agent’s environment affords the evolution of capabilities in the agent, and the agent’s evolved capabilities afford additional perceived affordances in the environment. According to Gibson, objects that have no perceivable affordances to an agent are not perceived by the agent.

Stamper’s ontological dependence schema captures such existential (also called ontological\(^{29}\))
necessity between affordances. For instance, “person stumbling” can only exist if “person running” or
“person walking” first exists; without these affordances “stumbling” is not perceivable by “person”, and
hence does not exist for that person.

Bringing an example closer to the topic of this thesis, a semiotic agent analysis of the term “physical
point-to-point integration” attempts to identify, and capture in the form of an ontological dependence
schema, affordances that are necessary for a software system designer to perceive actions or
operations relevant to “physical point-to-point integration”, such as the physical point-to-point
transmission of data between components.

A semiotic agent thus indicates a design capability and perceptive vantage point of a designer. For
example, by naming a semiotic agent “physical point-to-point integration designer” the analysis focuses
on a designer’s capability of perceiving all affordances necessary for defining or performing any or all
actions or operations afforded by a physical point-to-point integration. In other words, the name of a
semiotic agent captures a modeling intent to explore and delineate the name’s meaning, by identifying
the indicated affordances. For instance, the adjective “physical” in the above semiotic agent’s name
focuses the meaning analysis to physical affordances only, thus limiting the scope of the analysis to
exclude any capability of perceiving a “logical point address” (and anything this concept affords), which
is left for a different semiotic agent to perceive.

Semiotic agents are therefore utilized as a structuring mechanism for software system design
terminology in terms of afforded design capabilities and actions. This parallels how design
responsibilities are identified and allocated amongst designers in development organizations.

9.4. Semiotic Agent Analysis of Point-to-Point Integration

Figure 9-2 illustrates a semiotic agent analysis of physical point-to-point integration. The semiotic agent
model is read from left to right, that is, elements more to the left “afford” elements more to the right.
Elements are linked via ontological dependence links. Captured on the far left, using a solid ellipse, is the
semiotic agent **Physical point-to-point integration designer**. This semiotic agent represents a human
designer who is capable of designing a physical point-to-point integration. The designer is therefore

necessary condition for an agent to perceiving affordance B, then B does not exist for the agent, if A is not
perceivable by the agent.

The proposed approach presented in this chapter distinguishes between “action” directly performed by the
agent and “operation” performed by a software system.
capable of a) perceiving the affordances for automated operations, as well as b) performing relevant designer actions applicable to a physical point-to-point integration.

Figure 9-2: Semiotic analysis of point-to-point integration style.

Figure 9-2 illustrates one key operation: **Transmit data from a Physical Sender Address to Physical Recipient address**, which is an automated operation. It is captured on the far right using a dashed rectangle indicating a symbolic affordance—a software operation. Since it is a symbolic operation, it refers to its dependee affordances, **Physical Device Address**, **Communication Device [Sender]**, and **Communication Device [Recipient]**, using symbols; the affordances it directly depends on are linked via symbolic ontological dependence links (dotted lines). Note that, as a result of semiotic analysis, the label of this operation is more precise than the one we used in the first sentence of this subsection ("physical point-to-point integration"), including additional terms derived from its preceding affordances.

Data and Physical connection are affordances (captured as sold rectangles) that help define the automated operation. Figure 9-2 further illustrates that **Physical connection** affords **Communication Device**. **Communication Device**, from the perspective of the semiotic agent, is perceived as a symbolic semiotic agent—an agent that performs automated symbolic operations. In other words, it is a computational device that processes software code. The symbolic nature of this agent is indicated by an ellipse with dashed lines. Having a **Communication device** affords a **Sender** and a **Receiver** role for the **Communication Device**, as well as a **Physical Device Address**.

31 [Sender] and [Recipient] are role affordances of the Communication Device.
The ontological dependence schema in figure 9-2 indicates that all these affordances are necessary for defining the operation: Transmit data from a physical Sender Address to a Physical Recipient address. Removing any affordance from the schema should make defining the operation impossible. If this is not the case, then the semiotic agent schema is incorrect and needs to be revised.

9.5. A Fuller Semiotic Agent Analysis of Discussed Terms

Along with a more in-depth analysis of terms, this section illustrates that the exact operational meaning of synchronous and asynchronous messaging is, in fact, independent of the operational meaning of point-to-point or loosely coupled integration.

Generally speaking, the architect and the consumer designer seem to have conflated “orthogonal” operational meanings when discussing synchronous and asynchronous messaging in terms of point-to-point and loosely coupled integration. It is, for instance, possible to transmit messages asynchronously and point-to-point, and it is possible to offer loose coupling and synchronous messaging. Conflating operational meaning may involve implicit, yet unintended, design decision-making. Using semiotic agent modeling ensures that such unintentional misunderstandings are avoided, by making such conflations and related decisions visible and amendable to analysis.

Figure 9-3 shows how ontological dependencies of “higher level” terms are selectively composed from ontological dependence schemas of “lower level” terms. For example, to construct the operational meaning for the higher level term “Logical loosely coupled integration”, we define and link the semiotic agent Logical loosely coupled integration designer, via a “modifies link” to the lower level Physical point-to-point integration designer agent, we defined in figure 9-2. This indicates that everything Physical point-to-point integration designer perceives is also perceived by Logical loosely coupled integration designer. The logical loosely coupled integration designer however perceives more, and can perceive affordances relevant to a logical integration, such as perceiving the notion of Logical device address. The Logical device address, together with Sender and Receiver roles (of Communication device), affords defining the operation Transmit data from logical Sender Address to logical Recipient address and Translate logical device address to one or more physical device addresses (the latter is afforded by including the Physical device address affordance also). Without

---

32 Lower level terms describe design elements that are closer to hardware and software implementation details. Higher level terms abstract away some of those details to define design elements in a less implementation specific language.
specifically restricting the meaning of the logical device address (by placing it in context of additional agents and affordances), it can be interpreted quite generally, allowing for different kinds of loose coupling.

Figure 9-3: Fuller semiotic agent analysis

Figure 9-3 further illustrates the ontological dependences for the SOA term “Consumer”. °°Consumer
designer perceives Service, Message, Provider, and Own Identity. Selections of these in turn afford perceiving Service request Message, Messaging service, Feedback message, Provider address and Own address.

Finally, with these affordances defined, the operations Send service request message from own address to Provider address via the Messaging Service and Respond to feedback message from Provider are defined. Note that to reduce link clutter in the diagram, we capture some ontological dependence links by a simple identification and referencing scheme. A bracketed identifier, such as (e1), appended to a label uniquely identifies a model’s elements, while a bracketed identifier prefixed to a label indicates an ontological dependence link on the element referenced. For example, the affordance Logical Device address (e2) is uniquely identified by the identifier “e2”. The affordance (e2, e4) Logical Provider address indicates that the logical provider address is ontologically dependent on Logical Device address (e2), and Provider address (e4). Including a “symbol” in a prefixed bracketed identifier indicates a symbolic ontological dependence link.

After defining the semiotic agents and respective ontological dependence schema for the terms Physical point-to-point integration, Logical loosely coupled integration, and Consumer, we construct higher level semiotic agents. The physical Point-to-Point consumer designer is a modification of Consumer designer, and defines the terms Physical Provider address, Physical own address and the operation Send service request message from own physical address to Provider physical address via the Messaging service. We similarly define the Logical loosely coupled Consumer designer agent.

Stepping up another level, we modify these agents to define two synchronous and two asynchronous messaging related semiotic agents: Synchronous Physical Point-to-Point Consumer designer, and Asynchronous Physical Point-to-Point Consumer designer; Synchronous Logical loosely coupled Consumer designer and finally, Asynchronous logical loosely coupled Consumer designer. The ontological dependence schema for synchronous messaging indicates that, with respect to the consumer designer, synchronous messaging is defined by a designer action (captured by a solid rectangle) embodied in the manner software code is strung together so that when the software code is executed, the consumer waits for a feedback message from the provider, and then immediately handles the feedback message.

A careful analysis of the ontological dependences of these designer actions indicates that they are afforded by any consumer agent (and selected consumer’s affordances), including all agents that modify the consumer agent [the modification link acts in this respect like an ISA relationship in object-oriented analysis (Fowler & Scott, 2000)]. Synchronousity is thus applicable to physical point-to-point consumer
designers and to logical loosely coupled consumer designers. This is also the case for asynchronously.

To establish this, we trace back the affordances of the designer action and note the semiotic agents whose affordances are required. We also perform a “factoring” analysis (a type of analysis made possible by the “modifies” relationship we introduced) to identify lower-level affordances that can be “factored out”, leaving only higher-level affordances in the ontological dependence chain.

For example, tracing back from the designer action Wait for Feedback message from provider of the Synchronous Physical Point-to-point Consumer designer agent, we find that it is afforded by Feedback Message, Respond to feedback message from Provider, Provider, Feedback message response code, Send message routine call, Send service request message from own physical address to Provider physical address via the Messaging Service, Message, Service, Physical Provider address, Physical own address, Provider address, Own address, Physical Device Address, Communication Device and Physical Connection.

If tracing back from the action: Don’t wait for feedback message from provider, it can be found that affordances found differ only with respect to the following alternatives: Logical Provider address vs. Physical Provider address and Logical own address vs. Physical own address. If these different preceding affordances are not relevant with respect to the initial affordances upon which we performed factoring analysis (the wait/don’t wait for feedback message from provider operation), then the differences can be factored out and subsumed into a relevant higher level affordance such as Provider address and Own address, which then also removes all lower-level dependent affordances from the analysis results. This leaves a common higher level “thread” of dependent affordances, which in our case, are linked to the consumer designer semiotic agent only. Factoring analysis also helps in identifying opportunities where semiotic agent models may be restructured to make them more concise by introducing, moving, redefining and/or relinking affordances.

Semiotic agent modeling in figure 9-3 thus reveals that whether a consumer is involved in a direct point-to-point integration and uses a provider physical address, or whether the consumer is loosely coupled using a logical or symbolic address, has no bearing on the operational meaning of synchronous and asynchronous messaging. Equipped with such a semiotic agent analysis, architects and designers can clarify and document the exact operational meaning they have in mind during design discussion as well as make clear the level lowest level of abstraction of terminology they assume unproblematic, thus avoiding misunderstandings and unintended design decisions.
9.6. Relating Semiotic Agent Modeling to Intentional Agent and Goal Modeling

Equipped with the results from the preceding semiotic analysis, the goal graphs included in section 6, which presented the intentional viewpoints of the component designer and the architects, can be improved. Figure 9-4 illustrates a strategic rationale model representing orthogonal design choices. This strategic rationale model is a reworked model of the rationale models illustrated in figure 6-3, which conflate both of these choices. The strategic rationale model in figure 9-4 replaces the task and goal decompositions included in the models in figure 6-3.

In figure 9-4, the design task Consumer publishes data to Provider at the top indicates the overall behavior that a designer implements. Operationally, this software behavior is however inherently ambiguous, since the detailed operational behavior is not yet defined. The overall design task corresponds to the symbolic operation “o4” in figure 9-3: Send service request message from own physical address to Provider physical address via the Messaging Service, defined by the Consumer design semiotic agent.

The design task is decomposed into two hardgoals, each indicating a different dimension across which the design task can be decomposed into alternative design approaches. Each hardgoal is decomposed into two alternative design approaches, where each refers to a semiotic agent in figure 9-3. Linking the alternative design tasks to different semiotic agents indicates a specialization of terminology in response to design interpretation and decision-making.

Design tasks are further decomposed into more specific design tasks, which are linked to related semiotic elements. For example, Physical point to point integration which is associated with the Point to Point consumer designer semiotic agent, is decomposed into the design task associated with the symbolic operation: Send service request message from own physical address to Provider physical address via the messaging service. In this example, extension links between semiotic agents serve as alternative ways of addressing the higher level goal.

Once elementary design tasks across different alternatives are defined, they can be combined into packaged design solutions. In figure 9-4, packaged design solutions are visually indicated by the “PS” prefix in the design tasks. Each packaged design solution is associated with a corresponding semiotic agent.

How much semiotic analysis is required? A key question during semiotic agent analysis is to what extent the analysis should be performed? How many levels of semiotic agents need to be explored and how fine-grained must the affordance dependence structures be made? Ultimately, the answer to these
questions is subjective: it is up-to the designers involved in semiotic agent analysis to decide when they feel operational meaning has sufficiently been clarified. However, the semiotic agent concept provides a useful technique to guide designers while making these subjective decisions. Using semiotic agents, the question “How much semiotic analysis?” is transformed to the question “To what extent can the experience and know-how of a designer (captured as a semiotic agent) be relied on to interpret the meaning of affordances and terms, and does it matter that the agent may invoke different interpretations?”

For example, in figure 9-3 the agent: Consumer designer perceives an affordance: Service. However, what is the exact meaning of a “Service”? During this analysis exercise it was decided not to analyze the term further, and rely on the consumer designer’s knowledge and skills to interpret the meaning. This choice is further strengthened by the observation that with respect to analyzing the operational meaning of synchronous and asynchronous messaging, the exact operational meaning of a “Service” is inconsequential. While the exact definition of “Service” might, indeed, be essential, semiotic analysis supports making these kinds of judgments and offers guidelines as to when and for what agents various terms require defining.
The interrelationship between semiotic agent modeling and strategic actor and goal modeling can also provide some guidance. The level of detail to which a semiotic agent model is refined is guided by the level of design detail necessary to compare and contrast the design implication of design choices on non-functional design goals. For example, in figure 9-3, designers may want to elaborate on the specific meaning of *Provider* or the exact *Logical Provider address*. These can be identified during a concurrent semiotic agent and intentional actor analysis during which these terms are placed both into a larger context and also defined in relation to each other.

**Structural meaning of terms:** Besides operational and intentional meaning of design details, there also exists a structural meaning of artifacts. Structural meaning relates to a physical or conceptual structure an artifact contributes to. For example, a service offered by a provider affords a “Service Interface”, which is a structural concept in software architecture, and which is, in turn, symbolically referred to by a consumer. It is possible that the enterprise architect, when discussing point-to-point integration, might envision the distribution of the provider’s interface structure to consumer components. However, ultimately, every structural meaning leads to an operational meaning, such as “What does a service interface do?” or, “What support does symbolic reference to an interface offer to a designer?” Thus, the structural meaning of an artifact may be a convenient intermediate concept when capturing artifact operational, designer actions on artifacts, and intentional meaning.

### 9.7. Conclusions

Semiotic agent modeling should be seen as complementing other conceptual design approaches, and as a technique to help clarify the meaning of design terminology. Clarifying terminology often goes hand in hand with design decision-making. Using and linking agent and goal modeling with semiotic agent modeling helps reveal and make more precise design details that derive from design decisions. Those details can then be linked to design and business goals, making the contribution of decision-making on business and system goals more precise.
10. Conclusions

10.1. Summary of Findings

This thesis set out to explore whether and to what extent an organizational modeling and analysis approach that was developed in the area of early requirements engineering could be utilized for modeling and analyzing architectural decision-making in organizational settings to meet business goals. To pursue this overall objective this thesis asked several research questions to explore the roles that business goals, NFRs, the organizations setting and quantitative NFRs play during architectural decision-making to meet business goals, and to identify modeling constructs for making the roles these concepts play more explicit.

To systematically address these questions, this research first defined a core IAL derived from the i* and the NFR Framework modeling frameworks, and explored during a number case studies the fit of its constructs with the decision-making roles of the aforementioned concepts. Following is a summary of findings, ordered according to the research questions:

Q1.1. What role do business goals play during architectural decision-making?

During this thesis research, it was observed that in commercial settings, business goals played a significant role during architectural decision-making. Business goals justified the inclusion of NFRs as part of the requirements, and directly, or indirectly via included NFRs, justified architectural decision-making. The negotiations around architectural choices ultimately lead to explicit or implicit negotiations of business goals. The perceived importance of business goals helped indicate the priorities given to NFRs in the organization, and consequently helped explain preferences given to architectural design options over others.

The ongoing pursue of business goals also suggest the introduction of new features and the creation of development teams to design and implement those features. Business goals were also articulated by stakeholders on behalf of other, higher-level stakeholders, thereby representing higher-level interests during architectural decision-making.

Finally, during industrial case studies the articulating of business goals helped align architectural decision-making with organizational strategy; and supported designers to explain architectural choices to business-oriented stakeholders.
During the industrial case studies, it was also found that stakeholders and designers did not specifically distinguish between goals of the business or non-functional requirements – both were often mentioned in tandem during design discussions when explaining or justifying design preferences. Usually, technical oriented stakeholders mentioned NFRs while business oriented stakeholders mentioned business goals.

Q2.1. What modeling constructs are appropriate for making business goals and the role they play during architectural decision-making explicit?

Business goals, and particular, business goals articulated in a qualitative manner were often tentative, subjective and conflicting in nature and not achievable in an absolute manner. Softgoals thus appeared to be appropriate for representing business goals. Business goals were negotiated and translated by clients and organizational stakeholders into non-functional requirements. Such translations and derived expectations were captured using intentional dependencies between clients and organizational stakeholders and between organizational stakeholders and designers in the development organizations. The tentative nature of intentional dependencies, in that they can fail helped indicate the negotiated and social nature of these expectations amongst clients, stakeholders and designers.

Criticality attributes associated with softgoals indicated the priority client, stakeholders and designers attributed to business goals, and NFRs derived from business goals, which in turn helped focus design explorations and select amongst alternative design options.

Softgoals in conjunction with contribution links also supported representing how business goals are refined by stakeholders towards more detailed business goals and eventually to NFRs addressable by stakeholders and designers in the development organization.

Finally, having business goals linked via contribution and dependency links to architectural design options, helps indicate how architectural decision-making trades off business goals with each other. Having such links between business goals and architectural design options captured, helps indicate how architectural choices matter to business oriented stakeholders, thereby enabling explanations that refer to the consequences of design options on business strategy and tactics while reducing the need to explain these in architectural design terms. This supports making the governance processes more explicit in the organization, as well as raises the level of abstraction by discussing architecture using business terminology.
Q1.2. What role do qualitative non-functional requirements (NFRs) play during architectural decision-making to meet business goals?

NFRs serve as a link between business goals and architectural design deliberations. NFRs characterized architectural design problems, in terms of software system qualities alternative design options tradeoff between each other. NFRs thus served as criteria to evaluate alternative design approaches, and helped linked those criteria to the business goals of the organization. NFRs when originating from different stakeholders in the organization help indicate conflicting demands that stakeholders and designers must deal with during architectural decision-making. NFRs also helped clarify and explain NFRs, such as when some more detailed NFRs are introduced when explaining some higher level NFRs.

NFRs also helped explore what aspects, components or parts of design approaches establish the difference between alternative design approaches. NFRs also helped reveal deeper reasons for choices. Finally, NFRs also played a role to explain reusable knowledge, such as to explain the utility design patterns.

NFRs sometimes also served as “sign posts” to refer to reasons and criteria of choices, without elaborating in more detail what an NFR entails and how it relates to higher-level business goals.

In an organizational context, NFRs articulated by different designers helped compare and contrast alternative design approaches each designer put forward, and hence drove the discussion and evaluation of proposed design approaches.

In context of reusable design solutions, such as in design patterns, NFRs helped explain the design problem a design solution addresses, and why one design solution is preferred over other solutions.

Q2.2. What modeling constructs are appropriate for making qualitative NFRs and the role they play during the architectural decision-making to meet business goals explicit?

The concepts included in core IAL such as softgoals, and contribution links and types, intentional dependencies, and design tasks (intentional tasks) were sufficient to represent, link and analyze the relationship between NFRs and business goals on one hand and NFR and architectural decision-making
on the other. The refinement of NFRs and related decomposing of design tasks helped represent how alternative design approaches differ. However, core IAL needed extending to support dealing with reusable design solutions; this was accomplished by defining a packaged solution concept. The softgoal concept supported representing the design “forces” explained the design problem at hand, as well as why the proposed reusable solution is preferred over other alternative design solutions.

Q1.3. What role does the organizational setting play during architectural decision-making to meet business goals?

During case studies, it was observed that the organizational setting analyzed helped indicate whether members of the organization jointly or separately worked towards achieving the organization’s business goals. When organizations established a team to deal with a specific need, different stakeholders and designers holding different organizational responsibilities jointly work on addressing those needs. In another circumstance observed the organizational responsibilities were distinct, but with some overlapping area of decision-responsibility.

Organizational roles also helped delineate the scope of decision-responsibility thereby suggesting what decisions should be made by some stakeholders/designers and what decisions could be handed off to others.

During case studies, it was also observed that the responsibilities associated with architectural decision-making occurred in context of other organizational responsibilities, such as higher-level management. The responsibilities of higher level management included establishing organizational responsibilities, supplying them we goals and tasks to pursue, as well as, arbitrate when conflicting demands required dealing with evaluating and choosing amongst alternative design approaches.

Q2.3. What modeling constructs are appropriate for making the organizational setting and the role the organization setting plays during the architectural decision-making to meet business goals explicit?

Core IAL concepts such as intentional role, position and agents, in conjunction with intentional dependencies between them helped indicate the distribution of organizational responsibilities within an
organizational setting, and how these related to and/or influence architectural decision-making in the organization. This included the use of intentional dependencies to indicate the design goals and tasks some organizational stakeholders and designers expected others to address or fulfill, as well as to indicate the higher-level organizational stakeholders/designers who prioritized goals, and arbitrated conflicting decision preferences amongst lower-level organizational stakeholder and designers.

It was also found that some extending to core IAL was needed to representing type of responsibilities not covered by core IAL. This included intentional collective and intentional viewpoint concepts: The intentional collective concept supports representing joint decision-making of different organizational stakeholder and designers that are part of a team; put also to represent the organization as a whole. The intentional viewpoint concept supports representing organizational responsibilities that are not fully mutually exclusive.

Q1.4. What role do quantitative NFRs play during architectural decision-making to meet business goals?

In the case study that studied quantitative NFRs specified for control systems, quantitative NFRs captured the real time requirements as well as processing capacity needs of control systems. From a business perspective, quantitative NFRs indicated the market positioning of control system products, from simple embedded systems that targeted the lower end real time needs and processing capacities, which are positioned to serve lower-end markets to multi-site multi-server control systems that are positioned to server high-end markets.

Q2.4. What modeling constructs are appropriate for making quantitative NFRs and the role they play during the architectural decision-making to meet business goals explicit?

The measurement meta-model, the processing capacity calculation method based on scenario modeling integrated with a qualitative analysis of qualitative NFRs supported trading off qualitative and quantitative NFRs while positioning control system products in market segments with specific qualitative NFRs. Using this integrated method quantitative NFRs were not represented in a graphical manner. However, scenario models parameterized with quantitative NFRs, as well as templates associated with
selections of operationalizing softgoals supported the aforementioned integrated analysis method.

Following are some additional findings observed during the modeling and analysis of case studies.

**Goal- vs. actor-oriented modeling and analysis**: Studying architectural decision-making in industrial settings led to the conclusion that design discussions in some organizational settings lent themselves more to goal-oriented analysis, while for others, actor-oriented analysis was appropriate.

For example, at Mitel’s WML project (Chapter 5), the stakeholders and designers who were observed and interviewed saw themselves and each other as participants in a design team collectively responsible for finding the best possible tradeoff among the interests and concerns put forward by all. Stakeholders and designers, even though holding different organizational responsibilities, jointly analyzed the problem and formulated solution approaches. Their approach was to work as a team.

In such situations, it was not readily observable that intra-organizational boundaries of responsibilities played a significant role during decision-making. Stakeholders and designers, each holding different organizational responsibilities, were considered primarily as a “heterogeneous” source for concerns and interests during architectural decision-making.

Similarly, design goals were often prioritized by project participants as part of their collective decision-making effort, and priorities were not imposed in a top-down manner from the higher-level stakeholders included in the team. In one case study (the “OAMP” study summarized in Appendix A), one member of upper management explained that the higher-level goals and priorities included in the goal graphs were “motherhood” statements of the organization – suggesting that the knowledge of these goals are ingrained in the organizational participants and part of the organizational culture.

Another example, the Console case study (summarized in Appendix A), presented a cross-functional team that was observed during the formulation of the design objectives, and the subsequent prioritization of those objectives in order to evaluate and select amongst architectural approaches for an attendant console product. To support the selection process, the team developed, jointly and ad-hoc, a decision table of decision criteria (a list of NFRs) versus alternative architectural approaches. The team then assessed each alternative in relation to the criteria, placing a numeric value from one to ten (relative weight of importance) in the table cells. The fact that each cross-functional team member held a distinct organizational responsibility was not used as a factor when discussing the relative weights to place in the table cells, nor could it be observed that any member exercised any authority over others during the prioritization discussions.

The decision table developed by the Console team members was then complemented by an NFR goal
graph, which was produced by this author from a transcript of the recorded discussions. The NFR goal graph captured the design reasoning and justifications that team members discussed, while giving weights to the different design criteria. The NFR goal graph also helped clarify some of the criteria included in the table (mainly through the refinement of NFR softgoals into more precise NFR softgoals), indicating that the meaning of the criteria was not always shared amongst team members. The goal graph complemented the ad-hoc table, much like the NFR goal graph included in the KWIC case study in chapter 4 (figure 4-1) complemented the architectural reasoning captured in table form (table 4-1).

The Phoenix insurance study (Chapter 6), however, illustrated the situation where an actor-oriented approach was more appropriate. During The Phoenix Insurance case study, the organizational responsibility boundaries amongst stakeholders and designers played a role during architectural decision-making. The participants of the discussion did not see themselves as having shared goals, contrary to the Mitel team. The SOA architect and the component designer each held distinct design responsibilities. During design argumentations, each appealed to different design goals derived from higher-level decision-makers at the management level in the organization. Despite the fact that both the consumer component designer and the SOA architect worked for the same IT department, they did not act as a team, and there was no coherent collective thinking unifying their discussions and decision-making. The organizational relationship between them was therefore argumentative rather than cooperative. Utilizing intentional actors to delineate their respective organizational design responsibilities was thus appropriate and helpful to elucidate the different vantage points brought by each party to the design issues at hand.

The actor-oriented representation of higher-level organizational stakeholders, their prioritization and the delegation of design goals, was also found useful by the CTO and SOA architect to communicate organizational prioritizations, and the rationales for prioritizing design goals, amongst lower level (in the organizational sense, i.e. the management hierarchy) decision-makers. Hence, the actor-oriented approach to modeling architectural decision-making in the organization was seen as useful by the CTO and SOA architect to support the governance processes in the organization.

**NFR type and topic distinctions:** Goal graphs (either NFR goal graphs or strategic rationale models) produced during case studies made use of only a subset of modeling features provided by the NFR framework or i* Strategic Rationale modeling approach. For example, it was often not necessary to analyze NFRs softgoals into type and topic, nor was it necessary to use this distinction to refine NFR softgoals into more detailed softgoals. Nevertheless, even without these modeling features, the goal graphs were considered useful by practitioners to communicate key design alternatives and their
respective justifications and tradeoffs.

In addition, while softgoal refinements were often not structured strictly along type or topic softgoal decompositions, they still were frequently refined by type or topic without making the structure explicit in the model. NFR Softgoals were usually labeled using single phrases (such as reduce new product risk, or seamless UI interaction), which was of sufficient utility to practitioners in representing and communicating the design goals and reasoning of their decision-making. This for example helped them in asking “why” questions to unearth the higher-level, and more fundamental NFRs, that justified design decision-making.

**Degree of formalization when dealing with qualitative and quantitative NFRs:** While during most case studies the goal graphs produced, and in particular the labels used to denote goal in goal graphs, required no formal structuring, during the Siemens case study (Chapter 8) more formality was necessary. One of the objectives of the Siemens case study was to support the integrated analysis of quantitative and qualitative NFRs with a computational tool. Since the researchers and practitioners at Siemens had been using Microsoft Excel to capture and analyze quantitative NFRs, they were comfortable with approaches that required specifying and analyzing NFRs more formally.

To support the GS-PND method (Song et al., 2009) with analytical tools it was necessary to specify quantitative NFRs using a measurement meta-model and to analyze qualitative NFRs into types and topics, and to link measurements and type and topics to scenario models. This higher degree of formalization thus enabled parameterizing scenario models using both quantitative and qualitative NFRs, which in turn supported the computational analysis of the qualitative and quantitative NFRs in an integrated manner.

**Beliefs:** Although goal models produced during the case studies captured domain-specific design reasoning knowledge, belief elements that support the modeling with additional domain-specific knowledge were seldom observed and used. This may indicate that at least during the observed design discussions participants seldom reflected on the validity of design claims made to further support or refute each other’s claims. One example where a belief element was included into a model is illustrated in chapter 7, where some belief elements can be seen in the goal graph that represents the design reasoning included in the deviation pattern text. Incorporating these belief elements helped indicate the design assumption that must hold for the design approach to address memory utilization NFRs. The opportunity to strengthen the goal model by use of belief elements was however identified by the author of this thesis and was not mentioned in the associated design discussion that was reported in the literature. Whether belief elements could play a more significant role during modeling and analysis of
design discussions in industrial settings is thus inconclusive, and requires further studying.

**Contribution types**: Both the NFR framework and i* support finer distinctions between different contribution types, including those labeled sufficient (make), partially sufficient (help), and the like. During the goal modeling in the case studies, those distinctions were seldom used. More importantly was the indication of whether a contribution was positive or negative, and whether it was an intended contribution or a side effect (correlation). The resulting goal graphs, with this more limited set of contribution types, nevertheless were considered useful to stakeholders and designers for representing and communicating design reasoning among themselves. Consequently, there was no opportunity identified to analyze the goal graphs more formally using semi-automated procedures, such as the labeling procedure offered by the NFR framework (Chung, 1993). However, even without the evaluation procedure the goal models produced during case studies were considered useful by practitioners to offer support during architectural decision-making.

**Softgoal refinements**: Refining softgoals into more specific softgoals was found to be a key technique in clarifying NFRs during goal modeling. Such refinement relationships, even though mostly no more than three levels of refinement, where helpful in clarifying how stakeholders and designers interpreted non-functional requirements and how they proposed to address them. Using goal refinement higher-level NFRs, such as “competitiveness of products” (figure 5-6), was clarified to mean “user satisfaction” and “enable 3rd party services”.

High-level NFRs often have project specific meaning, and softgoal refinements can help pin down the specific meaning the stakeholders and designers in a project mean. For example, in the WML case study, “reduce risk during software development” was interpreted by stakeholders as a need to work towards backwards compatibility of telephone sets, an interpretation that was captured in the goal graph (figure 5-6). During the Siemens case study, there was also the requirement to reduce risk during software development. However, in this case, a reduce risk goal was interpreted to mean to reuse already existing software assets (the reduce risk NFR was however not specifically analyzed during the Siemens case study). The development of an application platform for this purpose was thus a part of the reduce risk objective and strategy.

**Intentional dependencies**: During the modeling and analysis of the industrial case studies, it was found that the use of intentional dependencies was mostly limited to indicating the delegation of design goals and tasks amongst organizational responsibilities and participants. During modeling, intentional dependencies were thus mostly used in a descriptive manner rather than to support an intentional analysis of opportunities and vulnerabilities (E. Yu, 1994; S. E. Yu, 2009). Intentional dependencies thus
mostly served as traceability links between the intents of the organizational participants and their delegated goals. More studies are needed to explore the utility and use of intentional dependencies as a means for analyzing how intentional actors seek out and evaluate collaborative opportunities or identify possible vulnerabilities in their relationships with other intentional actors (E. Yu, 1994).

**Usefulness of goal-oriented modeling:** Practitioners found that goal-oriented modeling and analysis of architectural decision-making was a stark departure from the kind of modeling and analysis they usually performed. However, practitioners immediately saw the value of goal modeling and were interested in exploring their usefulness to support decision-making. In all the case studies, when presented to project members, goal models helped consolidate design discussions and thus supported practitioners in converging discussions and arriving at decisions. This was the case although they had been constructed “off-line” (from transcripts or documents, by the author of this thesis).

In their feedback, practitioners in all the case studies pointed out the value of goal models to support documenting architectural decision-making, including the rationale behind the choices, and to link architectural decision-making to their business justifications. They also saw the value in having goals clarified through refinements. They however pointed out that tool support would be essential for the goal-oriented notation to be adopted within the development organization.

The main utility of goal graphs was seen as an approach that compactly summarizes, consolidates and facilitates design discussions, that supports understanding disagreements, by uncovering differences in the interpretation of vague high-level goals, that provides design rationale documentation for future reference, and that provides a compact representation that helps trace decision-making to higher-level organizational stakeholders and business goals. Since all creation and modeling of goal graphs were done by the researcher additional studies are needed to evaluate the overhead of such modeling when done by practitioners themselves.

**Usefulness of actor-oriented modeling:** In all but the Phoenix industrial case study, goal-oriented modeling received more attention than actor-oriented modeling, because practitioners discussed design rationales and approaches, collectively. However, during The Phoenix case study actor-oriented modeling was specifically used to highlight different argumentation viewpoints of different designers, and to support tracing design decision-making and intents across decision-makers in the organization—a link that was seen by the CTO and SOA architect as supporting SOA governance processes in the organization. They saw the value of actor-models in helping them in justifying the architectural choices they proposed to implement in terms of the organizations higher-level goals. As mentioned in chapter 6, while both the CTO and SOA architect considered actor-models a useful technique, they also felt that a
simplified version of actor-models would in particular be useful to support offering quick reminders to relevant stakeholder as to why choices were made.

**Process-orientation:** During the case studies, no strict “process-oriented” approach to architectural decision-making was observed. Stakeholders and designers did not develop architectural descriptions in a stepwise manner in response to predefined objectives. Nor did design states evolve systematically through a series of decision-making activities. Instead, stakeholders and designers often relied on informal descriptions of a problem, and then brainstormed various alternative solutions, with one solution approach eventually emerging in favor of others.

Also, stakeholders and designers did not always start by enumerating high-level design goals that need to be addressed and then proceeding to refine these until alternative architectural approaches that address design goals could be identified. Instead, design discussions were unstructured, moving in all directions from design objectives to solutions, from solutions to alternative solutions, from solutions to evaluation criteria, and from criteria to higher-level justifications.

Agent and goal-oriented models were constructed during this research in a manner that reflected the exploratory process. Usually, it began with several design alternatives and softgoals that were each then developed in a top-down or bottom-up manner, which elicited further identification of additional softgoal refinements or contributions. By asking “who” and “why,” higher level objectives were identified, along with additional organizational responsibilities that were then interconnected via intentional dependency links. This approach to model construction was helpful during the research process to explore the design reasoning and decision-making, to asking questions and uncover the deeper reasons for choices.

**Abstracting architectural design discussions, towards business goals:** Some stakeholders reviewing the models saw the linking from architectural decision-making to business goals as potentially aiding in systematically explaining architecture, and related architectural decision implications, to upper management.

### 10.2. Summary of Contributions

At the beginning of this thesis research, little work had been done on modeling and analyzing software architecture decision-making in general and on modeling and analyzing architectural decision-making using intentional actor and/or goal concept in particular. This work can thus be seen as contributing to several areas of research including goal-oriented modeling and analysis, actor oriented modeling and analysis as well as research on architectural decision-making in general.
Contribution to goal-oriented modeling and analysis: Chung and colleagues pioneering paper on the application of the NFR Framework to model and analyze the KWIC systems architecture was a first attempt to representing and reasoning about architectural decision-making using goal-orientation, and in particular the softgoal concept to represent and reason about qualitative NFRs during architectural decision-making (Chung et al., 1995).

This thesis research extended Chung’s and colleagues work by further exploring the use of softgoal concepts derived from the NFR and the i* framework (in form of core IAL) to model and analyze architectural decision-making documented in the literature and observed during industrial projects, identifying limitations and proposing extensions. In the course of this research work, the NFR Framework was for example adapted to represent and reason about design patterns and the application of design patterns during architectural design (Daniel Gross & Yu, 2000, 2001d). This adaptation can be seen as a contribution to both goal-oriented modeling and analysis of architectural decision-making, and to the design pattern domain and approach (Buschmann, 1996; Coplien, 1996; Gamma et al., 1995).

Contribution to architectural decision-making research: this thesis work can also be seen among the first that made the architectural decision a first class modeling entity. This is a research theme picked up by later works such as by Jansen and Bosch, Tang et al. and Zimmerman et al. (Jansen & Bosch, 2005; Tang et al., 2009; Olaf Zimmermann et al., 2007). This research was also amongst the first to explore, model and analyze during industrial case studies the linking of business goals through NFRs to architectural decision-making (e.g. (Daniel Gross & Yu, 2001c)). The importance of such a systematic linkages has recently been identified and investigated by Kazman et al. (R Kazman & Bass, 2005; Rick Kazman et al., 2005) and Clements et al. (P. Clements & Bass, 2010). These works however focus on methodical guidelines and not on offering a systematic modeling and analysis technique.

Contribution to actor-oriented research: This thesis work was also a first to apply intentional actor concepts to modeling and analyzing architectural decision-making (Chung et al., 1999; Daniel Gross & Yu, 2001c, 2010b). During these case studies intentional actors represented human decision-makers or their decision-making roles in organizational settings, an approach to intentional actor modeling that is still unique within the body of i* research applied to architectural design domain (see comparison to representative approaches in section 3.3).

Related to this interpretation of intentional actor concept, this thesis research also explored the inclusion of organizational stakeholders, who participate in architectural relevant decision-making, in intentional actor models of architectural decision-making. Including the broader and higher-level organizational context to justify and explain architectural decision-making is based on the specific
interpretation of intentional actor proposed by this research, which does not distinguish between social and technical focal points of interest during the modeling and analysis of architectural decision-making. No comparable work yet exists that explores this avenue of modeling and analysis.

This research also contributed to aspect-orientation research (Kiczales et al., 1997; S.M. Sutton Jr. & Rouvellou, 2000; Stanley M. Sutton Jr. & Tarr, 2002), by explored the utility and use of concern reasoning agents during architectural decision-making (Daniel Gross & Yu, 2002). Concern reasoning agents are a proposed application of the intentional role concepts, an abstraction of organizational responsibilities, to the modeling and analyzing of design reasoning associated with the design and composition of aspect and modules during architectural design and decision-making.

10.3. Future Directions

During this thesis research several areas were identified that can be expanded on during future research. Future work can be classified into several areas: theoretical underpinning, modeling constructs, empirical investigations, knowledge management, practical methodologies, tools and reasoning support as well as application domains.

Theoretical underpinning: This thesis research illustrates that architectural decision-making is intertwined with the creation and evolution of organizational structures and responsibilities in development organizations. Future work could focus on exploring the organizational design theory and approaches to provide guidance during actor oriented modeling and analysis of the development organization.

Organizational design theories could help further expose and make amendable to modeling and analysis the interrelationship between architectural decision-making and organizational design and evolution. Organizational design theory could for example serve as background for explaining broader structuring principles behind organizational responsibilities, giving intentional modelers additional criteria and tools during the creation and evaluation of intentional actor models of architectural decision-making. Organizational theory concepts could also provide common terminology with organizational stakeholders when introducing intentional architectural modeling and analysis to managerial stakeholders. Works by authors, such as Galbraith, Giddens and Burton et al. (Burton, DeSanctis, & Obel, 2009; Galbraith, 1971, 1977, 2002; Giddens, 1986) in the area of organizational design and evolution, as well as works by Beer and Malik in the area of cybernetics and the management of complex systems (Beer, 1972, 1979; Malik, 1984), appear particularly relevant to providing organizational theory background.
This thesis research illustrated the applicability of organizational semiotic theories of Stamper and Liu (K. Liu, 2000; Stamper, 2006) to the definition of architectural artifacts. Future research can further expand on this line of research and further investigate the application of organizational semiotic on the definition, creation and use of terminology within actor and goal models.

A key innovation proposed in this thesis is the use of intentional actors to attribute socio-psychological properties to technical artifacts under construction. The technical is thus treated as social within an organizational setting. This approach of promoting the technical to become a social participant in social interactions fits well with some sociological theories of technology such as Latour’s actor network theory (ANT) (Latour, 1987, 1996). Future work could further explore key observations and properties of actor networks postulated by ANT, such as the notion of “translation” to explain the social meaning of technical artifacts across different social participants, each belonging to a different community of practice, to further enhance and guide the kind of intentional actor modeling and analysis within and across development organizations.

**Modeling constructs:** During this thesis research, intentional actor modeling and analysis mainly focused on modularizing the reasoning and decision-making associated with stakeholders and designers. There was little use was made during case studies of intentional dependencies, in particular in analyzing intentional dependencies across actors to explore the strategic aspect of opportunity seeking and vulnerability avoidance. These intentional dependency concepts are however central to the definition and analysis of distributed intentional agents in the original i* modeling framework (E. Yu, 1994).

Future work could particularly focus on the defining, evolving and analyzing intentional dependencies amongst intentional actors during architectural and architecturally relevant decision-making in organizational settings. Intentional dependencies could also be explored to model and analyze the shifting boundaries of responsibilities across intentional actors, for example, when stakeholders negotiate and evaluate how much decision-making to retain in-house (or within a team, as illustrated in chapter 5), and how much to delegate to others.

Future research could also look at integrating the definition of intentional elements with knowledge of the structural and behavioral meaning of architectural artifacts. Such tighter integration could afford additional kinds of analyses, such as the identification of overlapping design goals across different intentional actors, each responsible for different parts of a software system (see chapter 6). The User Requirements Notation (Amyot & Mussbacher, 2003; ITU-T, 2008) is an example of somewhat tighter integration between the Goal Requirement Language (GRL), a variant of i* with Use Case Maps (UCMs), an architectural modeling language. Future research could further explore tighter integration between
This thesis research did not customize intentional modeling and analysis to specific architectural structures often found in development organizations, such as product families, product lines or application frameworks or platform (Bosch, 2000; Niemelä & Immonen, 2007). Future research could explore such more specialized architectural constructions to identify specialized intentional constructs specifically applicable to the modeling and analysis of such specialized architectural structures in organizational settings.

**Empirical investigations:** An important lesson learned during this thesis research is that collecting empirical research data during real-life development projects is difficult, and that obtaining access to relevant stakeholder and designers and/or to project documentation is often problematic. Yet, to study distributed decision-making in organizational settings, it is crucial to have access to relevant stakeholders and designers in organizations.

One approach to getting access in business organization is to provide evidence of the business value the modeling approach can provide. Future work should thus focus on establishing quantitative and the qualitative benefits when architects and other organizational stakeholders utilize intentional modeling and analysis in organizational settings. This could include for example studying the effort to produce intentional models, and the benefit such models have on particular decision-making tasks, such as to study the increased effectiveness in communicating design justifications to different stakeholders and designers in the organization, or to study in improved perceived quality of produced architectures (C. Lee, 2008).

Future research should also investigate the use of intentional modeling and analysis in-situ – during architectural problem solving. Ideally, the stakeholders and designers themselves should be given adequate training and tool support to model and analyze their own design discussions and decision-making. Alternatively, a study could look at the effectiveness of intentional modeling during facilitated group modeling and analysis sessions, similar to the “Dialogue Mapping” approach proposed by Jeff Conklin (2006).

Such an in-situ approach is particularly important for evaluating the process-oriented modeling and analysis features (Chung, 1993) associated with goal-oriented modeling and analysis. This suggests studying whether stakeholders and designers would be helped when following an approach were business and design goals are put up front, and are then clarified iteratively, and in a stepwise manner, during individual and/or group analysis discussions. This would include actor-oriented explorations whereby other relevant decision-makers in the organization, whose intents and decision-making has an intentional modeling and other non-intentional modeling notations and languages.

215
affect on goal achievement, are identified and included in the modeling and analysis process.

Another line of research with a focus on decision-making could further investigate the types of architectural and architectural relevant decisions and justifications made in organization and the implication of decision types on intentional modeling constructs. Such work could expand on works of Smolander et al. (Smolander, 2002), who systematically explored, using a grounded theory research methodology (Strauss & Corbin, 1991), stakeholders and rationales for using architectural descriptions in development organizations. Studies that follow up on works by Herbsleb and Grinter (1999a; 1999b) on the distribution of responsibilities in development organizations as well as the integration of distributed work, could also provide important insights into expanding the intentional modeling and analysis approach to support such decision-making tasks.

Knowledge Management: Reusing decision argumentation knowledge has much potential in improving the efficiency and quality of architectural decision-makers (O. Zimmermann et al., 2008; O. Zimmermann et al., 2009). This thesis proposed some extensions to core IAL to represent, capture, reuse and reapply organizational responsibilities, during architectural design in organizational settings. Future could further explore the utility of capturing and reusing such organizational architecture knowledge. In particular, whether the construction and evolution of organizational responsibilities associated with design and evolution of architectural design can be made more efficient. This also includes further exploring the idea of “design by synthesis” briefly discussed in chapter 6, whereby distinct intentional models that capture generalized responsibility knowledge are systematically merged to identify and establish combined responsibilities and organizational settings.

Practical methodologies: During this thesis research this thesis author performed agent and goal modeling of architectural decision-making while analyzing research data. To facilitate the adoption of intentional modeling and analysis in industrial setting, future research needs study adoption and use of intentional modeling by organizational participants themselves.

The adoption and use of intentional modeling and analysis raises various research questions. For example, in what usage setting would intentional modeling and analysis be performed? Would it for example be of advantage to perform the modeling and analysis during facilitated meetings (Conklin, 2006)? Who would be the main produces and consumers of such models; how could intentional models be used to present “the right information to the right stakeholders and decision makers in the organization (P. Kruchten et al., 2010)?

Would models be kept as historical records of decision-making, or would they mostly be used to facilitate communication amongst stakeholder and designers during project discussions, without specific
intent to retain such models for future reference and use and reuse (O. Zimmermann et al., 2008). Karsenty in his study of the utility of captured design rationales has, for example, shown that only part of a historical record of design rationales have been found useful when reviewed to facilitate future tasks that required prior rationale, and that in about half the cases rationales needed in the new task context had not been captured (Karsenty, 1996). Replication studies could be performed to identify whether intentional models that focus on stakeholder’s goals as the means for capturing rationales would improve the utility of captured rationales during future system maintenance and evolution effort, or when rationales could be reused during new projects.

Also, questions about training need to be considered. How much training is required before intentional modeling and analysis can be done by facilitators or architects? How much training is needed to read, understand and analyze the models? How could models be presented to different types of stakeholders to make the knowledge embedded in the models readily accessible. Management stakeholders, for example, might benefit from “dashboard’ views indicating the consequences of ongoing decision-making in the organization on organizational objectives.

**Tool and reasoning support:** During case studies, study participants acknowledged the perceived utility of intentional modeling and analysis approach presented to them, however, explained that without adequate tools support, it would be difficult to promote the approach within their respective organizations. A key question for future research is thus to determine what kind of tool support to offer to facilitate the adoption and use of intentional modeling and analysis of architectural decision-making in an organizational setting. As indicated in the previous section, different types of users, as well as different usage scenarios, would need different kind of tool support.

Tool support also raises the question of the kind of reasoning and analysis support to provide, in particular how much automated model analysis, in particular in light of softgoals, which come to deal with NFRs that are subjective, and difficult or even impossible to formalize. This also includes the need to dealing with the social semantics of intentional softgoal dependencies across intentional actors, which require human interpretation and agreement (E. Yu, 1994). Automation could thus mainly focus on argumentation support (Jureta et al., 2009), and to support stakeholders and designers in dealing with various commitments across a network of actors (E. Yu, 1994; S. E. Yu, 2009).

Automation could also focus on keeping track of model construction and evolution with respect to argumentation articulated, and changed, by different stakeholders and designers, as well as contextual knowledge that becomes available. This could also include support for process-orientation, such as stepwise construction of models using knowledge bases, as well as ongoing evaluations of justifications.
to what extent higher-level goal were achieved (Chung, 1993; E. Yu & Mylopoulos, 1996). Finally, the utility of interactive labeling procedures to support evaluating intentional actor and goal models also needs further studying (Chung, 1993).

**Application domains:** some application domains appear more suited to intentional actor and goal modeling and analysis of architectural decision-making than others. Key characteristics of application domains that appear to fit well the intentional actor concept are that they:

a) Involve multiple organizations or organizational participants (stakeholders and designers);

b) Organizations or organizational participants have a sufficient degree of autonomy to direct their own interest and choices;

c) Collaboration between organizations or organizational participant occurs when there exists some direct or indirect commonality of interest, which guides decision-making.

In such environments intentional modeling and in particular the modeling and analysis of intentional dependencies could provide deeper insights into the feasibility of the actor’s goal achievement within a network of interdependent actors, interests and decision-making.

This thesis research further explored intentional actor and goal modeling in enterprise organizations that develop software clients or for in-house consumption, and that exhibited at individual or team level to some extent the aforementioned characteristics. Other application domains such as global software development, involving distributed teams (Herbsleb et al., 2005), and outsourcing and modular packaging relationships between organizations (Garud, Kumaraswamy, & Langlois, 2003) as well as software ecosystems (E. Yu & Deng, 2011), also appear promising domains for future study.
Appendix A. Additional Case Studies

This section reviews additional studies performed during this thesis research but which were not included in the main thesis document. Some of the studies replicate results reported in the main thesis document, including these would have led to repetition. Others focused on research themes somewhat removed from the main theme (distributed decision-making) while replicating results on goal-oriented modeling and analysis. Instead, these studies are included in this Appendix with a brief description, and summary of results obtained.

**Mitel OAMP industrial case study:** The acronym OAMP stands for Operation, Administration, Maintenance and Provisioning. The purpose of the OAMP development project was to create a reusable application platform that supported managing and administering telephone system applications across the different Mitel products. During the Mitel OAMP case study project, documentation including business case, requirements, and architectural design documentation were analyzed. Both i* and NFR framework models were produced and presented to the project’s management team, who provided positive feedback for the usefulness of such a modeling and analysis approach, and suggested studying additional development projects at the Mitel organization.

Feedback included the meeting participants’ perceived value of having a goal graph representation for representing high-level business goals (“organizational motherhood statements”), and how these link to architectural decision-making; and the ability to represent and evaluate alternative architectural design approaches in terms of NFRs.

**Mitel Console industrial case study:** A console is a telephone system used by switchboard attendants in small business organizations. The console project’s goal was to develop a new, PC-based, attendant console hardware/software system. During this case study, several design discussions of a cross-functional team were observed and recorded, and relevant portions of it transcribed, modeled and analyzed, with the focus on the non-functional requirements.

Several NFR goal graphs of console design reasoning discussions were produced, which were compared to the tabular and quantitative analysis approach that was produced ad-hoc by the cross-functional team during design discussions. Due to logistic reasons, no feedback from project’s participants was directly obtained. However, comparing and contrasting the NFR goal graphs produced with the tabular analysis of alternatives, helped evaluate the strengths and weaknesses of NFR goal graphs vis-à-vis the tabular approach.
It was found that while the tabular, quantitative evaluation approach was useful in systematically arriving at one preferred design alternative over others, the quantification of feature priorities underlying the tabular approach failed to capture why some feature were preferred over others. The NFR goal graph method, on the other hand, by documenting rationales, indicated finer-grained distinctions during the analysis of alternative design approaches. The two approaches were thus complementary, each with own strengths and weaknesses.

IBM Eclipse ALF industrial open source case study: The ALF (Application Lifecycle Framework) project goal was to develop an event-driven workflow management engine for development environments of enterprise applications (http://www.eclipse.org/proposals/eclipse-almiff/). Several commercial tool-development organizations collaborated on this project.

A key aim of studying this development project was to further explore distributed design reasoning in an organizational setting. However, after some initial analysis of the project data, it became clear that project participants, although belonging to different tool provider organizations handled the project as a coherent virtual team. Little distributed reasoning amongst project participants occurred, as was apparent from the online telephone conference calls, and documentation produced.

Nevertheless, some preliminary organizational modeling was produced in which the ALF project, as a whole, was placed in context of the participant’s customers, and other third party development organizations. This did offer some opportunities for distributed modeling of intents and decision-making. However, since the ALF project was eventually dismantled, no feedback on these models was obtained.

Visual Basic/Prolog Telos prototype implementation: In order to be able to analyze research data using qualitative analysis approaches of textual data (Strauss & Corbin, 1991) in conjunction with conceptual meta modeling, the author of this thesis undertook the development of a prototype tool that combined these in one tool. The architecture of this tool, which connected a programmable word processing tool (MS-Word) with a prolog implementation of the Telos knowledge representation language33 (Jeusfeld, 1992; Mylopoulos, Borgida, Jarke, & Koubarakis, 1990), posed several design challenges.

One challenge involved dealing with crosscutting design time concerns across different architectural

33 At the time, the Concept Base deductive database system (Concept-Base, 2002) was not available on the Windows platform, a prerequisite for connecting such a system to a textual analysis tool implementation add-on to MS-Word.
elements. To systematically represent, document, and analyze cross-cutting concerns, a concern reasoning agent concept was introduced related to the notion of separation of concerns. Concern reasoning agents allow the reasoning about concerns during the design and composition of aspects and modules. Results of this case study were published at an international workshop (Daniel Gross & Yu, 2002). This case study was not included in the main thesis text, since the concern reasoning agent concept illustrated in the paper did not specifically deal with distributed reasoning.

**High performance web server cache:** This case study analyzed the architectural design decisions included in a high performance cache that were reported in a series of papers (Degenaro et al., 2001; Iyengar, 1999; S.M. Sutton Jr. & Rouvellou, 2000). Some actor- and goal-oriented representations of design reasoning that shaped different cache subsystem designs were developed. This modeling and analysis also drew from work that classified the design concerns into a concern modeling framework (S.M. Sutton Jr. & Rouvellou, 2000), and thus further a elaboration on the concept of concern reasoning agents, with links to non-intentional concern modeling framework.

**Observer pattern case study:** This case study aimed at developing a more formal representation of design goals and design reasoning by using more formal approaches to specifying the intended behavior and structure of design patterns (Helm, Holland, & Gangopadhyay, 1990; Holland, 1992, 1993), and architectural transformation rules based on graph rewriting (e.g. (Fahmy & Holt, 2000)) linked to quality requirements. Some attempts to include intentional actors into the modeling approach were also made.

Initially, this approach appeared to promise interesting benefits in specifying and systematizing the application of architectural design approaches to the systematic evolution of architectural artifacts. Eventually, however, this thesis author felt that the level of formality needed to apply such an approach limited its utility in industrial settings. Furthermore, it was not clear whether the level of formality of this approach was a good fit, when dealing with intents and reasoning of intentional actor in organizations, who often need to deal with intents and choices that are neither clear-cut, nor amendable to formal modeling and analysis. In any case, the effort to formalize solution approaches would not likely contribute in a major way to the decision-making tasks. This line of research was therefore not further pursued.
Appendix B: Notation Legend

NFR framework and IAL notation:

<table>
<thead>
<tr>
<th>Modeling element</th>
<th>NFR Framework (1995)</th>
<th>NFR Framework / OME(^{34}) Tool</th>
<th>IAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softgoal</td>
<td>O</td>
<td>![cloud]</td>
<td>![green_oval]</td>
</tr>
<tr>
<td>Operationalizing Softgoal / Design Task</td>
<td>O</td>
<td>![cloud]</td>
<td>![design_task]</td>
</tr>
<tr>
<td>Resource</td>
<td></td>
<td>![rectangle]</td>
<td></td>
</tr>
<tr>
<td>Goal</td>
<td></td>
<td>![green_oval]</td>
<td></td>
</tr>
<tr>
<td>Claim/Belief</td>
<td>![cloud]</td>
<td>![cloud]</td>
<td>![brief]</td>
</tr>
<tr>
<td>Means-Ends link</td>
<td>![arrow_up]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task decomposition link</td>
<td>![arrow_up]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contribution link</td>
<td>![arrow]</td>
<td>![arrow]</td>
<td></td>
</tr>
</tbody>
</table>

\(^{34}\) Organizational Modeling Environment (OME) tool, a Java based i*/NFR Framework meta-case modeling tool, available at: http://www.cs.toronto.edu/km/ome/
<table>
<thead>
<tr>
<th>Correlation link</th>
<th>Uses a textual correlation table</th>
</tr>
</thead>
</table>
| Contribution types | ++ strong positive satisficing  
| | -- strong negative satisficing  
| | + weak positive satisficing  
| | - weak negative satisficing  
| | Make, Break  
| | Hurt, Help  
| | Some+, Some-  
| | Make  
| | Help  
| | Some positive  
| | Some negative  
| | Hurt  
| | Break  
| And/Or | AND  
| | OR  
| Agent |  
| Role |  
| Position |  
| Collective |  

223
Generic Actor prefix (applicable to all Actor types)

Dependency (any intentional element as dependum)

Actor association

Association type

Actor creation

Use Case Map notation:

<table>
<thead>
<tr>
<th>Process Step</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>System Structure</td>
<td>![System structure]</td>
</tr>
<tr>
<td>Control Process</td>
<td>![Control process]</td>
</tr>
</tbody>
</table>
Other annotation types, and their initials, are allowed

Semiotic Agent notation:

<table>
<thead>
<tr>
<th>Semiotic Agent</th>
<th><img src="image" alt="Semiotic agent" /></th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbolic Semiotic Agent</td>
<td><img src="image" alt="Symbolic Semiotic agent" /></td>
</tr>
<tr>
<td>Role</td>
<td><img src="image" alt="Role" /></td>
</tr>
<tr>
<td>Affordance</td>
<td><img src="image" alt="Affordance" /></td>
</tr>
<tr>
<td>Symbolic affordance</td>
<td><img src="image" alt="Symbolic affordance" /></td>
</tr>
<tr>
<td>Ontological dependence Link</td>
<td><img src="image" alt="Ontological dependence Link" /></td>
</tr>
<tr>
<td>Symbolic dependence link</td>
<td><img src="image" alt="Symbolic dependence link" /></td>
</tr>
<tr>
<td>Modifies link</td>
<td><img src="image" alt="Modifies link" /></td>
</tr>
</tbody>
</table>
11. References


Microsoft Corporation.


