MODELLING STRATEGIC RELATIONSHIPS FOR PROCESS REENGINEERING

by

Eric Siu-Kwong Yu

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Eric Siu-Kwong Yu
Modelling Strategic Relationships for Process Reengineering
A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy, Graduate Department
of Computer Science, University of Toronto, 1995

Abstract

Existing models for describing a process (such as a business process or a software development process) tend to focus on the "what" or the "how" of the process. For example, a health insurance claim process would typically be described in terms of a number of steps for assessing and approving a claim. In trying to improve or *redesign* a process, however, one also needs to have an understanding of the "why" – for example, why do physicians submit treatment plans to insurance companies before giving treatment? and why do claims managers seek medical opinions when assessing treatment plans? An understanding of the motivations and interests of process participants is often crucial to the successful redesign of processes.

This thesis proposes a modelling framework ***i****** (pronounced *i-star*) consisting of two modelling components. The *Strategic Dependency* (SD) model describes a process in terms of *intentional dependency relationships* among agents. Agents depend on each other for *goals* to be achieved, *tasks* to be performed, and *resources* to be furnished. Agents are *intentional* in that they have desires and wants, and *strategic* in that they are concerned about opportunities and vulnerabilities. The *Strategic Rationale* (SR) model describes the issues and concerns that agents have about existing processes and proposed alternatives, and how they might be addressed, in terms of a network of *means-ends relationships*. An agent's *routines* for carrying out a process can be analyzed for their *ability, workability, viability* and *believability*. *Means-ends rules* are used to suggest *methods* for addressing issues, *related issues* to be raised, and *assumptions* to be challenged. The models are represented in the conceptual modelling language Telos. The modelling concepts are axiomatically characterized.

The utility of the framework is illustrated in each of four application areas: requirements engineering, business process reengineering, organizational impacts analysis, and software process modelling. Advantages of ***i****** over existing modelling techniques in each of these areas are described.

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor John Mylopoulos, for his guidance, advice, patience, and continuing interest in my work. Without his foresight and on-going encouragement to venture beyond the traditional subject matters of computing, this thesis would not have materialized. I have the greatest respect for him as a teacher, a mentor, a researcher, and a scholar.

I wish to thank the other members of my committee — Professors Hector Levesque, Mark Fox, Andrew Clement, Rick Holt, Dave Wortman, and Andy Mitchell — for carefully reviewing my thesis, and for their many useful suggestions and stimulating questions. I am very grateful to Professor Janis Bubenko, whose pioneering work in information modelling and organization modelling has inspired me with faith in my own work, for serving as my external appraiser on short notice. During the early stages of my research, Professor Fred Lochovsky was a constant source of guidance and encouragement. My heart-felt gratitude goes to him. Professors Bruce Ahlstrand, Agnes Meinhard, and Glenn Whyte introduced me to the fascinating field of organization theory, which provided a crucial foundation for this work. I owe them my great appreciation and gratitude.

Numerous fellow graduate students have contributed to this work through their encouragement, criticisms, and discussions. Lawrence Chung and Brian Nixon helped me clarify and sharpen my research by critiquing numerous versions of technical papers, and by providing support and encouragement in many other ways. Other fellow members of the research group led by Professor Mylopoulos have also offered invaluable help. Gerhard Lakemayer, Jim Desrivieres, Yves Lesperance, and Bryan Kramer, among others, shared their expertise and insight over many hours of discussions. Many thanks to David Mitchell, Mike Gruninger, Chris Beck, Spiros Mancoridis, Kevin Schlueter, Dimitris Plexousakis, and Brian Nixon, who offered many useful suggestions and stimulating questions during my preparation for the oral defence.

The interest that Professor Eric Dubois, Philippe Du Bois, and Sylvie Nigot have shown in my work has been a great confidence booster. Thanks for initiating the joint effort to link our research. I would like to express my special thanks to Carolyn Seaman, Lionel Briand, Walcélio Melo, and Professor Victor Basili, who deemed the Strategic Dependency model to be of sufficient interest to test it out on a large-scale software maintenance organization, and for sharing the experiences and insights from that study. Their report provides invaluable early feedback on the practical utility of this work. I would also like to thank Jeanette Blomberg for kindly discussing experiences with the Trillium case study.

Financial assistance was gratefully received from the Natural Sciences and Engineering Research Council of Canada, and from Bell-Northern Research of Ottawa.

Finally, I thank my family and friends for their love and support. My parents have unfailingly extended their support whenever I was in need. To Alice, who offered help in every possible way, I owe my deepest gratitude.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

> *Process* n., *A series of actions, changes, or functions that bring about an end or result.* Webster's Dictionary.

As computers become more pervasive in everyday life, computer-executed processes are increasingly being embedded in, and intertwined with, the many processes that humans engage in. For example, in the domain of health care, the processes of providing clinical treatment, of processing insurance claims, and of dispensing medication, are increasingly a mixture of human and computer-executed processes. When computers are introduced to automate or support certain parts of human processes, it is usually with the intent that the overall process would be improved, e.g., with faster service, lower cost, better manageability, and so forth. Information technology is an important component of many current initiatives to control health care costs while maintaining the quality of care. The availability of new technology often makes possible the redesign of processes leading to substantial improvements.

When attempting to redesign a process, there are usually many alternatives, each with different implications for the many parties (stakeholders) that may have an interest in the process. To identify, evaluate, and select process alternatives that can address many issues and concerns is a considerable challenge. A systematic, engineering approach that employs appropriate models, analytical techniques, and known design methods would facilitate the task of process improvement and redesign, increase the chances of success, and potentially lead to more effective technical systems by establishing clearer links between process design decisions and technical system alternatives. We use the term *reengineering*[1] to emphasize that process improvement usually involves a pre-existing process, so that descriptive models are required.

Existing process models typically describe a process in terms of activity steps and entity flows. For example, consider a health insurance domain in which medical costs are covered by an insurance company in return for premium payments. Treatment by a physician must be pre-approved for a physician to receive reimbursement. A claims manager issues approval by verifying that the patient's policy is applicable to the medical condition, and by confirming that the treatment plan is appropriate according to medical opinion. This kind of understanding is often depicted informally in a "work flow" diagram such as in Figure 1.1. In computing science, various types of models are used or have been proposed for modelling different kinds of processes.

[1]The term *reengineering* is usually used in a more specialized context, as in business process reengineering. In this thesis, we use the term *process reengineering* in a generic sense, to refer to the use of an engineering approach to the modelling, analysis and redesign of processes.

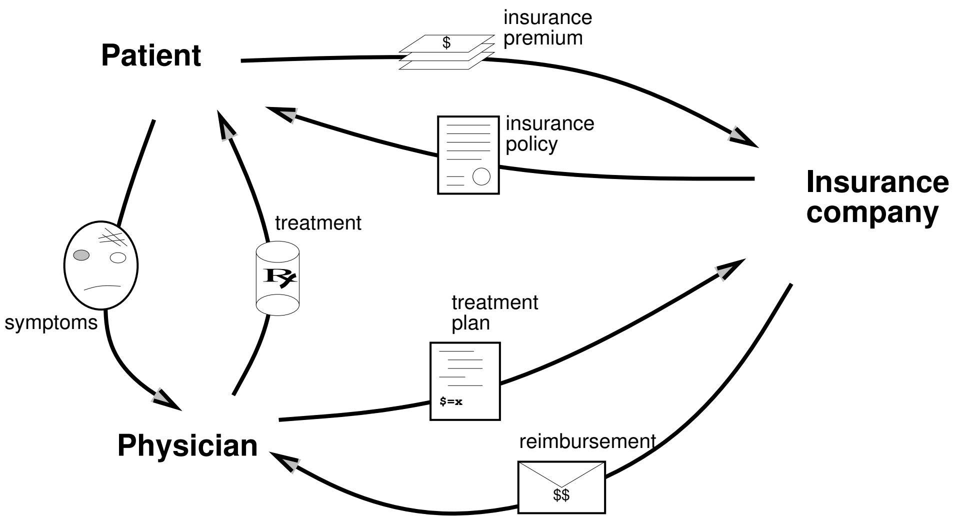

Figure 1.1: A "work-flow" model for a health insurance example

For example, SADT activity diagrams (Figure 1.2) and variations are widely used for systems analysis in information systems development [Ross77], while Petri net-based process models are in common use for a variety of purposes.

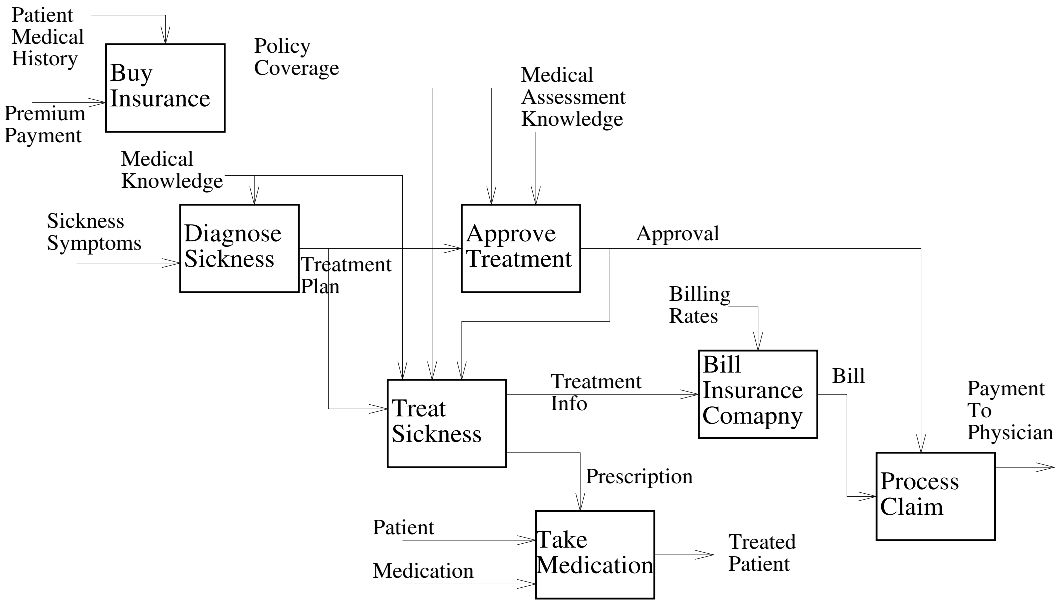

Figure 1.2: An SADT Activity Diagram for a health insurance example

However, in trying to understand an organization, the kinds of knowledge captured in these well-known types of models are often not enough. Most existing models provide a description of the "whats" in a process, but not the "whys". To have a deeper understanding about the processes depicted in Figure 1.1 and Figure 1.2, one needs to be able to answer "why" questions such as:

- Why do patients pay insurance premiums?

- Why do physicians submit treatment plans to insurance companies?
- Why do claims managers seek medical opinions on treatment plans?

If one is familiar with the health care domain, one would know that patients pay insurance *because* they want their medical expenses to be covered in case of sickness or injury. Physicians submit treatment plans to insurance companies for approval *because* they want to be reimbursed for giving the treatment. Claims managers seek medical assessment on treatment plans *because* they want to prevent unnecessary treatments, in order to control costs.

This kind of deeper understanding about the "whys" constitutes an important part of the knowledge about a domain, yet such "whys" cannot be expressed in conventional models that only allow descriptions of *entities* and *activities* in an organization and their interrelationships. We need a richer ontology that recognizes motivations, intent, rationales beneath the surface features of a process.

This work begins with the premise that a deeper understanding of processes can be obtained by taking an *intentional, strategic* view. In this view, the central unit to be modelled is the intentional, strategic *actor*. An *intentional* actor does not simply carry out activities and produce entities, but has motivations, intents, and rationales behind its actions. The intentional aspects of an actor can be characterized by such properties as goals, beliefs, ability, and commitment. An actor is *strategic* when it is not merely focused on meeting its immediate goal, but is concerned about longer term implications of its structural relationships with other actors, e.g., the opportunities and vulnerabilities that are present in a given configuration of relationships. A process needs to be modelled as a configuration of relationships among intentional, strategic actors. Process *reengineering* involves the exploration and selection of new configurations of relationships. Actors are concerned about the strategic implications of these new configurations. A strategic view also implies that the modelling framework should allow a high degree of incompleteness. Details that are not pertinent to the evaluation of alternative process designs can be omitted. The process model should acknowledge that actors may have the freedom (and the ability) to deal with problems as they arise during process execution, so that not all activities need to be specified ahead of time, during process design.

1.2 Research Objective and Approach

Objective. The objective of this research is to develop a richer conceptual framework for modelling processes which involve multiple participants (including both human and computer), so that one can have a more systematic reengineering of processes. This includes: identifying a set of modelling features for describing processes and the rationales behind them; providing formal representational constructs and their semantics; identifying some analytical capabilities; and outlining a design framework.

Approach. The approach taken is to first identify relevant concepts for modelling processes in organizational settings, drawing on the literature in studies of organizations.[2] Representational constructs for the concepts are then developed by building on existing conceptual modelling frameworks, making use of available knowledge structuring mechanisms. The semantics of the concepts are characterized by adapting formal techniques used for modellng agents in artificial intelligence. The framework is validated by applying it to several areas, and demonstrating that it advances the state-of-the-art in the process modelling and reengineering techniques described in the literature in each of these areas.

[2] Theories of organization are surveyed in, for example, [Morgan86] and [Scott87].

1.3 Application Areas

A richer framework for process modelling and reengineering would be applicable to a number of areas in computing. In this thesis, we apply the proposed framework to four areas and compare it to existing frameworks in these areas.

Requirements Engineering.

In Software Engineering, it is well recognized that capturing requirements that truly reflect users' needs is crucial to the success of a system development effort. A major obstacle to getting the requirements right is the difficulty in obtaining a deep enough understanding about the application domain (e.g., [Curtis88]). Decisions in technical system development need to be related systematically to this understanding.

During the early stages of requirements engineering, it is often necessary to help users identify different ways in which technical system solutions can serve their needs. Current requirements models that describe an organizational environment only in terms of entities and activities do not capture the many concerns that users have about the implications of adopting one solution versus another.

A richer model of the organizational environment would facilitate the requirements engineering effort. With motivations and rationales explicitly captured in a requirements model, it would also be easier to evolve a system to meet changing user needs, reducing the "legacy system" problem. Having better requirements would also lead to better, faster design and implementation of the software system.

Business Process Reengineering.

In recent years, the business community has become increasingly aware that information technology should not be used merely to automate existing business processes, but should be used as a basis for reshaping these processes to meet broader business objectives [Hammer90] [Davenport90] [Venkatraman91]. Most information systems are intended to improve organizational effectiveness in some way. For example, a health insurance system might be expected to speed up the approval of treatments, reduce the cost of policy administration, and improve the cost-effectiveness of medical coverage. Reengineering emphasizes the need to examine the surrounding business context of information systems.

A central tenet of reengineering is the need to ask "Why?" questions [Hammer90]. Without a clear understanding of the rationales behind existing practices and structures, one could not easily decide what changes could be made to business processes. By discovering underlying reasons, one can more readily identify outdated practices, and replace them with information technology systems and work arrangements that reflect new realities.

It is generally acknowledged that the practice of reengineering is still more art than science, and results are often unpredictable [Keen91] [Davenport93]. The predominant types of models used are variations of work flow or activity models similar to Figure 1.1 and Figure 1.2. A richer model that explicitly captures motivations and rationales would provide a more systematic framework for reengineering efforts, with better linkages to system development.

Organizational Impacts Analysis.

The success of a computing system depends on a great many factors that go beyond the technical system, but have to do with the surrounding (human) organizational environment (see, e.g., [Lyytinen87]). The stakeholders of an information system may have a wide-ranging set of concerns about an information system and how it might alter the work environment. For example, along with the introduction of a new information system, there may be changes in work roles and in skill levels and opportunities for advancement; jobs may be enriched or extra burdens may be imposed; there may be changes to the power structure (e.g., when there is competition for resources, who is more likely to win or lose); there may be increased social pressure and discipline, or of potential conflict; individual freedom or privacy may be curtailed [Attewell84].

Conventional models used in system development have been designed primarily for describing technical systems, and do not provide rich enough descriptions of human social organizations. A process model which can take some of these richer organizational issues into account would facilitate greater attention to these issues during system development efforts, and would potentially allow organizational analysis to be better integrated into system development processes.

Software Process Modelling.

Process modelling is also important for modelling and improving software processes. Recently, there has been a great deal of effort to develop software process models (e.g., [ISPW93] [ICSP93]). Many of these are designed for incorporation into computer-based software development environments (SDEs) so that process steps could be enacted automatically or semi-automatically.

However, to achieve successful development of high quality software, it is just as important to pay attention to the organizational environment (such as project teams) as to the technical support tools. A software process model that captures the motivations and rationales that lie behind the activities and flows in a software project would facilitate the systematic analysis and design of software processes, including the use of support tools (SDEs) within a human organizational context.

1.4 Framework Overview

The framework is called $\boldsymbol{i}^*$, as it attempts to articulate a notion of "distributed intentionality". It consists of two models: a **Strategic Dependency (SD)** model for describing a particular configuration of dependency relationships among organizational actors, and a **Strategic Rationale (SR)** model for describing the rationales that actors have about adopting one configuration or another.

1.4.1 The Strategic Dependency Model

The Strategic Dependency (SD) model is a network of dependency relationships among actors. The intuitive meaning of a dependency is that a depender, by depending on someone else (the dependee) for something (the dependum), can accomplish some goal or objective that it would otherwise be unable to achieve (or not as well). If the dependum is not forthcoming from the dependee, the depender would suffer as a result, i.e., its attempt to accomplish the objective may fail or may be compromised.

The SD model therefore aims to capture the intentional structure of a process, instead of the usual non-intentional, and non-strategic process models of activities and entities. It is a higher level characterization of a process because it captures what *matters* to the actors, while leaving out non-essential details.

The model distinguishes among several types of dependencies based on how agents constrain each others freedoms, and the extent to which they are vulnerable in their dependencies. Dependencies are threaded through *roles* and *positions*, as well as physical *agents*, creating an intricate web of relationships.

Analysis. The SD model supports analysis of who depends on whom for what, directly or indirectly. One can use the model to explore opportunities that are open to each actor, by matching the dependums that dependees offer and those that dependers want. One can analyze vulnerability, by tracing chains of dependencies. How far down a chain one might be concerned about vulnerability is based on a distinction of the degree of dependency into three categories – open, committed, or critical. Using the SD model, one can identify who are the stakeholders, and what are their stakes. To validate a model, one can compare answers to various types of queries to see if they agree with what is expected intuitively.

1.4.2 The Strategic Rationale Model

In the Strategic Rationale (SR) model, the rationales behind process configurations can be explicitly described, in terms of process elements and relationships among them. The main types of relationships are represented as means-ends links and task-decomposition links. Means-ends links are seen as applications of generic rules in particular contexts. Process elements include subgoals, subtasks, resources, and softgoals. The model is strategic in that elements are included only if they are considered important enough to affect the achievement of some goal. Agents may be able to accomplish something by themselves, or by depending on other agents. An interconnected collection of process elements serving some purpose for an agent is called a routine. An agent often has more than one routine for accomplishing something. Process reengineering involves modelling existing routines (e.g., by asking "why" and "how" questions) and discovering new and better routines.

Analysis. Beyond basic queries about nodes and links, the SR model offers four levels of analysis at a more aggregate level. An actor has the *ability* to accomplish something if it has a routine for it ("knowing how"). Next, one can check if the routine is *workable*, i.e., whether it is reducible to workable elements, through task-decomposition and means-ends links, or workable dependencies. Thirdly, one can check if a routine is *viable* with respect to desired qualitative criteria. Finally, one can check whether the assumptions involved in reasoning about the routine are *believable*, i.e., sufficiently justified.

Design. The framework provides support for *raising* issues, *addressing* them, identifying *correlated* issues, identifying *assumptions* and justifying them, and *settling* issues and accepting assumptions.

The framework is intended to provide interactive support for an argumentative style of reasoning, not to fully automate the reasoning. It is assumed that the type of strategic reasoning being supported is largely a judgemental, iterative process, frequently based on incomplete knowledge. The aim of the framework is to provide modelling features, which can lead to semi-automated support facilities to help human users express, manipulate, organize, manage, and draw conclusions from this knowledge.

1.5 Related Work

The approach taken in this work focuses on strategic relationships among intentional agents in open, distributed, and evolving organizational settings. The framework provides features to help describe processes ("modelling") and to guide change ("reengineering"). There is no existing framework that is directly comparable in aim and in approach to the proposed framework in its entirety. However, individual parts of the framework can be compared to existing work in various research areas. In this section, we give an overview of related work. More detailed comparisons are given in the discussion sections in each of the application chapters (4 to 7).

1.5.1 Process modelling.

Process modelling has been of interest to computing in a number of different areas, although the terminology, conception, and purpose of process modelling varies.

In information systems development, the "systems analysis" phase usually begins with an attempt to understand the work process within which the intended system is to be embedded. The dominant technique has been Structured Analysis (e.g., Data Flow Diagrams (DFDs) [DeMarco78], or Structured Analysis and Design Technique (SADT) [Ross77]). A process is a collection of activities interconnected by inputs and outputs (and controls and mechanisms, in SADT). The central abstraction mechanism is composition/decomposition. Advances in this area have introduced additional knowledge structuring mechanisms such as classification/instantiation, generalization/specialization, and time (e.g., [Greenspan84]). Some of these features have been incorporated in the recent trend towards object-oriented analysis (e.g., [Coad93]). In most techniques for systems analysis, organizational processes (involving humans) are treated much like system processes. There are no attempts to address a richer organizational environment.

A different line of research focuses on the application of information systems technologies to organizational or "office" work settings, where the systems have to work within richer, less well-defined work processes than traditional information systems (such as those oriented towards transaction processing). Office or organizational work settings [Fikes80][Hewitt86][Bracchi84] tend to be more:

- *open* – less structured or rigid, more unpredictable and open-ended,
- *distributed* – occuring over multiple locations or work units, involving multiple parties, and
- *evolving* – constantly changing, demanding greater flexibility and adaptability.

Although these characteristics of organizational environment were identified, most of the process models developed in this research area have focused on the technical systems that support work, rather than on understanding and redesigning the work processes. Process models included those that use Petri-nets for sequencing tasks (e.g., [Zisman78]) and adaptations of problem-solving concepts from artificial intelligence (e.g., [Barber82] [Croft88] [Woo88] [deJong89]). Knowledge-based techniques were also recognized as crucial for supporting organizational work (e.g., [Lochovsky83] [Sathi85] [Tueni88]).

As different types of computing systems are increasingly intermixed and cooperating to provide support in modern work settings, the more complex organizational issues raised by researchers in this area need to be addressed. For example, customer service representatives answering queries on the status of customer orders will likely be accessing transaction-oriented databases (or advanced information repositories), but also using various types of groupware, workflow, and problem-solving support systems.

These rich organizational issues presuppose an understanding of organizational participants as intentional actors. This dimension has been missing from processes models used in system analysis, and in organizational information systems. In artificial intelligence, models have been developed for the specification of agent behaviour in terms of operators for intentional concepts such as belief, goal, ability, and commitment. (e.g., [Cohen90] [Thomas91]). However, these modelling concepts were not conceived originally for process modelling in organizational settings, and require adaptation for use in $\boldsymbol{i}^*$. [Castelfranchi92] also proposed dependence concepts based on intentional concepts.

1.5.2 Process rationale.

While there has been many models in Computer Science which could be viewed as process models, there has been hardly any models for reasoning about processes and to support their redesign.

The closest work in this regard is work in design rationale frameworks, which have been developed for supporting software engineering (e.g., [Potts88][Conklin91]) and other design contexts (e.g., [MacLean91]). These frameworks have evolved from frameworks for argumentation and issue-based information systems [Conklin88], and are largely informal. A goal-oriented approach to design rationale was developed by Lee [Lee92]. A framework for addressing non-functional goals was developed by Chung [Mylopoulos92] [Chung93] [Chung94a] [Chung94b] for dealing with non-functional requirements during software development. A notion of satisficing is used to deal with soft concepts. The above frameworks are not aimed at reasoning about processes specifically, and do not make use of a special ontology for modelling processes.

1.5.3 Requirements Engineering.

The field of requirements engineering emerged as a subarea of software engineering, aiming to make the requirements phase of software system development more systematic and disciplined. In the research community, it has become well recognized that the requirements phase should be concerned with the relationships between systems and their environment, rather than stating requirements only in terms of the systems [Bubenko80] [Zave81] [Jackson83]. Formal requirements modelling languages (e.g., RML [Greenspan82,84,94], ERAE [Dubois86]) were developed to address problems of ambiguity, imprecision, and to help manage the large amount of knowledge that are covered in informal or semi-formal requirements (e.g., graphical notations such as SADT). These languages did not attempt to deal with a richer, organizational notion of process.

Recently, there has been recognition of the need to extend requirements modelling to address business goals and to capture the rationales behind information systems development [Bubenko91]. In the enterprise modelling framework of [Bubenko93], the need to understand "why", and to deal with fuzzy, informal, and non-functional issues are emphasized. The Enterprise Model consists of five interconnected sub-models – an Objectives Model, a Concept Model, an Activity and Usage Model, an Actors Model, and an Information System Requirements Model. The modelling constructs are made up of structured linkages among nodes whose contents are informal text. The modelling process is assumed to be issue-driven and cuts across the various sub-models.

Another line of work in requirements engineering has focused on the systematic development of requirements from higher goals [Feather87] [Dubois89] [Fickas92] [Dardenne93] [Feather94]. Software systems are viewed as one type of agent in a larger global system including humans and hardware components as well. The overall system is designed by systematically addressing (reducing) global goals, while making sure constraints are not violated. The final design consists of assignment of responsibilities (for actions) to agents. This line of work is consistent with the intent of $\boldsymbol{i}^*$ as a framework for systematic process reengineering. However, goal-oriented requirements engineering frameworks have so far not addressed the more open notion of process, but aim instead to arrive at tightly constrained relationships between agents (in terms of actions) as the end product of requirements engineering. There is no attempt to reason about strategic relationships among agents. The requirements process is thought of as the design (from scratch) of an overall system with global goals that need to be satisfied, rather than the on-going evolution (and hence redesign or reengineering) of an open, distributed, organizational environment with possibly divergent and competing goals and interests.

1.5.4 Business process reengineering.

Process modelling is considered an essential step in BPR. Process models are used to describe existing processes, understand their deficiencies, and to propose new process configurations. However, the types of models used are usually informal – for example, graphical flow charts depicting the flow of work products from one work unit to the next (workflow models). Semi-formal models (such as SADT, more commonly refered to as IDEF0 in the BPR context) which impose some discipline are considered advanced modelling techniques. Despite a recognized need for "understanding why", models in BPR do not yet offer specific features to support such understanding. Process design is based on case studies (success stories) and benchmarks (how things are done in other companies or industries, with statistical metrics – how many days it takes to process a customer request, handled by how many persons, error rates). Although generic knowledge in the form of rules-of-thumb exists, they can only serve as rough guidelines since their applicability to specific settings are not always clear.

A more systematic approach has been proposed in [Malone93] and [Lee93] using a goal-based approach. This is an application of the framework of [Lee92] to process reengineering. In the Action-Workflow model presented in [Medina-Mora92] (which is based on the earlier work of [Winograd86]), each process step is modelled in terms of a four-phased loop – proposing, agreeing, performing, and accepting – between a customer and a performer. This pairing of customers and performers suggests the presence of an intentional relationship, which is somewhat comparable to the depender–dependee relationship in $\boldsymbol{i}^*$. However the intentional semantics is not made explicit and there is no formal characterization.

1.5.5 Organizational impacts of computing.

The literature in this area offers perhaps the richest descriptions of organizational processes and theories for understanding and explaining them. For example [Keen81] explores the types of resistance to process change brought on by the introduction of information systems. [Markus83] explores the issues of power in systems implementation. Most work in this area draws on theories of organization, or more directly from source disciplines in the social sciences. [Lyytinen87] provides a survey of information systems problems and issues that cover the broader perspectives on organizational impact. These issues are seldom addressed in more technical frameworks, such as those for systems analysis and requirements engineering, but are often crucial to the success of a system.

In this research area, the format of process description is usually discursive text. The reasoning or interpretations used to explain processes or process change are typically given in argumentative prose. While natural language perhaps have the greatest flexibility and power for imparting a rich picture on the reader, it requires a considerable leap to bring the conclusions of such studies to bear on technical system design, which rely on models for conciseness and precision. Nevertheless, this research area offers a rich source for developing deeper understanding about computer-based systems and their organizational environment, and thus concepts for incorporation into formal models for such understanding.

March and Simon [March58] offered a theory of organization based on the concept of bounded rationality. Simon's concept of satisficing was used in [Chung93][Mylopoulos92] and adapted in $\boldsymbol{i}^*$ for dealing with soft concepts. Simon's work on means-ends reasoning provided much of the ground work for subsequent developments in artificial intelligence. The notion of strategic actors is common among theories of organization from a political perspective (e.g., [Crozier64] [Hickson71]). Dependence among actors has also been theorized by a number of writers (e.g., [Emerson62] [Thompson67] [Pfeffer78] [Malone90] [Rockart91]). Commitment has been explored (e.g., [Becker60] [Gerson76] [Winograd87]). Other conceptual frameworks that have been helpful in developing $\boldsymbol{i}^*$ include the "web" model of [Kling82] and the notion of boundaries in [Kling87]. Because of the multiplicity and complexity of organizational issues, many authors have advocated the need for multiple perspectives when trying to understand organizational phenomena [Morgan86]. [Scott87] classifies organizational theory perspectives broadly into "rational", "natural" and "open systems" perspectives. The $\boldsymbol{i}^*$ framework represents a small step in attempting to bring some of these concepts into a formal modelling framework for assisting work process understanding and redesign, in a representation that is more readily usable within a systematic systems development process.

Ethnographic studies offer an especially fine-grained understanding of organizational phenomena. [Suchman83] and [Suchman84] provided in-depth view of accounting work practice. [Blomberg86] offered insights on the organizational evolution of a design team as a new design tool was introduced. Some of the process configurations and reasoning from this case study are re-expressed using $\boldsymbol{i}^*$ to demonstrate the extent to which some of these organizational issues can be expressed by the framework (Chapter 6).

1.5.6 Software process modelling.

A great many process models have been proposed for various purposes in the software process modelling area [Curtis92]. They span the spectrum from semi-formal models for understanding (e.g., SADT-type models) to detailed formalisms suitable for execution (e.g., models using Petri-nets and extensions [Deiters90] [Bandinelli93], rules and triggers [Kaiser90], and plans [Huff88] [Mi93]).

None of these modelling frameworks address the organizational aspect of software processes from the viewpoint of strategic actors and intentional relationships. Furthermore, there is no modelling framework to support the systematic design or redesign of a software process. For example, the SEI Capability Maturity Model [Humphrey89] provides a five-level maturity gradation for assessing processes in software development organizations. It provides guidelines for achieving these levels (presented as itemized text), but does not provide a systematic framework for making the tradeoffs and understanding implications, which are needed when designing specific processes in particular organizations.

1.6 Thesis Organization

Chapters 2 and 3 present the two models of the $\boldsymbol{i}^*$ framework. The modelling features are illustrated with examples from a health insurance domain. The reengineering aspects of the framework are illustrated with (rudimentary) examples drawn from a health care reform setting. The representational constructs and axiomatic characterization of the modelling concepts are presented.

Chapters 4 to 7 present the application of the $\boldsymbol{i}^*$ framework to four areas of interest to computing science, and serve to demonstrate the expressiveness and utility of the framework. Each chapter begins with an introduction to some of the issues in the area, followed by a synopsis of how $\boldsymbol{i}^*$ contributes to addressing those issues. The application of the Strategic Dependency and Strategic Rationale models are then illustrated using a representative example problem. A discussion section compares in greater detail the approach offered by $\boldsymbol{i}^*$ with existing techniques in the area.

Chapter 4 makes use of an example concerning the requirements engineering of a computer-based meeting scheduler. This example has been adopted by a number of researchers in this community for comparing different approaches to requirements engineering. The process involved in acquiring goods in an organization is often used to illustrate the concept of business process reengineering. We use this example setting in Chapter 5 to demonstrate how ***i**** can be used to support BPR.

In Chapter 6, we make use of a case study that was reported in the literature about how a design team evolved in response to the introduction of a new computer-based design tool. We re-express in ***i**** the informal description of the organizational configurations and the explanation of the forces behind their evolution. In Chapter 7, we show how ***i**** can be used to reason about the rich organizational contexts of software processes, and to help in their systematic design or redesign, filling a need that is unaddressed by existing models in the area.

Chapter 8 concludes this thesis by summarizing the results and contributions of this work, and identifies some avenues for future research.

Chapter 2

The Strategic Dependency (SD) Model

The Strategic Dependency (SD) model provides an intentional description of a process in terms of a network of dependency relationships among actors. An intentional process model aims to capture the underlying motivations and intents behind the overt activities and flows in a process. The intentional approach to process modelling acknowledges that actors have freedom of action, within the bounds of social (inter-actor) constraints. Because of its richer modelling concepts, the model provides a better basis for an analyst to explore the broader implications of a process than conventional, non-intentional models. The model can help identify stakeholders, analyze opportunities and vulnerabilities, and recognize patterns of relationships, such as various mechanisms for mitigating vulnerability. The SD model is axiomatically characterized by adapting intentional concepts (such as goal, belief, ability, and commitment) developed for modelling agents in artificial intelligence.

Section 2.1 presents the features of the SD model informally by way of examples. Section 2.2 describes how the model is embedded in the knowledge representation language Telos. Section 2.3 discusses the modelling and analytical capabilities of the model, and includes an extended example comparing three alternative configurations of health care provisioning. Section 2.4 presents the formal characterization of the model.

2.1 Modelling Features

A Strategic Dependency (SD) model consists of a set of **nodes** and **links**. Each node represents an "actor", and each link between two actors indicates that one actor depends on the other for something in order that the former may attain some goal. We call the depending actor the **depender**, and the actor who is depended upon the **dependee**. The object around which the dependency relationship centres is called the **dependum**.

An **actor** is an active entity that carries out actions to achieve goals by exercising its knowhow. A dependency is intentional if the dependum is somehow related to some goals or desires of the depender. By depending on another actor for a dependum, an actor (the depender) is *able* to achieve goals that it was not able to do without the dependency, or not as easily or as well. At the same time, the depender becomes *vulnerable*. If the dependee fails to deliver the dependum, the depender would be adversely affected in its ability to achieve its goals. In the SD model, the depender's internal goals and desires are not explicitly modelled.

Figure 2.1 shows a Strategic Dependency model of a health care domain. It presents (some of

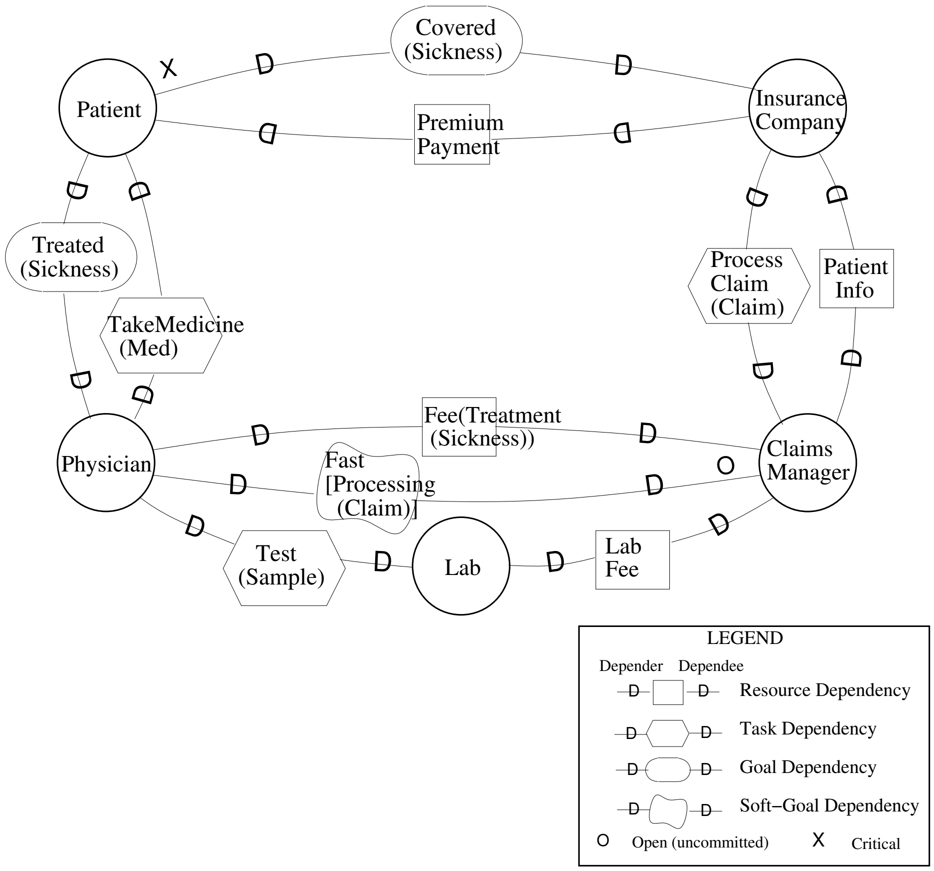

Figure 2.1: Example of a Strategic Dependency model from the health care domain

the) relationships among patients, physicians, labs, and insurance companies. Patients depend on physicians for treatment, while physicians depend on patients to take medication, and on laboratories to perform tests. Physicians and labs depend on fee payments from claims managers in insurance companies. Insurance companies depend on their claims managers to process claims. Claims managers depend on their companies to provide patient information. Insurance companies depend on patients to pay premiums. And patients depend on insurance companies to cover them in the event of sickness.

Dependency Types. We distinguish among four types of dependencies, based on the type of the dependum. In world modelling, it has been found useful to distinguish among three basic ontological categories: entities, activities, and assertions [Greenspan84]. Entities are used to model objects in the world. These can be physical or informational. Activities produce changes in the world. An assertion expresses a state or condition about the world. From these basic categories, we get three types of intentional dependencies: Resource dependency, Task dependency, and Goal dependency. A fourth type, which we call “Softgoal dependency”, is based on a notion of *non-functional* requirements (or quality requirements) in software engineering [Chung91,93,Mylopoulos92].

In a **Goal-dependency**, the depender depends on the dependee to bring about a certain state in the world. The dependee is given the freedom to choose how to do it. With a goal dependency, the depender gains the ability to assume that the condition or state of the world will hold, but becomes vulnerable since the dependee may fail to bring about that condition.

When a patient depends on a physician to have his sickness cured (Figure 2.2(a)), it is usually up to the physician to decide how to treat the patient. The patient is only concerned about the outcome. This would be appropriately modelled as a goal dependency. When a physician depends on an intensive-care unit nurse to keep a patient's blood pressure normal, this would also be a goal dependency. The intensive-care nurse is trained to give the appropriate types of injections to maintain normal blood pressure. Leaving one's car with a repair shop saying "just get it fixed" is another example of a goal dependency.

In a **Task dependency**, the depender depends on the dependee to carry out an activity. A task dependency specifies *how* the task is to be performed, but not *why*. The depender is vulnerable since the dependee may fail to perform the task.

When a physician asks her patient to take medication (e.g. four times a day, two hours after meals), she is depending on the patient to perform a task. If the patient does not take the medication (in the way specified), the physician's ability to cure the patient would be impaired. A physician depending on a lab to perform clinical test by giving specific instructions would be modelled as a task dependency (Figure 2.2(b)). A carpenter's dependence on his apprentice's to run his errands (without explaining why), a passenger telling a driver which route to take (without indicating the destination), a store manager asking a clerk to stock the shelves in a certain way, these are also examples of task dependency.

Task specifications should be viewed as constraints rather than as the complete (and therefore adequate) knowhow for performing the task. This is one reason why a dependee may fail in performing the task. Another reason may be that the dependee decides not to perform the task, even when it is able to, e.g. if it decides there are more important things to do (which may be due to other commitments).

In a **Resource dependency**, one actor (the depender) depends on the other (the dependee) for the availability of an entity (physical or informational). By establishing this dependency, the depender gains the ability to use this entity as a resource. At the same time, the depender becomes vulnerable if the entity turns out to be unavailable.

The Policy Administration department's dependency on patients premium payments would be modelled as a resource dependency (Figure 2.2(c)). A carpenter's dependence on a tool from a tool shop, a driver's dependence on gasoline from a gas station, a retailer's dependence on information about a customer's credit worthiness from the credit card company, could all be modelled as resource dependencies.

A resource dependency is different from the usual notion of a "flow" in that the latter does not indicate the presence or absence of intentionality in the flow. A resource flow does not necessarily imply a resource dependency. For example, if a cashier would not suffer whether or not the customer's credit information was available, then that information is a mere flow, not a resource dependency.

The notion of goal dependency described earlier is based on a clear-cut, black-and-white notion of goal achievement. The world is either in the stated condition or not. Often, there are goals that are not that sharply defined, but are goals nonetheless. They are not clear-cut because their meaning is not objectively known. There is no prior agreement between depender and dependee about what constitutes the achievement of that goal. When this is the case, very often the goal is clarified during the process of trying to achieve the goal, by identifying alternatives, and the depender indicating which alternative to take. Often these alternatives are identified by the dependee, but the decision is taken by the depender. We call this a softgoal dependency.

In a **Softgoal dependency**, a depender depends on the dependee to perform some task that meets a softgoal. The meaning of the softgoal is specified in terms of the methods that

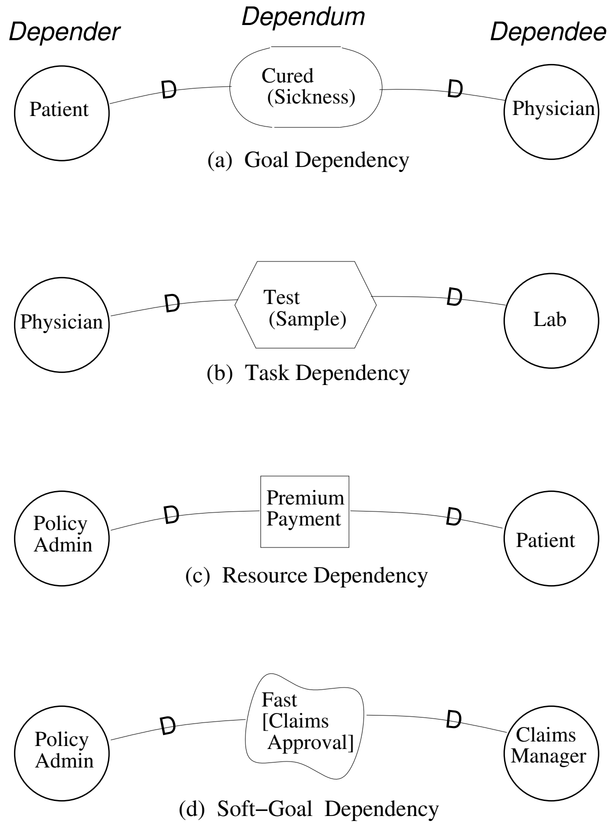

Figure 2.2: Dependency types

are chosen in the course of pursuing the goal. As in a goal dependency, a depender gains the ability of having the goal condition brought about, but becomes vulnerable in case the dependee fails to bring about that condition. The difference here is that the conditions to be attained are elaborated as the task is performed.

When a head nurse requires nurses in the ward to respond to patients *promptly*, the meaning of "promptly" is not clear-cut. A newcomer would have to find out in terms of what kinds of task should take priority. This is a softgoal dependency. In the insurance example, the policy administration department's dependency on claims managers for fast claims approval would typically be a softgoal dependency (Figure 2.2(d)).

These four types of dependencies also characterize how decisions fall on either side of the dependency. Under *goal dependency*, the dependee is free to, and is expected to, make whatever decisions are necessary to achieve the goal (the dependum). Under *task dependency*, the depender makes the decisions. The depender's goals are not given to the dependee. Under *resource dependency*, the issue of decisions does not come up. A resource is the finished product of some deliberation-action process. It is assumed that there are no open issues or decisions to be addressed. Under *softgoal dependency*, the depender makes the final decision, but does so with the benefit of the dependee's knowhow. The four types of dependencies thus indicate who will handle problems if and when they arise.

Dependency Strength. The model also distinguishes among several degrees of dependency. On the depender side, a stronger dependency means the depender is more vulnerable, and is likely to take stronger measures to mitigate vulnerability. On the dependee side, a stronger dependency implies that the dependee will make a greater effort in trying to deliver the dependum. The model provides for three degrees of strength: Open (uncommitted), Committed, and Critical. These apply independently on each side of a dependency. Graphically, we use an "O" for open, unmarked for committed, and "X" for critical. (Figure 2.3).

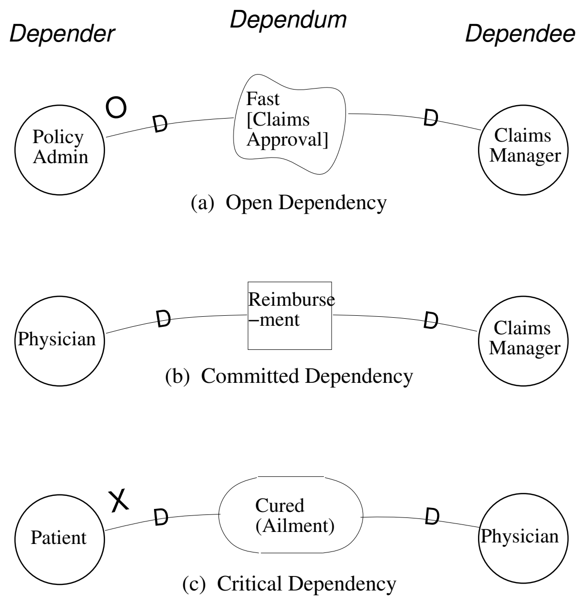

Figure 2.3: Three degrees of dependency strengths

In an **Open Dependency**, a depender would like to have the dependum goal achieved, task performed, or resource available, so that it could be used in some course of action. Failure to obtain the dependum would affect the depender's goals to some extent, but the consequences are not serious. On the dependee side, an open dependency is a claim by the dependee that it is able to achieve the dependum for some depender.

In a **Committed Dependency**, the depender has goals which would be significantly affected – in that some planned course of action would fail – if the dependum is not achieved. The depender might have invested considerably in a course of action which could not be reversed without loss. Because of its vulnerability, a committed depender would be concerned about the viability of the dependency. (The concept of viability is developed more fully in Chapter 3.) On the dependee side, a committed dependency means that the dependee will try its best to deliver the dependum, e.g., by making sure that its own dependencies are viable.

In a **Critical Dependency**, the depender has goals which would be seriously affected – in that all known courses of action would fail – if the dependum is not achieved. In this case, we assume that the depender would be concerned not only about the viability of this immediate dependency, but also about the viability of the dependee's dependencies, and the dependee's dependee's dependencies, and so forth.

Agents, Roles, and Positions. Actors in realistic social contexts have many dependencies on other actors as well as dependencies from other actors. Grouping and categorizing dependencies as belonging to sub-units of a social actor can serve as a way of modelling the "internal" structure of an actor, while still preserving the intentional actor abstraction provided by the SD model – i.e., modelling processes in terms of external relationships. Furthermore, a finer grouping of dependencies would help identify more precisely how one dependency might lead to other dependencies.

We use the term *actor* to refer generically to any unit to which intentional dependencies can be ascribed. To model the sub-units of a complex social actor, we define three types of sub-units – agents, roles, and positions – each of which is an actor in a more specialized sense.

A **role** is an abstract characterization of the behaviour of a social actor within some specialized context or domain of endeavour. 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 that it can be used to refer to human as well as artificial (hardware/software) agents. An agent has dependencies that apply regardless of what roles 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 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.

Roles, positions, and agents can each have subparts. 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.

2.2 Representational Constructs

The Strategic Dependency model is embedded into a formal conceptual modelling framework, so as to allow for the effective usage and management of the potentially large amounts of knowledge involved when modelling real-world processes. We have chosen to embed the concepts of the SD model into the conceptual modelling language Telos [Mylopoulos90]. In doing so, we obtain an object-oriented representational framework, with classification, generalization, aggregation, attribution, and time. The extensibility of Telos, due to its metaclass hierarchy and treatment of attributes as full-fledged objects, facilitates the embedding of new modelling features such as Strategic Dependency concepts.

The schema for the Strategic Dependency model is defined at the metaclass level in Telos (Figure 2.4). Domain classes such as `Physician` would be defined as instances of some metaclass, in this case `PositionClass`. Metaclasses are instances of metametaclasses. Thus `PositionClass` and `ActorClass` are both instances of the metametaclass `Class`. This metaclass facility in Telos allows the schema to be expressed within the same framework as domain objects.

An instance of `ActorClass` (e.g., `Physician`) can have as an attribute some instance of `DependsClass` (e.g., `FeeForTreatment`). This is used as the basic construct for representing strategic actor dependencies. The schema for this is defined by the links labelled `depends` (if the named actor is the depender) and `depended` (if the named actor is the dependee).

We make use of the Telos facility for allowing attributes on attributes to specify the other party in a dependency. Since the attribute class `DependsClass` is a full-fledged object, we can define an attribute `dependee` on it. The example shows `Physician` as having a dependee (`ClaimsManager`) on its dependum `FeeForTreatment`.

The four types of dependencies are defined as specializations on each of `DependsClass` and `DependedClass`. For brevity, Figure 2.4 only shows the specializations for Resource Depen-

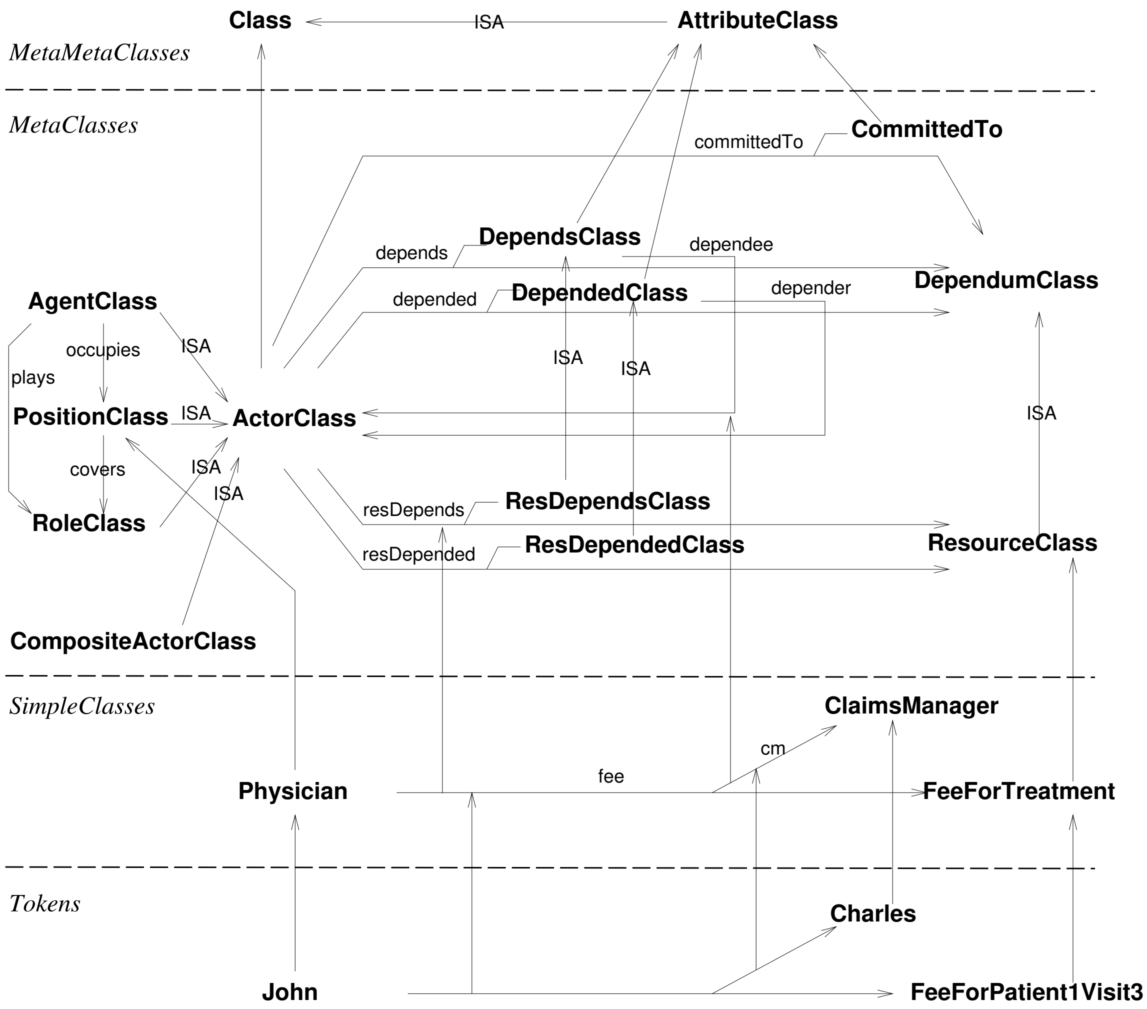

Figure 2.4: Partial schema for Strategic Dependency Model, with domain example

dency. Commitment is represented as another attribute on `ActorClass` with attribute value belonging to `DependumClass`. This can be used to qualify any dependency. Criticality is defined analogously.

A sample syntactic representation[1] for the `Physician` position in Telos is as follows: (For another example, see chapter 7 — software process modelling).

```
TELL Class Physician IN PositionClass
      ISA ProfessionalPosition WITH
  resDepends, committedTo
    fs:FeeForTreatment WITH
      dependee
        cm:ClaimsManager
      end
  goalDepended, commitsTo
    td:$Treated(p.injury)$ WITH
      depender
        p:Patient
      end
```

[1]A syntactic variant of Telos is used in this thesis for simpler presentation.

```
    taskDepends, committedTo
        tm:TakeMedication(p.med) WITH
            dependee
                p:Patient
            end
    covers
        tp:TreatingPatient(p)
        bi:Billing(p.insurCo)
    integrityConstraint
        correctClaimsManager:
            $cm=p.insurCo.claimsMgr$
end
```

Because Telos allows integrity constraints on any class, the semantics of the Strategic Dependency model can be incorporated and enforced by stating them as integrity constraints in the appropriate metaclasses, as illustrated in the following:

```
TELL MetaClass CommittedTo IN AttributeClass
    COMPONENT <ActorClass, committedTo, DependumClass, Alltime>
        WITH
    integrityConstraint
        committedDependerVulnerability:
            $(Forall C/CommittedTo)(Forall x/Individual)(Forall eta/Individual)
                (x in from(C) and eta in to(C)) ==>
                (Exists u/RoutineClass)(memberOf(u, x.routine) and
                      not Workable(x, eta) ==> not Workable(x, u)
end
```

Each Telos object is ultimately represented in terms of four components: source, label, destination, and time. The object `CommittedTo` has as source the object `ActorClass` and as destination `DependumClass`. The object `C`, being an instance of the `CommittedTo` class, has as source (referred to via the expression `from(C)`) an instance of `ActorClass` (e.g. `Physician` in Figure 2.4, and as destination (`to(C)`) an instance of `DependumClass` (e.g., `FeeForTreatment`). The integrity constraint applies to instances of `from(C)` and `to(C)`, called `x` and `eta` respectively in the formula. It says, if `x` and `eta` are related by `CommittedTo`, then some routine `u` would be unworkable for actor `x` if `eta` is unworkable for `x`.

2.3 Process Modelling and Analysis Using the Strategic Dependency Model

The Strategic Dependency model provides a set of concepts for modelling processes in terms of the intentional dependencies among actors. In this section, we discuss how the features of SD model can provide a deeper understanding about processes than existing, non-intentional process modelling frameworks. The modelling and analytical capabilities of the model are illustrated through an extended example comparing three alternative configurations of health care provisioning, based roughly on the three types of arrangements known as "full indemnity insurance", "managed indemnity insurance", and "managed care" [Loomis94].

Opportunity and Vulnerability. The set of nodes and links in an SD model form a dependency network. By following the *chains* of dependencies, one could explore the expanded possibilities that are accessible to an actor. From a vulnerability viewpoint, an actor could also use the dependency network to determine how it could be affected adversely by these dependencies.

By enlisting the help of dependees, a depender expands opportunities, and can achieve what would otherwise be unachievable. The patient in the example of Figure 2.1 is able to have his sickness treated, by depending on a physician. A patient typically does not have the ability to treat himself. A physician may not have the ability to do clinical tests all by himself. But he can get tests done by depending on a laboratory. Given an SD model of a process, one could ask: What new relationships among actors are possible? By matching the dependencies from dependers ("wants") and those from dependees ("abilities"), one can explore opportunities that are available to these actors. Classification and generalization hierarchies facilitate the matching of dependums. The "down side" of a dependency for a depender is that the depender becomes vulnerable to the failure of the dependency. A depender would be concerned about the *viability* of a dependency.

Health care configuration 1. In the configuration of Figure 2.5 ("full indemnity insurance"), a patient depends on his physician to have his sickness treated. The physician depends on a laboratory for clinical tests. Both the physician and the lab depend on an insurance claims manager to pay them for services rendered. The patient depends on the insurance company to provide this coverage in return for insurance premium paid. Patients want insurance premiums to be affordable, and to be able to receive treatment quickly when they get sick. Physicians want insurance companies to process claims quickly so they will receive payments promptly for services rendered to patients.

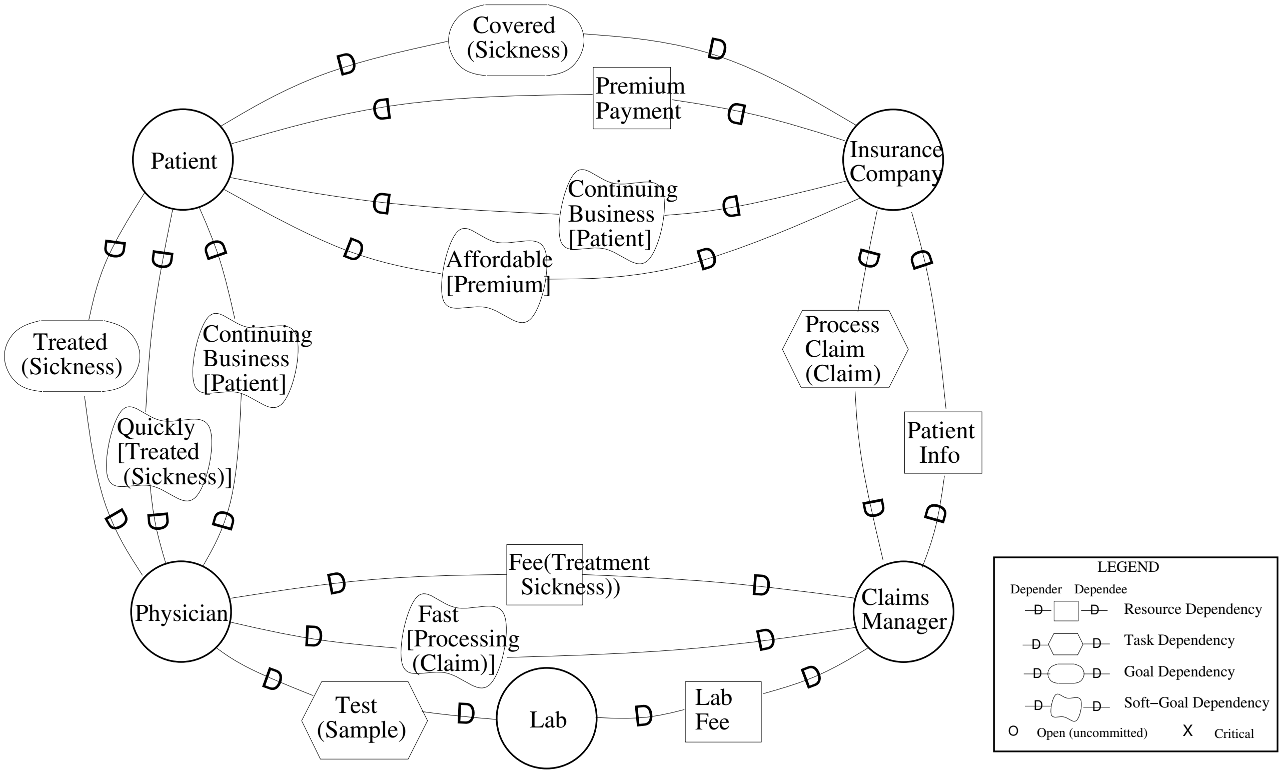

Figure 2.5: Health care configuration 1 – "full indemnity insurance"

Further details of the model can be given using knowledge structuring mechanisms such as classification, generalization, aggregation, and time. For example, there are different classes of physicians and laboratories offering different types of services, and classes of patients with different needs (types of sickness, age groups, varying demands for quality of service from health care providers and insurers). Their wants and abilities can be modelled as specializations of the fairly general dependencies indicated in Figure 2.5. The matching of these dependencies (using the query facility of the underlying representational language, such as Telos) would allow for an analysis of which wants are met or not met, and which abilities are utilized or not utilized. This type of analysis pertain to the opportunity aspect of the notion of intentional dependency.

Concerning the vulnerability aspect, we analyze an SD model for the viability of dependencies, by looking for patterns of dependencies that may serve to *enforce* commitment, *assure* success, or *insure* against failure.

Mitigating vulnerability. Various mechanisms can contribute to fortifying a dependency and to mitigate vulnerability.

A commitment is *enforceable* if there is some way for the depender to cause some goal of the dependee to fail, e.g., if there is a reciprocal dependency. A patient's dependency on a physician to have a sickness treated (and quickly) is likely to be enforceable because the physician depends on the patient for future business. On the other hand, the physician's dependency on the lab to perform tests may not be enforceable (according to the SD model of Figure 2.5) since the lab has no dependency on the physician.

Physicians depend on claims managers for fast claims processing. But there are no dependencies from claims managers to physicians. This suggests that physicians' dependency for fast claims processing may not be viable.

Assurance means that there is some evidence that the dependee will deliver the dependum, apart from the dependee's claim. For example, knowing that fulfilling the commitment is in the dependee's own interest would be an assurance (independently of whether or not the depender has enforcement mechanisms). For example, if fast claims processing is in the claims manager's own interest, regardless of whether this is desired by the physician, then the physician's dependency for fast claims processing is likely to be viable due to this assurance pattern. (This is shown in the next configuration.)

Insurance mechanisms reduce the vulnerability of a depender by reducing the degree of dependence on a particular dependee. A depender can improve the chances of a dependum being achieved by having more than one dependee for the same dependum (or parts thereof). A patient who gets a second opinion on his medical condition and treatment methods is making use of an insurance mechanism. A physician can send test samples to two independent labs. Purchasing an insurance policy from an insurer is of course another example of the insurance mechanism. In contrast to enforcement or assurance, insurance measures can be taken on the depender side without involving the original dependee. The different mechanisms for dealing with vulnerability are often used in combination. Chapter 7 (software process modelling) contains further examples of these types of analyses.

Note that the SD model provides the formal representation of the nodes and the links in a dependency network, and thus allows for analysis based on network topology, e.g., chain analysis, loop analysis, and node analysis (the confluence of various incoming and outgoing dependencies at an actor node). However, we do not attempt to give precise definitions for concepts such as enforcement, assurance, and insurance in terms of SD constructs because whether such mechanisms provide *sufficient* viability in a dependency is usually a matter of judgement from the viewpoint of specific strategic actors. In other words, these concepts are more appropriately treated as *soft concepts*, as introduced under the softgoal dependency type. In the Strategic Rationale model (Chapter 3), this type of reasoning is modelled and supported using an argumentative style of reasoning.

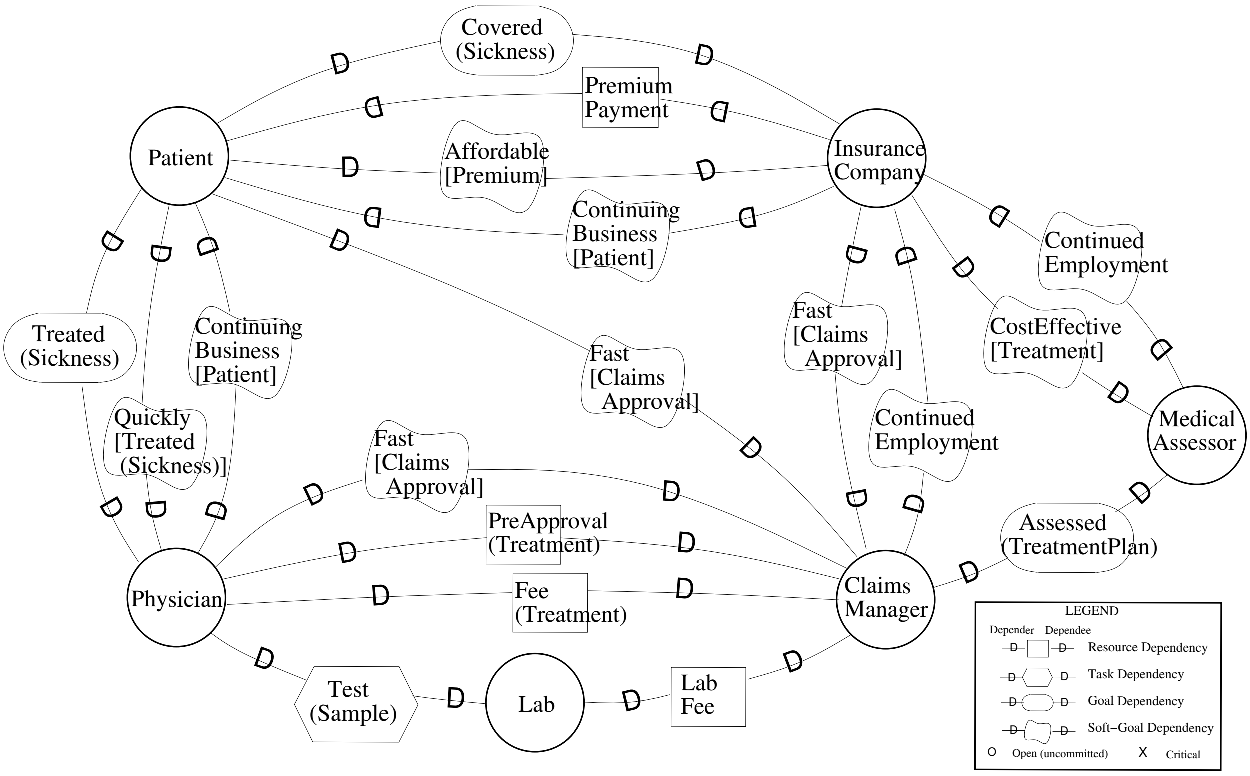

Figure 2.6: Health care configuration 2 – "managed indemnity insurance"

Health care configuration 2. In the configuration of Figure 2.6 ("managed indemnity insurance"), the insurance company tries to control costs and make premium affordable by requiring pre-approval of treatment. Patients' desire for quick treatment cannot be met by physicians unless there is fast approval from claims managers. The insurance company also wants fast approval for its own good. Patients' conditions can get worse while waiting for treatment, increasing the costs of treatment that the insurance company has to pay [Blackwell93]. The company thus wants fast claims approval from its claims managers. There is a convergence of interests. In the previous configuration (Figure 2.5), there is a potential conflict of interests: the insurance company can improve its cash flow by delaying payment to physicians, at the expense of physicians' cash flow.

The insurance company wants to approve only cost effective treatments. Claims managers do not have the ability to judge whether a treatment is cost effective. They depend on a medical assessor. This extra step involves hand-overs, slowing down the claims approval process. (Note that this configuration shows the claims managers as being ultimately responsible for fast claims approval. If not, the model would show a dependency from insurance company to medical assessor, with only a non-intentional flow between claims manager and medical assessor – see Chapter 6 (business process reengineering) for a discussion and representation of processes having the problem of work items "falling through the cracks".)

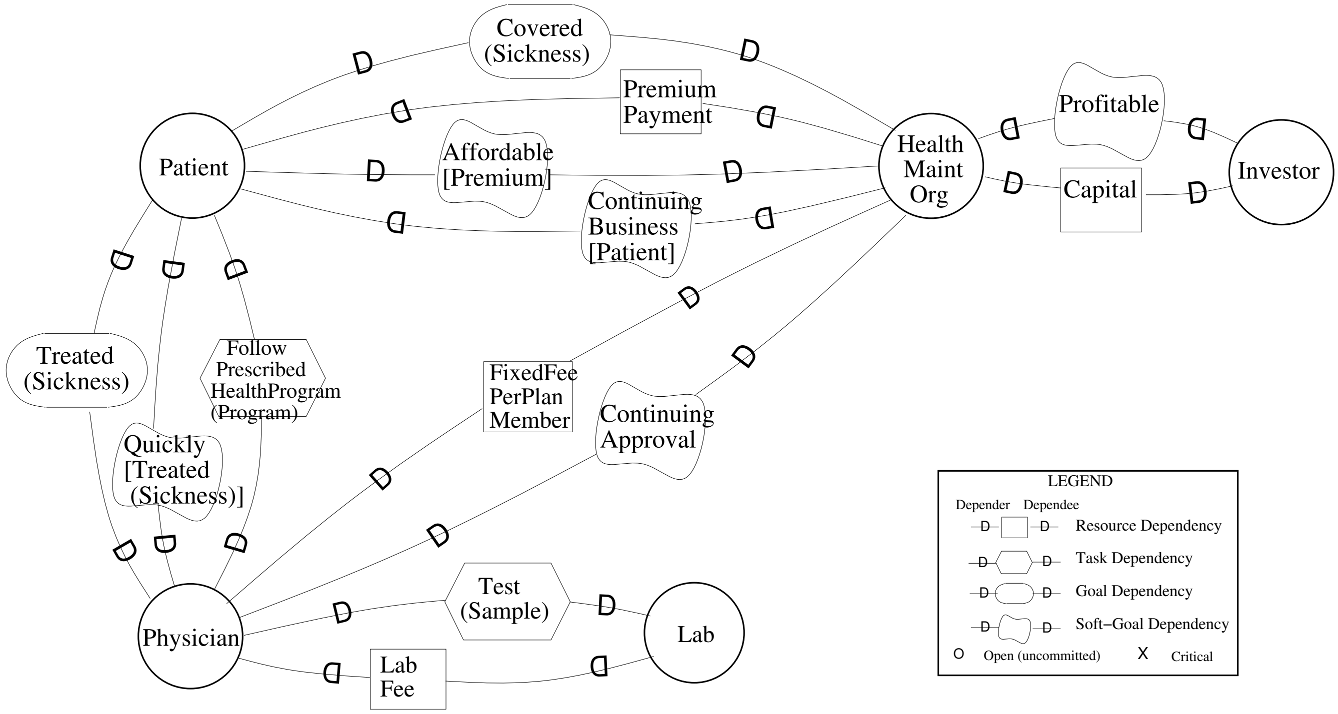

Figure 2.7: Health care configuration 3 – "managed care"

Health care configuration 3. In the configuration of Figure 2.7 ("managed care"), the patient gets health care by becoming a member of a health plan offered by a "health maintenance organization (HMO)". Unlike in indemnity insurance, the physician does not receive a fee for service, but is paid a fixed amount per member patient. Patients must go to physicians selected and approved by the HMO. Patients can get quick treatment (no waiting for pre-approval of treatment). Physicians pay for lab tests. It is now in the hands of the physician to control costs (e.g., by being more careful about ordering lab tests, and by prescribing fitness programs to patients to reduce sickness). This makes health care premiums more affordable to the patient, and more profitable for investors (according to one line of argument).

Although the above examples of health care provisioning configurations are drawn from the current health care reform debates in the United States [Loomis94], the models are not intended to be accurate or complete descriptions of the actual or proposed alternatives. For example, in many schemes, insurance premiums are not paid for directly by the patient, but by his/her employer. The patient's freedom to choose a health service provider, or an insurance plan, is often an issue. We have not attempted to reflect the actual issues in these debates. The examples used here are much simplified, but include enough detail to illustrate some of the powers of framework for dealing with realistic modelling scenarios.

Agents, Roles, and Positions. In the above configurations, we have also omitted differentiations among the various more specialized notions of actors – agents, roles, and positions. The modelling and analysis would be more intricate when these distinctions are introduced. Computer support in the analysis would then be especially helpful.

The agent, role, position distinction provides a way to separately identify those dependencies that are attributable to a role, as opposed to those that are attributable directly to a concrete agent. Performance measures, for example, are typically characteristics of a concrete agent. Training and education background pertain to an agent, since they accompany the agent when he/she moves to another position (e.g., the medical degree of Doctor). Qualification requirements, however, pertain to positions and roles (e.g., the position of Chief Surgeon). Roles typically have dependencies that relate to the "tasks" or "functions" that an agent may become associated with or disassociated with, for example, by taking on or moving away from a position.

When we separate out these "components" of a social actor, each component is a partial description of the social actor. Each component gives some hints about how the social actor might behave, or is expected to behave, in some specialized, narrow context. Such hints are useful because they give a first-cut approximation, thus simplifying the picture. But we also need to recognize its limitations, because they can often be misleading. It does not take the other aspects of the social actor into account. The behaviour of the social actor can only be understood in its totality.

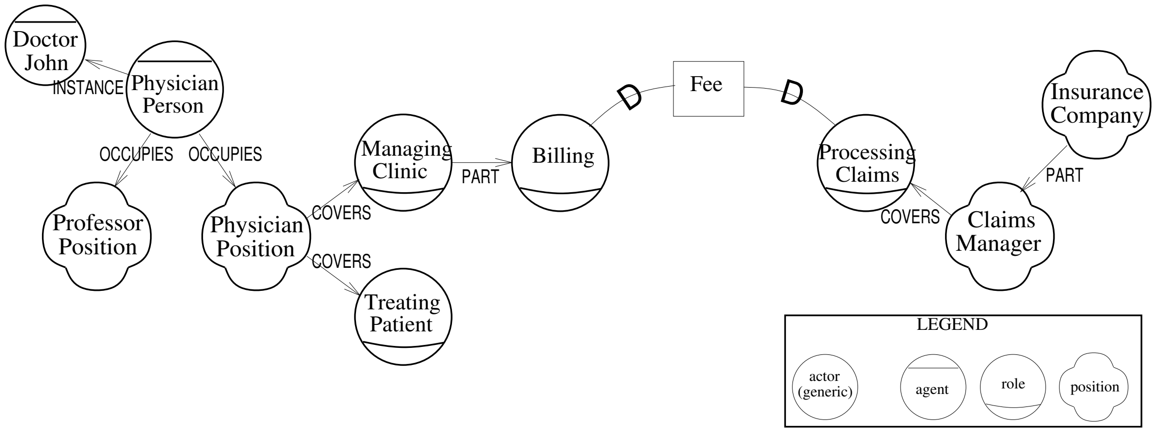

Figure 2.8: Agents, Roles and Positions

Each association from agent to position to role can be problematic. For example, while there may be certain expectations on a role, when this role is covered under a position, the combination of roles may result in conflicts that make those expectations less likely to be fulfilled. Similarly, a position may entail expectations that a concrete agent with specific abilities, knowledge, and aspirations may not exactly match. Figure 2.8 shows an example of some agent-role-position relationships in a health care setting. Further examples which make use of the agent-role-position distinctions are presented in chapters 6 (organizational impact analysis) and 7 (software process modelling).

Intentional relationships versus non-intentional flows. It is important not to view these chains as mere flows. There is possibility of failure at each dependency, and we want to know its implication. For example, if the pharmacist sent the wrong medication, that could adversely affect the physician's ability to cure the patient. The type of dependency reveals whose goals might be affected by a problem, and indicates which side will deal with the contingency.

Given a non-intentional description of a relationship between two actors, e.g., between a car owner and a repair shop with respect to the owner's car, one could not say, without further knowledge about the relationship, which dependency type would be appropriate for modelling the relationship. One would have to know, given certain types of uncertainties (problems that require decisions), which side would come up with the options, and which side would make the decision. If, when a problem with the car arises, the two sides have no way of addressing the problem except to replace it with another car, then this is a resource dependency. If, when the car develops a problem, the owner tells the repair shop to "just get it fixed", this is a goal dependency. If the owner tells the shop to "have it fixed economically", this is a softgoal dependency. For instance, if the car stalls easily, the shop might suggest overhauling the carburetor, or, as a stop-gap measure, one could raise the engine idle setting. The owner would select (and agree to) one alternative before the shop would start work. If the owner goes to the shop and asks to have the idle setting raised, without giving the shop the chance to suggest alternatives, this would be a task dependency. The owner is telling the shop what to do, without making use of the shop's knowhow for achieving the goal.

2.4 Formal Characterization

The Strategic Dependency model has been presented informally in the above, using descriptive text, a graphical notation, and illustrative examples. In this section, we consider the formal characterization of the modelling concepts of the SD model.

In conceptual modelling, the purpose of taking a more formal approach (compared to, say, boxes-and-arrows notations which do not have clear semantics) is to give a more precise meaning to the concepts, so as to be able to adjudicate among different interpretations of a given model, and to offer a basis for building tools which assist in reasoning with the models [Greenspan94].

In artificial intelligence research (e.g., [Cohen90][Thomas91][Lesperance91]), intentional concepts such as goal, belief, ability, and commitment have been formalized for characterizing artificial agents. However, these formulations are not directly applicable to the $\boldsymbol{i}^*$ framework, and therefore need to be adapted, for at least the following reasons:

- $\boldsymbol{i}^*$ deals with intentional *relationships* among agents, not just intentional properties of agents in isolation.
- $\boldsymbol{i}^*$ models are meant to be descriptive. In reasoning about strategic agents, we need to assume that agent behaviour can be quite unpredictable and open. Agent models developed in AI are primarily intended to be prescriptive specifications. Designers are expected to produce artificial agents (e.g., robots) whose behaviour do not violate the specification.
- $\boldsymbol{i}^*$ has a semi-formal component, involving the concept of satisficing, which is not readily amenable to formalization using conventional techniques, because the interpretation(s) of "soft concepts" are typically context-dependent.

In this section, we discuss the characterization of the SD concepts in relation to formal agent modelling concepts developed in AI. We do not aim to provide a full integration of the concepts into a particular formal system. The extent to which $\boldsymbol{i}^*$ concepts can be formalized, in view of the limitations cited above, remains an open issue for future research.

The axioms in this section, though primarily suggestive, nevertheless provide a sharper characterization of the SD concepts than in the informal presentation. Support tools for the framework need to provide facilities that are consistent with the characterization given in these axioms.

2.4.1 Preliminary Concepts

We first explain some of the underlying concepts and rationales for adopting our approach to formalizing the SD model.

We adapt the concepts of ability and commitment as developed for agent modelling in artificial intelligence. The concepts of "routines" and "workability" are introduced to deal with the needs of strategic actor reasoning.

The Strategic Dependency model aims to present a picture of agents by explicitly modelling only their external intentional relationships with each other. The semantics of these external relationships, however, are characterized in terms of some presumed internal intentional features of the agent. Some of these internal features are explicitly modelled in the Strategic Rationale model, described in the next chapter.

We assume that the internal intentional structure of an agent x includes at least the following three components: $\mathcal{U}_x$ – a set of routines, $\mathcal{H}_x$ – a set of means-ends rules, and $\mathcal{E}_x$ – a set of "primitively workable elements".

All free variables in formulas are understood to be universally quantified.

Routines. The internal characterization of an agent centres around the routines held by the agent and the elements that make up the routine. A **routine** is the primary vehicle through which an agent can accomplish what it wants. It is a template for the agent's recurring activities. A routine consists of elements, which include subgoals, subtasks, resources, and softgoals. There can be relationships among elements specified as constraints, such as temporal precedence.

A routine can be seen as a plan skeleton. It provides the rough outline for the specific actions to be carried out when instantiated, but allows the details to be worked out at the time of execution.

An agent's set of routines is explicitly enumerated in the Strategic Rationale model. Although means-ends reasoning can be computer-supported, the incompleteness inherent in strategic reasoning requires the user to explicitly sanction each inference, possibly drawing on knowledge and judgement that is not explicitly represented in the model.

Ability. When an agent has a routine that can serve some purpose, e.g., to achieve a certain goal, we say that the agent has an **ability** to achieve that goal. We use the predicate $A(y, \eta)$ to indicate that agent y has ability to produce or achieve the intentional element η.

- Ae: $A(y, \eta) \equiv \exists u(\mathcal{U}_y(u) \wedge purpose(u, \eta))$

Agent y has the ability to achieve η iff y has in its repertoire of routines $\mathcal{U}_y$ a routine u whose purpose is the achievement of η. Note that in contrast to operational reasoning (as in AI planning and action, or the specification of such planning and action), we do not require a guarantee that the agent can indeed achieve the goal by some sequence of actions. We only require that it knows what to do, at a coarse-grained level. This is a weaker notion compared to those used for the characterizations of ability in an operational setting. (We will refer to the usual AI planning setting (e.g., [Nilsson80] as "operational", in contrast to the "strategic" setting here.)

Workability. In order to provide a stronger notion of ability suitable for strategic reasoning, we introduce the notion of "workability" – to indicate that an agent believes that some routine would work (at "run-time"), even though it is incompletely specified or known (at "strategy-time", i.e., during process analysis or design). During strategic reasoning, an agent is content to reduce a solution to a level at which all components are workable.

A routine is judged to be workable if each of its explicitly mentioned elements is workable, and if all of the constraints in the routine are expected to hold. We say that an element η is workable for agent y – $W(y, \eta)$ – if that agent has a workable routine to produce η.

Intuitively, an element η is made workable by reducing it through the routine to primitively workable elements (though not necessarily primitively executable actions), or by delegating some of the elements to other agents. A primitively workable element is one that is judged to be workable without further reduction. In the SD model, our main concern with routines is the delegation part. The SR model described in the next chapter explicitly models the reduction.

Commitment. Workability is a local property – it is one agent's judgement about whether its own routines would work. When a routine involves dependencies on other agents, the depender agent usually cannot make judgements about the workability of the routines in dependee agents, because it does not know enough about those routines, or the workability of the elements in those routines. However, it is important for agents to be able to assess the workability of routines which involve dependencies on other agents. The notion of *commitment* offers a way out of this dilemma. We take the view that if an agent is able to achieve η, and is committed to doing so, then η is workable for that agent. Commitment thus provides an abstraction that allows workability to be judged without having to know about the routines used to achieve η, or the need to judge the workability of the individual elements that make up those routines. Commitment is the property that bridges the gap between ability and workability.

- WCA: $A(y, \eta) \wedge \exists x C(y, x, \eta) \supset W(y, \eta)$

If agent y is able to achieve η, and y is committed to achieving η for x, then η is workable for y. We will call this the **Workability-Commitment Assertion**.

This may be compared to the characterization of ability as $A(y, \phi) \equiv_{def} \exists x C(y, x, \phi) \supset \phi$ (where ϕ is a proposition about the world) in a framework for specifying artificial agent operational behaviour [Thomas91]. For strategic reasoning, we are only concerned about workability ($W(y, \eta)$), not the (stronger) requirement that η be actually achieved. As in [Thomas91], commitment is taken to be a primitive intentional state of an agent.

Ideally, one would like the workability commitment assertion to be always true, i.e., so that it can be stated as an axiom (that holds for all x and y). This would be appropriate in the context of artificial agent specification. One can prescribe to have artificial agents *designed* to meet this specification. However, since we are modelling (describing) agents in the real world (natural or artificial), we cannot assume that such agents can and will abide by the prescription. In the SD model, this assertion typically appears within the belief context of some agent, such as the depender. Such a belief may need to be justified, if the agent is concerned about the viability of the dependency. In this case, the belief may be "raised" as an "assumption". We then call it the *Workability-Commitment Assumption*. This is further discussed under the SR model.

Transfer of Workability. The Workability-Commitment Assertion only says that if y is able and committed on η, then η is workable for y. For delegation to work, we want η to be workable for x the depender, not just for y the dependee. The delegated element needs to be workable in some routine of x, in order that that routine becomes workable *through* the delegation. We therefore also need the following **Workability-Transfer Assertion**:

- WTA: $W(y, \eta) \wedge C(y, x, \eta) \supset W(x, \eta)$

If η is workable for y and y is committed to achieve η for x, then η is workable for x. Again, this typically appears as a belief of the depender (x), and may require justification, in the form of a *Workability-Transfer Assumption*.

These two assertions together make delegation workable.

$$W(x, \eta) \subset \exists y(A(y, \eta) \wedge C(y, x, \eta))$$

The element η is workable for agent x if there is some agent y who is able to achieve η and is committed to achieving η for the depender x.

With the above preliminaries, we are now ready to present the axiomatic characterization of the dependency concepts.

2.4.2 The Dependency Operators

The SD model provides an external characterization of an agent in terms of two sets of dependencies: incoming dependencies (the agent as dependee) and outgoing dependencies (the agent as depender). We use the left half-arrow (pointing from right to left) to denote incoming dependency and the right half-arrow to denote outgoing dependency.

We will use η to denote a generic dependum. Axioms for different types of dependums are variations of the generic ones. For goal dependency, η is an assertion representing the goal g; for task dependency on task t, η is $done(t)$; for resource dependency on resource r, η is $avail(r)$; for softgoal dependency on softgoal s, η is $satisficed(s)$.

We start with a basic dependency on the dependee side. If agent y offers itself as a dependee with η as dependum, we expect at least that y has a routine for achieving η, i.e., y is able to achieve η.

- De: $\overleftharpoon{D}(y, \eta) \supset A(y, \eta)$

We do not require an implication in the other direction, because y may not want to offer to achieve η for other agents, even if it is able. Intuitively, this operator indicates that agent y is able *and willing*. We will use this as the characterization of the open (uncommitted) dependency operator on the dependee side (We will not separately define an $O\overleftharpoon{D}$ operator).

With commitment, the open dependency becomes a committed dependency. The committed dependency on dependee side is defined as conjunction of Commitment and the depended predicate.

- CDe: $C\overleftharpoon{D}(y, x, \eta) \equiv \overleftharpoon{D}(y, \eta) \wedge C(y, x, \eta)$

On the Depender side, the situation is more complex. We need to define some intermediate concepts, so as to address the opportunity and vulnerability aspects separately.

Intuitively, the opportunity aspect of an outgoing dependency could be characterized as agent x's belief that there is some other agent y who is offering itself as a dependee (i.e., is able and willing), and if this agent y commits to x, then η will be workable for x.

- DrOpp: $\overrightharpoon{D}_{opp}(x, \eta) \supset B(x, \exists y(\overleftharpoon{D}(y, \eta) \wedge (C(y, x, \eta) \supset W(x, \eta))))$

where B is some belief operator (such as those used in artificial intelligence for characterizing artificial agents, e.g., in [Cohen90] or [Thomas91]). However, this characterization would prevent η from being used as an outgoing dependency if the above belief is not yet established in agent x. We therefore adopt a more flexible approach, by treating $\overrightharpoon{D}_{opp}(x, \eta)$ as a primitive intentional property of x. When the *workability* of the outgoing dependency is questioned, we then raise the *believability* of the above condition as an *issue* to be addressed, using a qualitative reasoning scheme. This aspect of $\overrightharpoon{D}_{opp}$ is covered under the SR model in the next chapter, and the associated axiom is given in section 3.6.2.5 (iii).

We now consider the different levels of dependency strengths. For an open dependency, on the depender side, the depender recognizes the opportunity, and has the "open" level of vulnerability.

- ODr: $O\vec{D}\ (x,\eta) \equiv \vec{D}_{opp}\ (x,\eta) \wedge O\vec{D}_{vul}\ (x,\eta)$

We take the notion of committed dependency on the depender side to consist of two parts:
i) x recognizes the opportunity being offered by y, and
ii) x is (self-) committed to (using) η (and is therefore vulnerable).

- CDr: $C\vec{D}\ (x,y,\eta) \equiv C\vec{D}_{opp}\ (x,y,\eta) \wedge C\vec{D}_{vul}\ (x,\eta)$

$C\vec{D}_{opp}\ (x,y,\eta)$ is similar to $\vec{D}_{opp}$ except that it is for a particular y.

In a critical dependency, the depender is more vulnerable than in a committed dependency, but otherwise has the same characteristics.

- XDr: $X\vec{D}\ (x,y,\eta) \equiv C\vec{D}\ (x,y,\eta) \wedge X\vec{D}_{vul}\ (x,\eta)$

The characterization of vulnerability is based on the extent to which the workability of η would affect the workability of the routine in which η is supposed to serve. To formally distinguish among the three degrees of vulnerability offered by the model – open, committed, and critical, we make use of the concept of routines. We assume that the depender is using the dependum η in some routine.

A dependency is **open** if the dependum disables only part of a routine, i.e., the dependum being unworkable would make some part of that routine unworkable.

- ODrVul: $O\vec{D}_{vul}\ (x,\eta,u) \supset \exists u'(\mathcal{U}_x(u') \wedge subroutine(u',u) \wedge (\neg W(x,\eta) \supset \neg W(x,u')))$
- ODrVul2: $O\vec{D}_{vul}\ (x,\eta) \equiv \exists u(\mathcal{U}_x(u) \wedge O\vec{D}_{vul}\ (x,\eta,u))$

We say that a depender depends on η to the **committed** level if the routine becomes unworkable if η is unworkable.

- CDrVul: $C\vec{D}_{vul}\ (x,\eta,u) \supset (\neg W(x,\eta) \supset \neg W(x,u))$
- CDrVul2: $C\vec{D}_{vul}\ (x,\eta) \equiv \exists u(\mathcal{U}_x(u) \wedge C\vec{D}_{vul}\ (x,\eta,u))$

The committed level of vulnerability has a disabling effect on one routine. A much stronger dependency results if a dependum can have a disabling effect on all routines that the agent has for achieving some purpose. We call this a **critical** dependency.

- XDrVul: $X\vec{D}_{vul}\ (x,\eta) \supset \forall\eta'\forall u\forall\alpha(\mathcal{U}_x(u) \wedge (\mathcal{H}_x(\eta',u,\alpha) \wedge \neg W(x,\eta)) \supset \neg W(x,\eta'))$

For all of x's routines which have η' as an end, an unworkable η would lead to η' being unworkable, given the agent's set of means-ends rules $\mathcal{H}_x$. In other words, x has no other way of making η' workable. Rules are explained more fully in the next chapter.

The formal characterization given in this section captures the central notions of intentional dependency, without committing to a particular choice of logic for the underlying intentional operators. Further refinement of these notions based on choices of different underlying logics is left as future work.

2.5 Summary

In this chapter, we presented a model for describing processes in terms of intentional, strategic dependency relationships among actors in organizational settings. By making the intentional properties of a process explicit (in contrast to non-intentional models such as workflow models), one is able to analyze strategic implications of a process configuration, such as the opportunities that are open to an agent, and the vulnerabilities that it faces. Mechanisms for mitigating vulnerability, such as those for enforcing commitment, assuring success, and insuring against failure can also be analyzed. The model is embedded in the Telos conceptual modelling language. Actors are differentiated into agents, roles, and positions.

Examples from the health care domain were used to illustrate the various modelling concepts. Throughout this thesis, the examples are meant to be illustrative only, and do not aim to provide the degree of completeness that would be typical of an actual application. Some of the examples in the application chapters (4 to 7) are more detailed. However, the scalability of the framework to deal with realistically large domains remain to be tested. [Briand94] offers some initial insights into the modelling of an actual organization using the SD model.

The concepts of the SD model were characterized axiomatically, by adapting intentional concepts originally developed for modelling agents in artificial intelligence.

Chapter 3

The Strategic Rationale (SR) Model

The Strategic Rationale (SR) model provides an intentional description of processes in terms of process elements and the rationales behind them. While the Strategic Dependency (SD) model maintains a level of abstraction by modelling only the external relationships among actors, the SR model foregoes that abstraction in order to allow a deeper understanding about strategic actors' reasoning about processes to be explicitly expressed. The SR model describes the intentional relationships that are "internal" to actors, such as means-ends relationships that relate process elements, providing explicit representations of "why" and "how" and alternatives. The rationales are at a strategic level, so that the process alternatives being reasoned about are strategic relationships, i.e., SD configurations. Using knowledge represented in and organized by these modelling concepts, process alternatives can be systematically generated and explored, so as to help actors find new process designs that better address their interests, needs, and concerns. An axiomatic characterization provides a basis for the future development of tools for supporting process analysis and design.

Section 3.1 presents the modelling features of the SR model by way of examples. Section 3.2 shows how these modelling concepts are embedded in the conceptual modelling language Telos. Section 3.3 outlines the analytical capabilities of the model. Section 3.4 describes how the model is used to support systematic process design. Section 3.5 gives the formal characterization of the model.

3.1 Modelling Features

The SR model is a graph, with several types of nodes and links that work together to provide a representational structure for expressing the rationales behind processes.

There are four types of nodes, based on the same distinctions made for dependum types in the SD model – **goal, task, resource**, and **softgoal**. There are two main classes of links: means-ends links and task decomposition links. A **means-ends link** indicates a relationship between an end – which can be a goal to be achieved, a task to be accomplished, a resource to be produced, or a softgoal to be satisficed – and a means for attaining it. The means is usually expressed in the form of a task, since the notion of task (as introduced in preceding chapter) embodies *how* to do something. This is done by way of describing the elements (components) of a task. A task node is linked to its component nodes by **task decomposition links**. There are four types of task decomposition links – subgoal, subtask, resourceFor, and softgoalFor – corresponding to the four types of nodes. These links also can connect up with dependency links in Strategic Dependency model(s), when the reasoning goes beyond an actor's boundary. A **routine** is a subgraph representing the rationales for one process (one particular combination of elements that constitutes a means for accomplishing some end). Means-ends links are taken to be applications of generic means-ends relationships called **rules**.

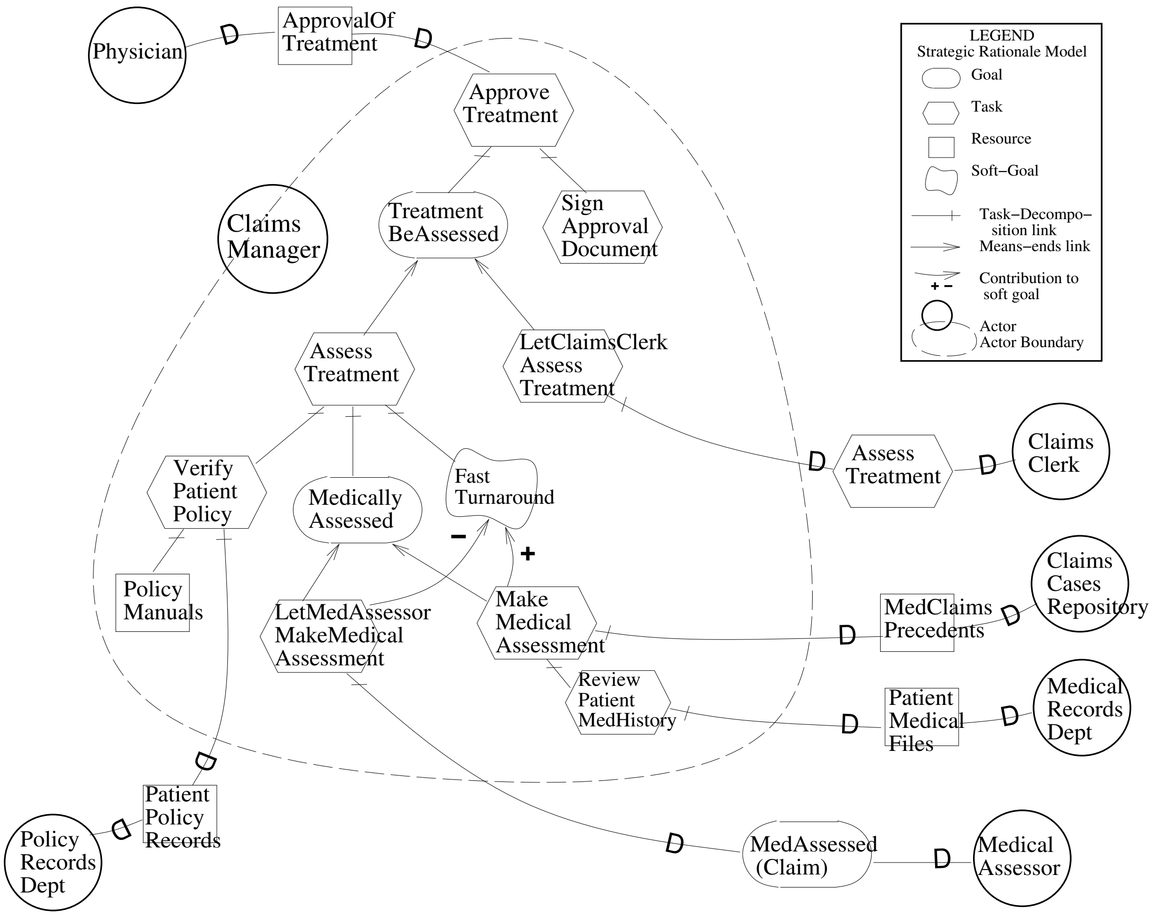

Figure 3.1: A Strategic Rationales Model – insurance claims manager example

Figure 3.1 shows a graphical representation of an SR model that captures some of the rationales involved in an insurance claims setting. A physician must submit a treatment plan to the insurance company for prior approval, or else the treatment may not be reimbursed. The insurance company verifies that the type of treatment is covered by the policy, and that the proposed treatment is reasonable according to medical opinion. In the preceding chapter, we have looked at the external dependencies for this example. Now we look inside the agents to show some of the different configurations which the insurance company could adopt to process the submitted treatment plan.

Starting from the leftmost side, the model shows that the Claims Manager is able to produce `ApprovalForTreatment` via the `ApproveTreatment` task. This task consists of two components – the subgoal that `TreatmentBeAssessed`, and the subtask of signing off the approval document. Note that we are modelling only those task elements that are considered important enough to be of strategic concern to the actor.

One way to have the treatment plan assessed is to let a claims clerk do it. Another way is for the claims manager to do it herself. This alternative requires the claims manager to verify the policy (that the medical condition and the treatment plan are covered under the patient's policy, and that the policy is in force) and also to have the treatment plan assessed for its medical appropriateness. The latter can be achieved by relying on someone with special medical knowledge (a medical assessor) to do it, or by doing it herself, making use of case knowledge about previous claims, from a repository. Figure 3.2 shows the main link types of the Strategic Rationale model.

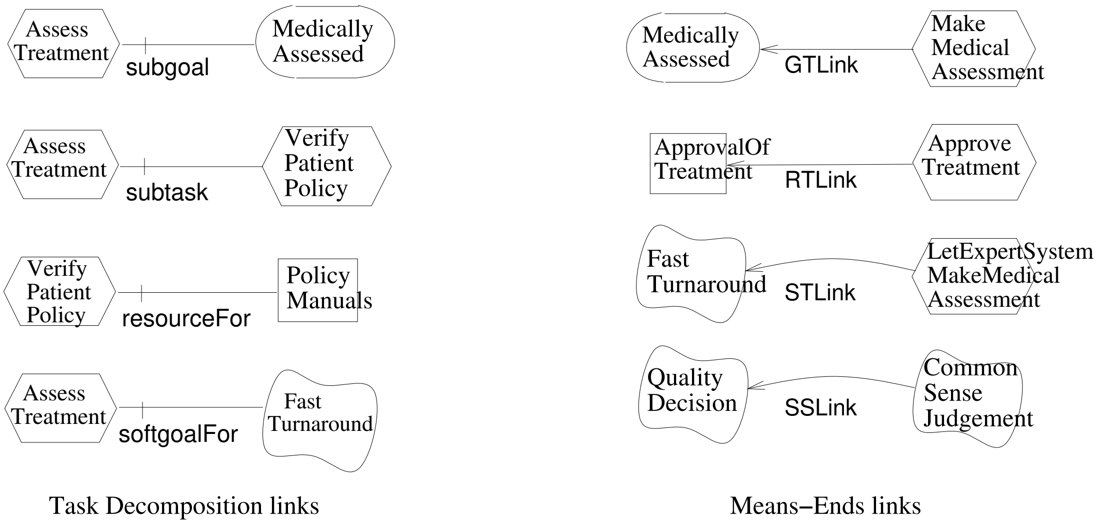

Figure 3.2: The main link types in the Strategic Rationale model, with domain examples

Task-Decomposition Links. A task is modelled in terms of its decomposition into its subcomponents. These components can be goals, tasks, resources, and/or softgoals. The distinction among these have been introduced in the Strategic Dependency model, where they appear as dependums in the strategic relationships between actors. These same distinctions are useful in elaborating on the makeup of a task.

A **goal** is a condition or state of affairs in the world that the actor would like to achieve. It is expressed as an assertion in the representation language. *How* the goal is to be achieved is not specified, allowing alternatives to be considered. In the example, the `MedicallyAssessed` component of `AssessTreatment` is modelled as a goal, which indicates that there can be different ways for achieving it.

A **task** specifies a particular way of doing something. When a task is specified as a subcomponent of a (higher) task, this restricts the higher task to that particular course of action. The `VerifyPatientPolicy` component of `AssessTreatment` is modelled as a task. This means that `AssessTreatment` involves a particular way to verify patient policy, as specified in turn by the decomposition of `VerifyPatientPolicy`.

A **resource** is an entity (physical or informational) that is not considered problematic by the actor. The main concern is whether it is available (and from whom, if it is an external dependency). `PolicyManuals` is a resource for `VerifyingPatientPolicy` in the example.

A **softgoal** is a condition in the world which the actor would like to achieve, but unlike in the concept of (hard-) goal, the criteria for the condition being achieved is not sharply defined *a priori*, and is subject to interpretation. When a softgoal is a component in a task decomposition, it serves as a quality goal for that task, and thus guides (or restricts) the selection among alternatives in further decomposition of that task. `FastTurnaround` is a softgoal for `AssessTreatment`. This indicates that, for the task of `AssessTreatment`, how fast is fast enough is not a sharp, a priori defined criteria. A qualitative judgemental evaluation is indicated. Where there are alternatives in further elaboration of`AssessTreatment`, the softgoal `FastTurnaround` is used as a selection criteria, e.g., to choose between`MakeMedicalAssessment` (done by the Claims Manager herself), and`LetMedicalAssessorMakeMedicalAssessment`).

There may be **constraints** among the components of a task, such as temporal relationships. These are not shown in the graphical notation, but appear in the formal language notation (Telos).

Each task-decomposition link can be "open" or "committed". **Committed** means the agent beliefs the routine will fail if this element fails (see sections on Analysis (Section 3.4) and formal characterization (Section 3.6) for the concept of workability). **Open** means that the routine would be affected, but would not necessarily fail. If the link is an outgoing dependency link, the link can also be "critical". **Critical** means that the agent believes there is no other way to succeed.

Means-Ends Links. The SR model also provides for several types of means-ends links. The "end" can be a goal, task, resource, or softgoal, whereas the "means" is usually a task. In the graphical notation, the arrowhead points from the means to the end.

In a **Goal-Task Link** (`GTLink`), the end is specified as a goal, and the means is specified as a task. The link between the goal `MedicallyAssessed` and the task `MakeMedicalAssessment` is a goal-task link. This task specifies the "how" through its decomposition into components.

In a **Resource-Task Link** (`RTLink`), the end is specified as a resource, and the means is specified as a task. The link between `ApprovalForTreatment` and`ApproveTreatment` is an example of a resource-task link.

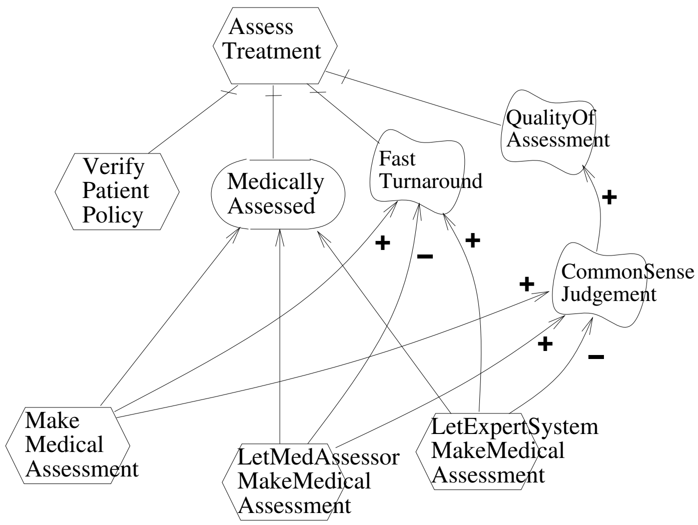

Figure 3.3: An example of softgoal links in the Strategic Rationales model

There are two common link types involving softgoals. A **Softgoal-Task Link** (`STLink`) have a softgoal as the end, and a task as the means. For visual perspicuity, softgoal links are shown as curved arrows in the graphical notation. For example, to achieve `FastTurnaround` (the softgoal), a claims manager can `MakeMedicalAssessment` herself (the task). On the other hand, letting a medical assessor do it (the task) leads to a negative contributions towards `FastTurnaround`. Links involves softgoals require an extra attribute to indicate the type of contribution – positive or negative, enough (sup) or not enough (sub). [Chung93] provides a framework for qualitative reasoning using these concepts, using a multi-valued evaluation scheme (satisficed, denied, conflict, unknown, etc.). In this thesis, we simply assume that such frameworks for propagating and drawing conclusions from a qualitative network of arguments exist. Most of our examples will only indicate positive and negative contributions, but will not illustrate the reasoning steps. The **Softgoal-Softgoal Link** (`SSLink`) permits the development of a means-ends hierarchy of softgoals, until eventually some softgoals are addressed by linking to tasks. (Figure 3.3).

At an actor boundary, an incoming dependency link is also an implicit means-ends link, with the dependum being the "end". An outgoing dependency link is usually also a task-decomposition link, the dependum being one of the task components.

Other means-ends link types are possible as a result of the combinations of element types for the means and for the end.

In a **Task-Task Link** (`TTLink`), both the end and the means are tasks. This is allowed due to the inherent openness (incompleteness) assumed by the modelling framework. Even though a task specifies the "how", it is still possible to have alternatives arising from it because further components can be added. An example of this is a specialization (IS-A) relationship between two tasks. Thus a task-task link can still be seen as a means-ends relationship. There can be multiple alternatives of tasks (the means) which accomplish the "ends" task. This is unlike the modelling of the decomposition of non-intentional activities into more detailed non-intentional sub-activities (cf. RML [Greenspan84]).

The above link types all have task as the means specification. However, it is sometimes desirable to have restricted form of a task as the "how". For example, goal reduction: given a goal as the end, the means could be specified as a conjunction of subgoals (with no other types of components). This would be a **Goal-Goal Link** (`GGLink`).

Routines. In order to refer to a particular set of choices in the SR graph, we introduce the notion of a routine. A **routine** is a subgraph in the SR graph with a single link to a "means" node from each "end" node. A routine therefore represents one particular course of action among the multiple alternatives presented at each OR node – one means-ends branch out of the many possible to address each task component node. (However, in the case of softgoals, since the means-ends links (SS and ST) represent partial contributions of means to ends, a routine will include multiple means-ends links contributing to softgoals.)

The notion of a routine is used to refer to one process and its rationales. One example is the subgraph that includes `LetClaimsClerkAssessTreatment` and `SignApprovalDocument`. Another example is the subgraph that includes `VerifyPatientPolicy`, `LetMedicalAssessorMakeMedicalAssessment`, and `SignApprovalDocument`. The routine is a convenient unit for analysis when evaluating alternatives. (see next section).

Routines typically have connections to other actors, via dependency links in Strategic Dependency models.

The term em routine is used to convey the typically recurrent nature of a work process. By describing a routine in terms of intentional elements (goals, tasks, resources, and softgoals), the SR model acknowledges that additional problem solving usually takes place at the time a routine is carried out [Suchman83,87]. This is in contrast to mechanistic notions of *procedure*, or the concept of *plan* as a series of primitive actions in classical artificial intelligence.

Rules. The means-ends links in an SR graph are shown as embedded in a particular context. They are rationales. However, these links can be seen as *applications* of more generic relationships – which say that *whenever* you have some element as an end, you can use some other element as a means to that end. For example, there could be another means-ends link in the same SR graph (say, on a different branch) that is an application of the same principle. We call the generic principle a *rule*. A rule is a means-ends link that is not yet bound. A **rule** consists of an applicability condition, a means, and an end. A means-ends link is an application of a rule in a context in which the agent believes the applicability condition to hold.

For example, `MakeMedicalAssessmentLocally` can be a GT-Rule which has been applied as the means-ends link linking the goal node `MedicallyAssessed` and the task node `MakeMedicalAssessment`, with an applicability condition stating that the actor doing the assessment must have the requisite knowledge for doing medical assessments.

Beliefs. Rationale elements such as applicability conditions in rules are supplementary information that are not directly part of the means-ends hierarchy (such as the routines depicted in Figure 3.1). They provide supporting evidence for or against alternatives in the choice of routines. These are modelled as *beliefs*, constituting a supplementary network of nodes and links in an argumentative style of reasoning.

3.2 Representational Constructs

Figure 3.4 shows a schema for the SR model. The middle section of the figure deals with task decomposition links and their corresponding dependency links. A task can be decomposed into subgoals, subtasks, resources and softgoals. Each of these have a dependency link counterpart. For means-ends links and rules, only the Goal-Task (GT) and Resource-Task (RT) cases are shown. The others are analogous.

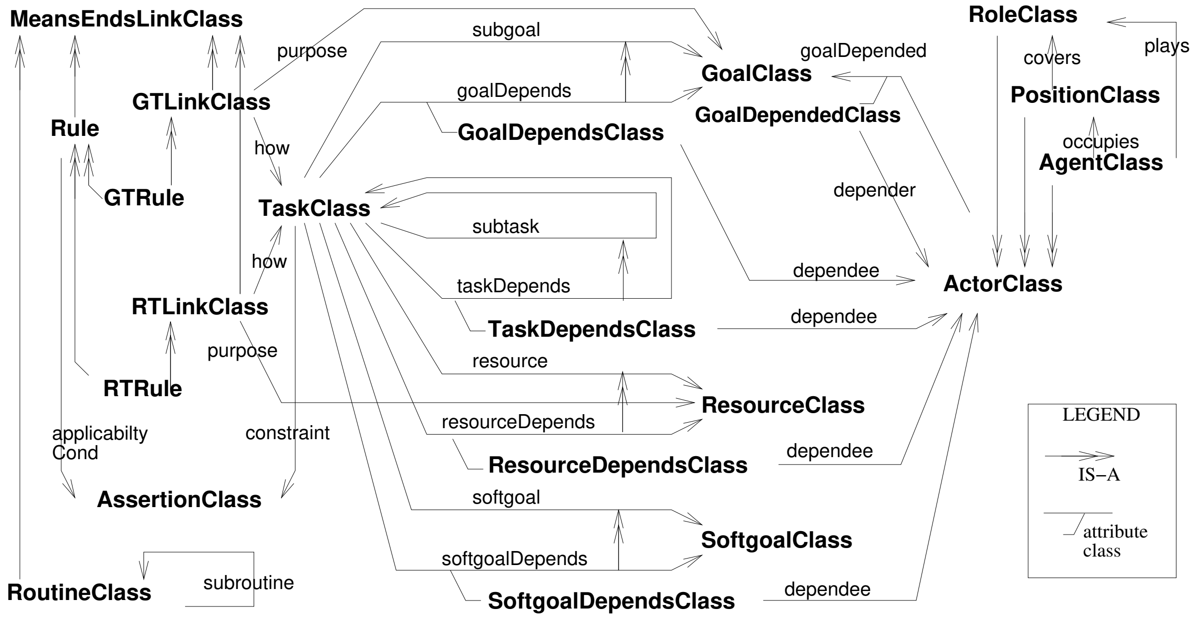

Figure 3.4: A partial schema for the Strategic Rationale model, showing task decomposition links and some classes of dependency links

```
MetaClass TaskClass ISA ElementClass, ActivityClass IN Class WITH
  attribute
    subtask: TaskClass
    subgoal: GoalClass
    resource: ResourceClass
    softgoal: SoftgoalClass
    taskDepends: TaskDependsClass
    goalDepends: GoalDependsClass
    resourceDepends: ResourceDependsClass
    softgoalDepends: SoftgoalDependsClass
    constraint: Assertion
  END
```

Two domain class definitions corresponding to the example in Figure 3.1 are:

```
Class ApproveTreatment IN TaskClass WITH
  subgoal
    assessed: TreatmentBeAssessed
  subtask
    sign: SignApprovalDocument
  END

Class AssessTreatment IN TaskClass WITH
  subgoal
    ma: MedicallyAssessed
  subtask
    vpp: VerifyPatientPolicy
  softgoal
    ft: FastTurnaround
  END
```

The dependee actor is attached as an attribute of the link from the task to its dependum. This permits a dependum to have multiple dependees.

```
Class VerifyPatientPolicy IN TaskClass WITH
  resource
    pm:  PolicyManual
  resourceDepends
    pi: PatientPolicyRecords WITH
          dependee
             prd: PatientRecordsDept END
  END
```

An incoming dependency is a link from an actor to a dependum, with the depender represented as an attribute on the link. Figure 3.4 only shows the one for goal class (attribute `goalDepended`), the others are analogous.

The left-hand section of Figure 3.4 shows relationships among means-ends links, rules and routines. Means-ends links have a purpose and a how. Each type of means-ends link is a specialization of this, with different types of intentional element as purpose and how. (For brevity, the figure only shows the Goal-Task (`GTLinkClass`) and Resource-Task (`RTLinkClass`) types of means-ends links.)

```
MetaClass MeansEndsLinkClass  IN Class WITH
  attribute
    purpose: ElementClass
    how: ElementClass
```

```
  END

MetaClass GTLinkClass  ISA MeansEndsLinkClass WITH
  attribute
    purpose: GoalClass
    how: TaskClass
  END
```

Here are two domain class definition examples:

```
Class ExpertSystemCanDoMedicalAssessment IN GTLinkClass
   WITH
  purpose
    m: MedicallyAssessed
  how
    les: LetExpertSystemMakeAssessment
  END

Class ExpertSystemsTendNotToMakeGoodJudgement IN STLinkClass
   WITH
  purpose, negative
    csj: CommonSenseJudgement
  how
    les: LetExpertSystemMakeAssessment
  END
```

A routine is a specialization of a means-ends link, with subroutine as an additional attribute. There is also a constraint that the `purpose` of the routine must be a member of the element refered to by the `how` attribute of the routine.

```
MetaClass RoutineClass IN Class ISA MeansEndsLinkClass
 WITH
  attribute
    subroutine: RoutineClass
  constraint
    purposeOfSubroutineMatchesHow:
      $ (Forall u/RoutineClass)(memberOf( u, this.subroutine ) ==>
          (memberOf(u.purpose, this.how) or
            (Exists t/TaskClass)(memberOf(t, this.how)
                         and Subelement(u.purpose, t) ))$
 END

Class ApproveTreatmentRoutine IN RoutineClass WITH
  purpose
    apv: ApprovalForTreatment
  how
    at: ApproveTreatment
  subroutine
    lcr: LetClaimsClerkAssessRoutine
 END

Class LetClaimsClerkAssessRoutine IN RoutineClass WITH
  purpose
    ta: TreatmentBeAssessed
  how
    lc: LetClaimsClerkAssess
 END
```

A rule is a specialization of a means-ends link, with the added attribute of applicability condition.

```
Metaclass Rule IN Class ISA MeansEndsLinkClass
  WITH
 attribute
   applicabilityCondition: Assertion
 END

Class CanDoMedicalAssessment IN Rule
  WITH
 purpose
   m: MedicallyAssessed
 how
   do: AssessTreatment
 applicabilityCondition
   hasKn:  $ HasMedicalAssessmentKnowledge(this.do.actor) $
 END
```

For analysis (to be discussed in subsequent sections), we need to have access to the collection of routines, rules and primitively workable elements that an actor has. Thus the actor construct in the SR model needs to have additional attributes beyond those in the SD model. These are defined in the `ActorSRClass` metaclass.

```
Metaclass ActorSRClass ISA ActorClass WITH
  attribute
    routine: RoutineClass
    rule: Rule
    pwe: ElementClass
 END
```

3.3 Process Modelling Using the Strategic Rationale Model

The Strategic Rationale model provides a rich set of concepts for modelling processes and the reasoning behind them. In this section, we discuss the expressiveness of the SR model, and how it provides a deeper understanding in comparison to existing modelling frameworks.

Understanding "Why" and "How". Conventional process models that view a process as activities with flows between them (such as the workflow model of Figure 1.1 in Chapter 1) do not capture the "why" dimension of a process. For example, the process described in Figure 3.1 could be described as consisting of the three steps: `VerifyPatientPolicy`, `MakeMedicalAssessment`, and `SignApprovalDocument`. Such a description offers a non-intentional view of a process, leaving out the motivations and rationales behind the process, and does not encourage the consideration of alternatives.

By offering means-ends links, the SR model provides a view of process that is goal-oriented. By being agent-oriented (in conjunction with the SD model), the i^* framework conveys where these intentional forces are coming from, and directed towards. At any node, when one asks a "how" question a means is sought with the current node being its desired end. The goal that `TreatmentBeAssessed` can be accomplished by the claims manager doing `AssessTreament`, or via `LetClaimsClerkAssessTreatment`. Conversely, when one can ask a "why" question – one seeks to discover the end for which the current node is the means. One can also ask a "how-else" question, by first asking the "why" question. By being able to express why and how, the model gives a deeper understanding based on means-ends reasoning. One can see that there are alternatives, and that actors have choice. One can thus better anticipate the implications of change.

Task decomposition and composition. Many modelling schemes have incorporated the composition/ decomposition dimension so that descriptions of processes (or other types of objects) can be hierarchical (as opposed to flat). The composition dimension is the main organizing mechanism in structured analysis (e.g., SADT) and has been found to be valuable in modelling large systems. With a hierarchical decomposition dimension, the example of Figure 3.1 could be described in levels of detail: `ApproveTreatment` as a top-level, decomposed into `AssessTreatment` and `SignApproval` at the next level, the first of these decomposing into `VerifyPolicy` and `AssessMedical`, and so forth.

The SR model extends this by allowing task decomposition to include different types of components, not just a decomposition of activities into sub-activities. In a non-intentional context, activities are merely "carried out". There is no notion of success or failure, or goal-achievement by different means. Under SR, we assume an intentional, strategic modelling context. Thus we want to distinguish among task components that differ in the degree of openness or uncertainty (and thus the extent of problem solving that may be needed to further pursue it). A goal means it is expected there can be different ways of achieving it (alternatives). A task means there are constraints on how to perform it. A resource means it is assumed to be unproblematic. It is also important to be able to associate quality concepts which constrain the selection among alternatives. At the bottom of the means-ends hierarchy in the SR model (i.e., not further reduced as far as the SR model is concerned), task elements can still be goals or tasks (i.e., they need not be primitive actions, and need not be fully constrained). Process execution typically would require further problem solving at run-time. For this reason, there are often softgoals remaining as bottom nodes in the SR model, to guide and constrain choice among alternatives at run-time. Bottom nodes in SR often serve as "intentional interfaces" between actors, i.e., as dependums in the SD model.

Benefiting from use of a Knowledge-Based approach. By adopting a knowledge-based approach, the SR model (as in SD) acquires the knowledge structuring dimensions of classification, generalization, aggregation, and time. Earlier modelling languages (e.g., RML [Greenspan82, 84] and ERAE [Dubois86]) have developed these. Some of these facilities have also been adopted by recent commercial object-oriented analysis frameworks (e.g., [Coad90]; see discussion in [Greenspan94]). Beyond providing organization for the knowledge when doing one model, a crucial benefit is the ability to make use of knowledge across many models, (e.g., from other projects and also from neighbouring domains) to compare and detect similarities, and so forth. Libraries of knowledge can be made available for browsing and for selecting from. They can encode past case experiences, as well as generic knowledge (through the use of generalization/ specialization hierarchies).

The power of object-oriented, knowledge-based approach to modelling is further enhanced by the addition of the means-ends relationship dimension in the SR model. Means-ends links provide yet another dimension for structuring knowledge, and thus for making use of means-ends knowledge across cases. Means-ends links are seen as applications of generic rules. During modelling, the construction of means-ends links can be supported by the application of rules. A rule is a generic means-ends relationships which would be suggested when its applicability condition is found to hold. Rules can be used in both the forwards and backwards directions.

- Given means, suggest some possible ends. This supports the asking of "why" questions during model construction.
- Given ends, suggest some possible means. This supports the asking of how questions. (See example in section 3.1.)

Rules are used primarily to suggest possibilities to the human user of this tool, who has to decide whether any of these indeed accurately models the world.

The framework of [Lee93] (also in [Malone93]) includes goals, but there is no clear separation of goals from activities, and the use of means-ends reasoning is not formally characterized.

Process Analysis and Design Activities. Besides being more expressive (and thus providing a deeper intuitive understanding for the modeller and model user), the SR model offers assistance in a number of analysis and design activities which can be computer-supported, taking advantage of the formal representation of the SR model.

In analyzing a process (as expressed in an SR model), one can examine and trace the network of linkages including means-ends links, task-decomposition links, and their connections to SD models (as well as knowledge structuring links that are part of the underlying knowledge representation facility, e.g., classification, generalization, etc.) More importantly, one can do analysis that is of strategic concern at the actor level (as opposed to at the node and link level of task elements): whether an actor knows how to do something, whether it will work, how well it will work, and why the agent believes it will work. All of this is from the stand point of the strategic interests of each agent.

In design, one can systemically explore alternatives, by seeking means to ends. Analytical support can be used to guide and evaluate alternatives as they are proposed. The technical details to support these activities are presented in subsequent sections of this chapter.

Some qualifications. 1. The i^* framework is aimed at modelling strategic relationships and reasoning. Such knowledge is not expected to be complete. Computer support for such reasoning is therefore much weaker than in the support of operational reasoning (e.g., in classical AI problem solving or planning). The components in a task decomposition is not assumed to be a complete list, but only those that are strategically significant, as judged by the actor (and as recorded by the modeller). In strategic reasoning, one does not expect to have hard and fast conclusions that have guaranteed outcomes (in contrast to conventional logical inference), nor to have quantitatively optimized solutions. A dialectic reasoning scheme (also called argumentation, issued-based information system) approach that has been used in design rationales support frameworks (e.g., [Potts88]) is therefore adopted. Qualitative reasoning techniques (e.g., as in [Chung93]) are also appropriate.[1]

Unlike problem solving in AI, the objective is not to automatically generate some solution, so there is no automatic chaining of means-ends links. Each step requires the judgement and input of the modeller, as in other rationale modelling frameworks. The means-ends relationships being modelled may not match (be instance of) any of the existing rules. In this case, the modeller could still assert the link, as an instance of some new class. It can be used as a rule in the future if a generic applicability condition can be stated.

[1]Exploration of what AI planning techniques can be applicable to Strategic *planning* is future work, i.e., where there are temporal constraints on strategic *actions*. Strategic action and therefore planning have not been considered in this thesis research. The SR model supports reasoning about the relative merits of various strategic configurations, but not about the actions for going from current configuration to proposed configurations.

2. The reasoning that is modelled need not be the actual (historical) reasoning that led to the process. It can be an "*a posteriori* rationalization", reflecting the perceptions of those participants involved in the current reasoning (use of the model).

3. These representational constructs can be used in different ways. Further guidance in the usage of these constructs in the form of methodologies may be necessary. The methodology may recommend or stipulate performing modelling steps in a particular order. The development of methodologies for particular usage contexts is beyond the scope of this thesis (but see Chapters 4 to 7 for the potential application of the framework in four different usage contexts). In this section we have enumerated some of the basic modelling activities that are supported by the modelling framework, from which one can construct methodologies for various usage contexts.

Extended Example. As further illustration of the expressiveness of the SR model, we now show how the reasoning behind the different health care arrangements of Section 2.4 can be expressed in an SR model. In Figure 3.5, we have an SR model for the managed indemnity case, where a physician needs to get pre-approval for giving treatment to a patient, in order to receive payment for the treatment subsequently.

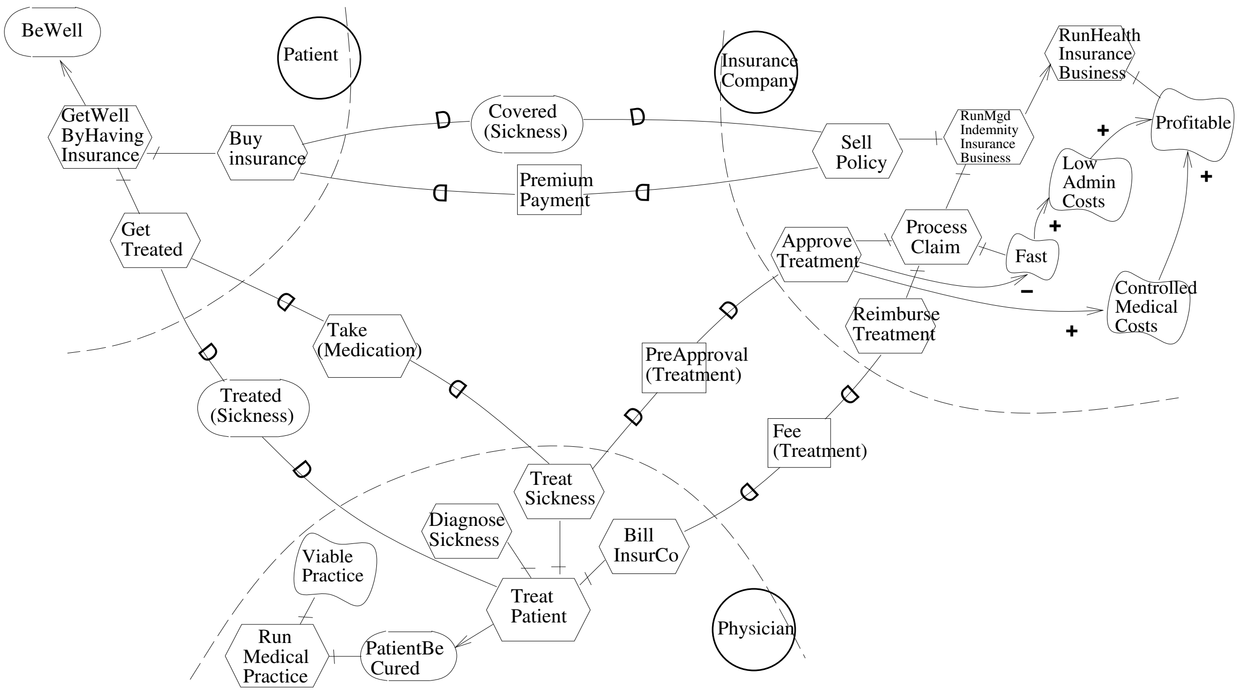

Figure 3.5: Modelling the reasoning behind one health care configuration – "managed indemnity insurance"

On the patient side, the overall goal is to be well. One way to be well is by buying insurance and getting treatment when sick. The patient depends on a physician to get treated, and on an insurance company to cover expenses incurred during sickness.

For a physician, running a medical practice includes the subgoal that patients be cured and the softgoal that the practice be viable. There may be different ways to achieve the goal that patients be cured. In the process modelled in Figure 3.5, treating a patient involves diagnosing the sickness, treating the sickness, and billing the insurance company. Treating sickness depends on receiving preapproval of the treatment from the insurance company, and on the patient to take prescribed medication.

For a for-profit insurance company, an important concern in running a health insurance business is that it be profitable. The managed indemnity arrangement is one way of running a health insurance business. It involves selling policies to patients (and depending on them for premium payments), and processing claims. Claims processing includes pre-approving treatments, and then reimbursing physicians once the treatment is completed. The need to pre-approve treatment contributes negatively to the desire for fast claims processing, which is important for lowering administrative costs. On the other hand, pre-approval of treatment contributes positively to controlling medical costs, which contributes towards the profitability of the insurance business. These arguments can be handled systemically using a qualitative reasoning scheme (e.g., as in [Chung93]).

The modelling power of the SR model is more apparent when it is used to reason about a number of alternatives. In the health care setting, patients want health plans to be affordable, yet provide the peace of mind that comes with good coverage and quick access to treatment when needed. Physicians want their practices to offer effective medical treatments, and be economically viable at the same time. Insurance companies want to improve profitability by controlling medical costs and lowering administrative costs.

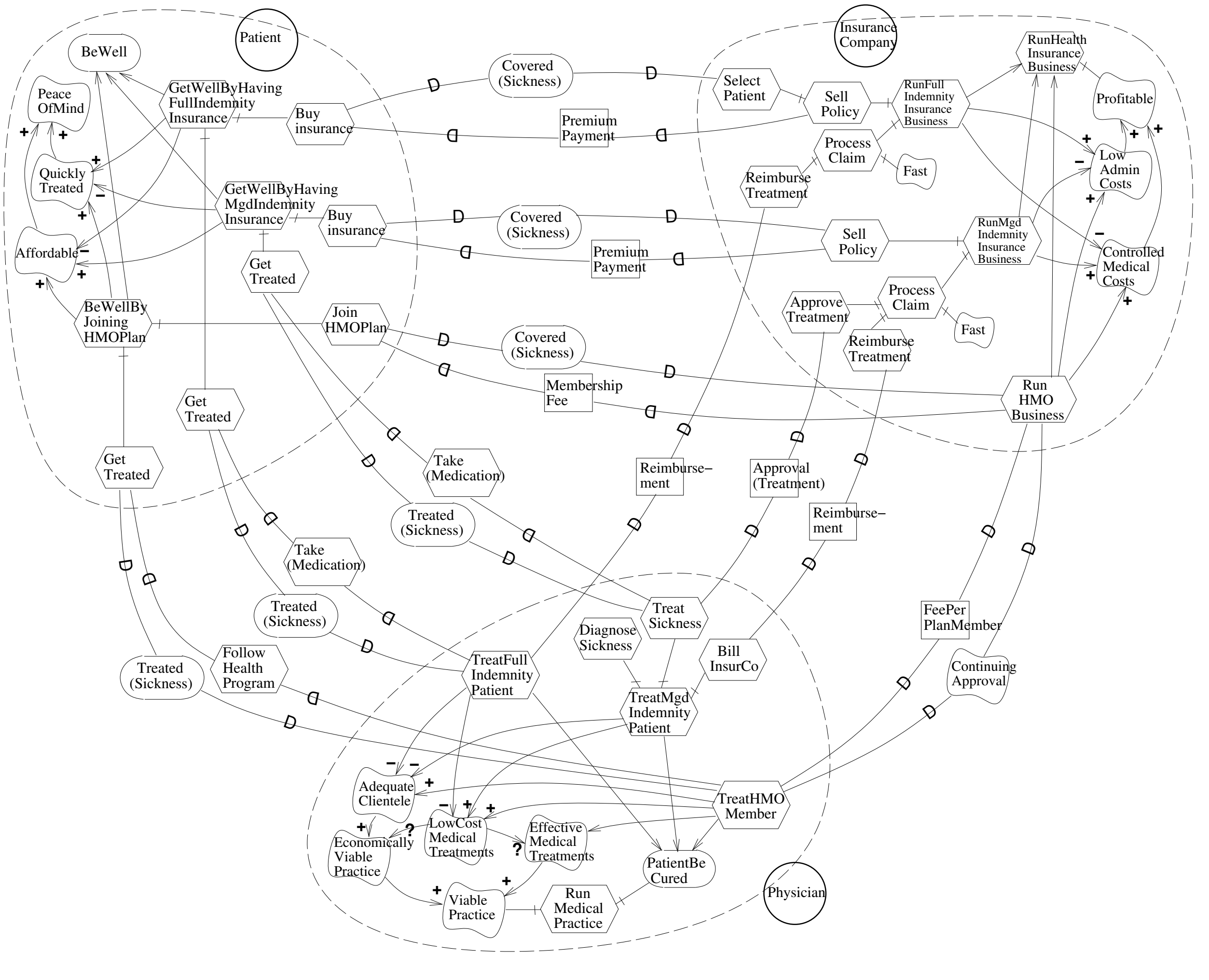

Figure 3.6: Modelling the reasoning about three alternative health care configurations – "full indemnity", "managed indemnity", and "managed care"

Any new proposals for health care reform would be judged by each party (stakeholder, strategic actor) in terms of how their own interests and concerns are affected. Figure 3.6 shows all three configurations discussed in Chapter 2 in a single SR model, so that one can see how the process compositions of the three are related, and how they address the concerns and interests of the three parties.

For the patient, full indemnity is good for receiving treatment quickly (no waiting for pre-approval), but bad for affordability – insurance companies have to charge more because medical costs cannot be controlled. Administrative costs are low, because claims processing is fast. Physicians can offer effective medical treatment unhampered by insurance company pre-approvals. However, the volume of clientele may suffer due to the difficult affordability of this type of insurance plan.

The third alternative – managed care – also has its pros and cons for each of the three parties. The patient can get treated quickly, and the plan is more affordable, although the choice of health care provider is restricted (not shown). The insurance company has a much reduced administrative cost by eliminating claims processing. Also, the burden of controlling medical costs has been shifted to the physician. The physician gets a steady clientele, but the effectiveness of the medical treatment under such a scheme is inconclusive, since it is not known whether lower cost medical treatments conflicts with effective medical treatment.

We have illustrated these modelling features informally in this section. In the next section (analysis), we describe some technical concepts for supporting analysis in the SR model. In the section on design, we outline technical concepts for assisting the systematic generation of alternatives in the search for process designs that are more agreeable to all stakeholders.

3.4 Process Analysis Using the Strategic Rationale Model

The SR model offers analytical support at the level of nodes and links, as well as at the level of actors.

The SR model supports analysis on its structural elements, using the query language of the underlying knowledge representation facilities (in this case, Telos). One can pose queries to the model to determine, e.g., what resources does a task require, what are the means for a given end, what are the softgoals relevant to a task, and so forth. One can follow up and down the means-ends hierarchies, and across actor boundaries along dependency links in the related SD models.

Because of the intentional semantics of the SR constructs, this type of analysis can give a richer understanding about the "whys", "hows", and "how-elses" and "what-ifs" on a process than conventional, non-intentional process models, where the types of analyses are primarily limited to a matching of inputs and outputs, and checks for consistency and completeness.

More importantly, beyond these basic analytical capabilities, the model can provide higher level, more aggregate types of analysis. In strategic analysis, one would like to determine whether a process (existing or proposed) addresses the interests and concerns of each strategic actor. We define several additional concepts to augment the analytical power of the SR model, aimed at answering the following questions:

1. Does the actor have a process for accomplishing the goal? – we call this **ability**.

2. Is the process going to work? – we call this **workability**.

3. How well will it work? – we call this **viability**.

4. What evidence are there to confirm or disconfirm that it will work? – we call this **believability**.

These concepts are defined as the Telos predicates `Able`, `Workable`, `Viable`, and `Believable` respectively. They may be evaluated by issuing `ASK` queries.

Ability. In the SR model, the main integrating concept is that of the routine, which treats a collection of nodes and links together as serving some purpose at the actor level. These connect up with the dependency links in the SD model to make up a process description involving multiple interdependent actors. In our formulation, an actor has the ability to do something if it has a routine for it. In the example of Figure 3.1, we say that `ClaimsManager` has the ability to provide `ApprovalOfTreatment` to `Physician` because it has the `ApproveTreatment` routine.

Having ability does not necessarily imply that one can achieve something all by oneself. Various degrees of delegation and external dependency may be involved. `ClaimsManager` has the ability to achieve `MedicallyAssessed`, since she has routines (two in this case) for addressing it. In one of these, she delegates the entire goal to `MedicalAssessor` (but is still "responsible", or else the model would have a direct dependency link from `Physician` to `MedicalAssessor`, skipping the `ClaimsManager`.) In the other, she does the assessment herself, but still depends on others for `MedicalClaimsPrecedents` and `PatientMedicalFiles`.

This usage of the term "able" is perhaps weaker than what is implied in the general English usage of the term. Recall that for strategic modelling and reasoning, we want to have a notion of process which does not require full reduction to the minute details of "primitive actions" (unlike the notion of plans in operational settings). Thus we would like to be able to say that an actor knows of a way to do something, without necessarily implying that the actor is able to carry it out. (For example, an actor may need to know how to do something, in order to be able to tell a dependee how to do it, but still not be able to do it himself.)

Workability. The notion of "workability" is introduced to provide a second level of analysis, beyond the basic notion of ability. An actor having a routine (and thus ability) means that it knows what to do to the extent that the routine is reduce partially to some elements. But whether the actor believes it is able to successfully carry out or achieve these elements is a separate matter, and is captured in the notion of workability.

We say that an element is **workable** if the actor believes (at "process design time") that it can successfully carry it out or achieve it (at "run-time"). An element in a routine is workable if the elements in all of its subroutines are workable. This recursive evaluation terminates either at the actor boundary, where there is a dependency, or at an element which the actor considers to be **primitively workable**. This means that the element is judged to be not worth further pursuing/reducing during strategic modelling. The actor is confident that, at execution time, it will be able to carry out the reasoning and actions required to achieve the result.

For example, there is a routine for approving treatment involving `LetClaimsClerkAssessTreatment`. This makes `ClaimsManager` "able" to approve treatment. On further analysis, suppose the claims manager believes that claims clerks do not have adequate medical assessment knowledge (failure of the Workability Commitment Assumption), rendering the task `LetClaimsClerkAssessTreatment` unworkable. This would make the routine containing this task unworkable.

On the other hand, suppose the routine containing the branch `AssessTreatment` with `LetMedicalAssessorMakeMedAssessment` is workable, then the element `ApproveTreatment` is still workable, since there is a workable routine for achieving it.

At an actor boundary, an open element is workable if there is some actor offering this element. A committed element is workable if there is another actor committed to producing this element as dependum to this first actor.

The concept of workability is characterized more precisely in section 3.6.

Workability analysis does not provide any special treatment for softgoals. Softgoals are ignored in the evaluation of workability. The qualitative reasoning dimension of softgoals is brought in only in the analysis of viability.

Viability. The notion of viability provides a third level of assessment, based on a qualitative assessment on how well the softgoals in a routine are met. This provides a finer-grained evaluation than workability, which is based only on a binary evaluation on the elements of a routine.

A qualitative reasoning framework employing a notion of satisficing has been developed [Chung93]. Using a multi-valued evaluation scheme and labelling procedure, an assessment of the qualitative attributes of some system can be obtained from the assessment on the elements of the system. The framework was originally developed as a way for systematically addressing non-functional requirements during software development. Here, we generalize the concept of non-functional requirements to that of softgoals in process modelling.

We say that a routine is viable if all its softgoals are satisficed. Softgoals at the top level are selectively applicable to elements at lower levels in the functional means-ends hierarchy, as selected by the means-ends hierarchy of softgoals, via the parameters in the softgoal nodes. For example, in Figure 3.1, if `FastTurnaround` is not satisficed, then `AssessTreatment` via `LetMedAssessorMakeMedAssessment` is not viable, even though it may be workable.

In the example of Figure 3.6, asking the viability of `RunHealthInsuranceBusiness` would trigger the evaluation of the softgoal `Profitable`, which triggers evaluation of its contributing elements `LowAdminCost` and `ControlledMedicalCost`, and recursively their contributing elements. In a more detailed example, there would probably be other softgoals besides profitability, such as market share, public image, etc. These may reduce to other softgoals such as CustomerSatisfaction and so forth. The network of softgoal links will likely have cross-correlations.

Believability. Since the SR model relies on judgement and argumentation, there are many assumptions made as the model is constructed. A fourth level of assessment is in the believability of these assumptions. A qualitative treatment of these is also assumed. Belief nodes are known as "argumentation goals" in [Chung93].

This approach of dividing the analysis into several levels allows finer-distinctions and offers greater flexibility in modelling. An actor may be able (has a routine), without the routine necessarily being workable. A workable routine may not be viable, and a viable routine may not be believable. The modelling is more flexible because the more sophisticated levels can be arrived at incrementally (following some methodology). Pragmatically, each deeper level involves knowledge that is potentially harder to get at, requiring more effort, but increases the level of confidence. Furthermore, these properties may be shared differently across actor classes in a generalization hierarchy. For example, ability tends to be more easily sharable, whereas workability is more specific to particular actors or actor classes.

The Extended Example, Continued. In the extended example of Figure 3.6, one could analyze the three main alternatives in terms of ability, workability, viability, and believability, from each stakeholder's perspective.

The insurance company would analyze its policies and practices to see whether and how well they meet the objectives in running a profitable health insurance business.

Ability: – Do we have a routine for approving treatment? and for selling policy?

Workability: – Do we have the skills and resources to address all the elements in routine? Further reduction would be necessary if the workability of these elements cannot be judged.

Viability: – Are the softgoals of low administrative costs and controlled medical costs satisficed?

Believability: – Does `PreApprovedTreatment` really lower medical costs? A counter argument might be: a disallowed medical procedure may actually have preventive benefits; not allowing it may worsen the patient's condition, thus incurring higher costs later.

Similarly, the patient wants to analyze each health care configuration in terms of whether he is able to get cured (be well); whether the process of his being cured is workable, whether the softgoals (affordable, fast, offering peace-of-mind) are satisficed, and whether the reasoning about this process is believable.

3.5 Process Design Using the Strategic Rationale Model

Process design is concerned with thinking up new ways of doing things (reorganizing the process). In modelling, the SR graph is constructed and elaborated to reflect conditions in the world as it exists, or to express alternatives that have already been developed. In design, the SR model is used (in conjunction with the SD model) to help systemically explore new ways of doing things and reorganizing.

The framework supports design in terms of five categories of activities:

1. raising issues,
2. addressing issues,
3. identifying related issues to raise,
4. settling issues and accepting assumptions, and
5. identifying assumptions to question and justifying them.

3.5.1 Raising Issues

In modelling known processes, goals are recorded as such, means-ends links representing solutions to those goals are recorded. There are no attempts to solve (or re-solve) those goals. In analysis, one determines whether goals are (adequately) addressed. Again, there is no attempt to solve them. In design, however, one *does* want to change the process structure (find new structures) in order to *address* the issues. Even so, during a particular design effort (or redesign effort, if there is an existing process), not all the issues that appear in an SR model are to be raised (or not all at the same time). Typically, intentional elements (goals, tasks, resources, and/or softgoals) that are not workable or not viable (as determined by analysis) are candidates for raising as *issues*. However, whether an intentional element should indeed be raised is still a matter of judgement and choice by the user. For example, a user could decide that there are too many of these, so only a subset is raised in order to obtain some initial solutions. Conversely, elements that are workable and viable can still be raised – for example, in the hope of even better solutions.

The SR model provides representation for distinguishing between raised and unraised goals. In the Telos notation, we use the predicate `Issue` to indicate that the object is a raised goal (issue). We use the term *issue* to avoid confusion with the term goal, which is used to refer to (one type of) intentional element in the process model. An issue is a *design goal*, which is *addressed* during process design by constructing or altering process structures (which are made up of intentional elements – goals, tasks, resources, and softgoals).

Some examples of issues are:

- ability as an issue, e.g., `Issue(Able(ApprovalOfChiroTreatment))` – this would lead to a search for a routine which can deal with approving a (request for) chiropractic treatment
- workability as an issue, e.g., `Issue(Workable(ApprovalOfChiroTreatment))` – this would lead to a sequence of design steps which attempts to make the approving of chiropractic treatment workable.
- viability as an issue, e.g., `Issue(Viable(ApprovalOfChiroTreatment))` – this might include making the approval process faster and more cost effective.

The selective raising of issues can be used to limit the scope of a process redesign effort. In a smaller scope effort, one would avoid raising issues that are high up in some means-ends hierarchy. For example, one might try to find faster ways for claims processing, but avoiding raising the issue of how else to run a health insurance business (which could lead to new process designs that eliminate claims processing altogether). Some proponents of business process reengineering (see Chapter 6) advocate radical change for dramatic improvements. In the SR model, this would mean raising issues (asking "why") at very high levels in the means-ends hierarchy, and then seeking new solutions to those very high-level goals. The setting of scope in an redesign effort has important impacts on stakeholder interests (e.g., employees and departments who do claims processing), and therefore is an important area to be covered in a methodology for applying the framework.

Believability can also be raised as an issue. We deal with this in a separate subsection below (3.5.5), because there is special support for identifying assumptions to raise for questioning.

3.5.2 Addressing Issues

To address an issue, we look for (alternative) routines through means-ends reduction to process elements that are refined enough to be either accomplishable by the actor itself, or can be delegated to another actor via an outgoing dependency. Means-ends rules can be used to help in identifying means to ends. The process designer has to make judgements whether the applicability conditions for those rules hold in the anticipated new environment, and to justify them in terms of beliefs.

Workability provides a simple first-cut criteria for evaluating proposed designs. Viability assessment provides additional guidance on how to generate alternatives, and how to select among them.

- When addressing `Issue(Able(ApprovalOfChiroTreatment))`, using a rule which says `ChiroTreatment` IS-A `ExcludedTreatment` (excluded treatment are those that are not covered by any of the insurance companies policies, and therefore can be rejected without verifying patient policy, vs. uncovered treatment). A rule may say to approve ExcludedTreatment, you need the resource `ExcludedTreatmentList`. Thus one reduces `Issue(Able(ApprovalOfChiroTreatment))` to `Issue(Able(ExcludedTreatmentList))`, which one then tries to address.

- In addressing `Issue(Workable(ApprovalOfChiroTreatment))`, when one alternative is found to be unworkable, one could explore another alternative.
- In addressing `Issue(Fast(ApprovalOfChiroTreatment))`, one might invoke methods, which says: try to reject request for `ExcludedTreatment` at the earlier possible step in the process.

3.5.3 Identifying Cross-Impacted Issues to be Raised

Proposed process changes may induce cross-impact on other concerns. In particular, softgoals frequently have contributions to several other softgoal. For example, pre-approval reduces medical costs, but incur administrative costs, and slow down treatment for patients. These issues, though not originally raised, need to be considered for raising because they are cross-impacted by the issues that were explicitly raised. These cross-impacted issues therefore should be taken into account when searching for and evaluating new alternatives.

For example, in addressing `Issue(Fast(ApprovalForChiroTreatment))`, a correlation rule would suggest to raise `Cost(ApprovalForChiro)` as an issue as well.

3.5.4 Settling Issues

Issues that are no longer actively being addressed need to be marked as settled. In the Telos representation, we simply reverse the `Issue` predicate to false. Because of the presence of a historical record of temporal information in Telos, issues that have been raised and subsequently settled can be distinguished from issues that never been raised.

Some issues may be settled before others. Typically, issues that have become able, workable, viable, and/or believable are candidates to be considered as settled. Again, it is up to the user to make a judgement.

3.5.5 Identifying Assumptions to Question

We use the term *assumption* to refer to a belief whose believability is raised as an issue. Assumptions need to be "addressed" by providing justifications for them. Unlike process elements (goals, tasks, resources, softgoals) that are raised as issues, the addressing of an assumption does not alter process alternatives, but adds to or modifies the network of rationales that justify the assumption. (This is the difference between goal and belief). An assumption is "settled" when it is accepted, i.e., is a belief that no longer needs further justification.

The $\boldsymbol{i}^*$ framework provides support for identifying certain types of assumptions for questioning, based on the semantics of the modelling concepts.

i. rule applicability

   When a means-ends link is an application of a rule, there should be evidence to support the belief that the applicability condition of the rule holds. For example, if there is a means-ends link that is an application of the rule `DelegateAssessTreatment`, its applicability condition `DelegateeHasAssessmentKnowledge` should be a supported belief. An assumption that is raised is represented in Telos as an "issue" on the believability of that assumption: e.g., `Issue(Believable(DelegateeHasAssessmentKnowledge))`.

ii. task constraints

   In the modelling of a task, there may be constraints among task elements. For example, the task `ApproveTreatment` has the constraint that the subgoal `TreatmentBeAssessed` be achieved before the subtask `SignApprovalDocument` is carried out. When such constraints appear in a routine, they need to be justified (i.e., there needs to be evidence to support the beliefs that the constraints will hold when the routine is carried out.

iii. matching open depender-dependee

The presence of an open dependency on the depender side would suggest the need for the availability of a dependee. Suppose the claims manager dependency on the `ClaimsCasesRepository` for `MedicalClaimsPrecedents` is supplemented by an open dependency on `MedicalClaimsExpert` (a human expert who have a lot of experience in dealing with these cases) for same, then one should look for evidence that a `MedicalClaimsExpert` is available as an open dependee with `MedicalClaimsPrecedents` as dependum.

iv. matching committed depender-dependee

If a depender has a committed dependency on some dependee, then the dependee should have a committed-depended-on by the depender. The `ClaimsManager` has committed dependency on `MedicalAssessor` to have `MedAssessed(Claim)`. So `MedicalAssessor` should have a committed dependency from the `ClaimsManager` on the same dependum.

v. the Workability-Commitment Assumption

If `MedicalAssessor` is committed (to someone) to achieve `MedAssessed(Claim)`, then it is assumed that it has a workable routine to do so. This assumption needs to be supported.

vi. the Workability-Transfer Assumption

If `MedicalAssessor` is committed to someone (`ClaimsManager` in this example) to achieve `MedAssess(Claim)`, and the achieving of this dependum is workable for `MedicalAssessor`, then it is assumed that the dependum `MedAssessed(Claim)` will become workable for the depender (`ClaimsManager`).

3.5.6 Justifying and Accepting Assumptions

Each assumption can have positive and negative support. For example, the workability-commitment assumption above may have a positive support from the belief `MedicalAssessorIsCertified`, and a negative support from the belief that `MedicalAssessorsHasBeenUncooperative`. Each of these can in turn be raised for further questioning. A number of dialectic, argumentation frameworks (or issue-based information system, design rationale frameworks) have been proposed for managing this types of argumentation structures, including [Chung93].

The addressing of assumptions can be done by reducing an assumption to other, more fundamental assumptions, until a belief that is not raised as an assumption is reached (or an assumption that has been settled (accepted). The reduction of assumptions can be supported by belief-reduction rules. These are supported as "argumentation methods" in [Chung93].

3.6 Formal Characterization

In this section, we consider the formal characterization of the concepts of the Strategic Rationale model. As in section 2.4, the axioms are intended to provide a sharper characterization of the concepts than the informal presentation in preceding sections, and to serve as a basis for building tools to support the framework.

We use the variable η to denote an intentional element. Where a distinction among different types of intentional elements are needed, we use *g, t, r,* and *s* to denote goal, task, resource, and softgoal respectively. A task decomposition link is denoted by the predicate $el(\eta, t)$, linking element η to the task t. The different types of TDlinks are denoted as *subgoal(t,g)*, *subtask(t, t')*, *resourceFor(t,r)*, and *softgoalFor(t,s)*. The predicate $cx(t, \kappa)$ indicates that κ is a constraint that applies to task t. The predicates $oel(x, \eta)$ and $cel(x, \eta)$ are used to indicate open and committed task-decomposition links respectively.

A means-ends link is represented by the predicate $mel(l, \eta, u)$ where l refers to the link, routine u is the means, and intentional element η is the end.

The repertoire of routines that an actor x has is denoted by $\mathcal{U}_x$. That a routine u is in actor x's repertoire is written as $\mathcal{U}_x(u)$. $\mathcal{E}_x$ is the set of primitively workable elements of actor x. The means-ends rules of actor x is denoted by $\mathcal{H}_x(\eta, u, \alpha)$, where η is the purpose, u is the how, and α is the applicability condition.

$\mathcal{B}_x$ is actor x's set of primitive beliefs, i.e., beliefs that need no further justification (analogous to primitively workable elements). The belief-reduction rules (used for justifying beliefs) for actor x are denoted by $\mathcal{J}_x(b, b', \alpha)$ where b' is the support for b, and α is the applicability condition.

A subroutine is also a routine. The *purpose* of a subroutine must match the *how* of the (higher-level) routine, although there can be task-decomposition links in between.

- $$(subroutine(u', u) \wedge purpose(u', \eta)) \supset (how(u, \eta) \vee \exists t(how(u, t) \wedge subel(\eta, t)))$$

where $subel(\eta, t)$ means that η is an element in some decomposition or sub-decomposition of the task t.

- $$subel(\eta, t) \subset (el(\eta, t) \vee \exists t'(el(t', t) \wedge subel(\eta, t')))$$

3.6.1 Process Analysis

An agent x has the ability to achieve η iff it has a routine for doing so.

- Ae: $\quad A(x, \eta) \equiv \exists u(\mathcal{U}_x(u) \wedge purpose(u, \eta))$

A task t is workable if all its elements are workable, and all of its constraints are believed to hold.

- Wt: $\quad W(x, t) \subset \quad \forall \eta(el(\eta, t) \supset W(x, \eta)) \wedge \forall \kappa(cx(\kappa, t) \supset B(x, \kappa))$

The criteria for an element being workable depends on whether it is an open element or a committed element of the task. An open element η is workable if η is an open dependency, or if it is workable under the (stronger) criteria of a committed element. A committed element η is workable if η is primitively workable, or if there is some workable means-ends link linking it to a workable routine, or if η is an outgoing dependency and there is another agent y committed to producing η for x.

- We: $\quad W(x, \eta) \subset (oel(\eta) \wedge W_o(x, \eta)) \vee (cel(\eta) \wedge W_c(x, \eta))$
- Weo: $\quad W_o(x, \eta) \subset (\overrightarrow{D}_{opp}(x, \eta) \vee W_c(x, \eta))$
- Wec: $\quad W_c(x, \eta) \subset \mathcal{E}_x(\eta) \vee \exists l \exists u(mel(l, \eta, u) \wedge W(x, l) \wedge \mathcal{U}_x(u) \wedge W(x, u)) \vee (\overrightarrow{D}_{opp}(x, \eta) \wedge \exists y B(x, \overleftarrow{CD}(y, x, \eta)))$

The intuition behind these is as follows. For an open element, either x knows how to do it (it is primitively workable), or x knows someone who can do it (and therefore x does not feel the need to further elaborate it into a routine). For a committed element, either x knows how to do it, or else x must obtain commitment from someone who can do it, or else x must *further reduce* it through a routine until it is workable.

A routine u is workable if all of the elements specified in its `how` attribute is workable and all of its subroutines are workable.

- Wu: $W(x,u) \subset (\mathcal{U}_x(u) \wedge \forall \eta'(how(u,\eta') \supset W(x,\eta'))$
  $\wedge \forall u'(subroutine(u',u) \supset W(x,u')))$

Thus W can be evaluated recursively.

A means-ends link l with u as the means and η as the end is workable if the agent has a rule for that means-ends relationships and the agent believes the applicability condition α of that rule to hold.

- Wl: $W(x,l) \subset \exists \eta \exists u(\mathcal{U}_x(u) \wedge mel(l,\eta,u) \supset \exists \alpha(\mathcal{H}_x(\eta,u,\alpha) \wedge B(x,\alpha)))$

Viability and Believability are characterized operationally, through the computation and propagation of values in a goal graph, as in [Chung93], and partially formalized in the next sub-section. Viability corresponds to the addressing of "NFR-goals" and "Satisficing goals". Believability corresponds to the addressing of "Argumentation goals".

3.6.2 Process Design

For process design, we need a formal characterization for the notion of issue. We adopt a goal-oriented view of argumentation ([Chung93]), i.e., that an issue to be addressed is treated as a goal, the addressing of a goal is by reduction in a goal graph. The manipulation of the goal graph is interactive, with guidance and support from the framework due to the semantics of the modelling constructs.

We use the predicate $G(x,\phi)$ to denote that ϕ is an issue to be addressed by x. (Note that these are goals to be addressed during process design, which are to be distinguished from the goals, tasks, resources and softgoals that appear as intentional elements in the structural description of routines.) The common types of issues are those that apply to ability, workability, viability, and believability (A, W, V, and B, respectively). In process analysis, we are given a process description and we evaluate that process with respect to A, W, V, and B. In process design, we aim to come up with processes which evaluate to the desired values of A, W, V, and B. We use the following shorthand notation:

$$\dot{A}(x,\eta) \equiv_{def} G(x, A(x,\eta))$$

$\dot{W}$, $\dot{V}$, and $\dot{B}$ are similarly defined.

Because the raising and addressing of issues is an interactive, user-driven process, some of the support are only suggestive (discretionary), while others are mandatory. (A methodology may impose additional mandatory actions than those indicated in the following.) As notation, we use ? as a prefix to an issue to mean that it is only a suggestion to raise that issue.

3.6.2.1 Raising Issues

- $?\dot{W}(x,\eta) \subset \neg W(x,\eta)$
- $?\dot{V}(x,\eta) \subset \neg V(x,\eta)$

If η is not workable (viable) for x, then suggest raising the workability (viability) of η as an issue for x. There is an analogous axiom for believability, which is treated at the end of this section. Note that we do not need an analogous axiom for ability. The absence of ability to achieve something is typically not an issue in itself. According to our framework, as long as there is a routine, there is ability. The issue of interest is usually whether it is workable.

3.6.2.2 Addressing Issues

Ability reduction (reduction to smaller-scope, finer-grained abilities)

- $?(\dot{A}(x,u) \wedge \dot{B}(x,\alpha)) \subset \dot{A}(x,\eta) \wedge \mathcal{H}_x(\eta,u,\alpha)$

If you want x to be able to achieve η, and x has a rule which says that one can achieve η by using routine u, provided applicability condition α holds, then suggest raising x's ability to carry out routine u and the believability of α as issues.

- $?\dot{A}(x,\eta') \subset \dot{A}(x,u) \wedge \mathcal{U}_x(u) \wedge how(u,\eta') \wedge \neg A(x,\eta') \wedge \neg \mathcal{E}_x(\eta')$

If routine u in x's repertoire of routines is raised as an issue and η' is one of the *how* elements of that routine, and η' is not one of x's primitively workable elements, and x is not already able to achieve η', then suggest raising x's ability to achieve η' as an issue. Note that means-ends reduction is by applying rules to ability, not to workability.

3.6.2.3 Identifying Related Issues to Raise

There are many issues that should be raised as a result of other issues having been raised. The $\boldsymbol{i}^*$ framework provides support for suggesting (or requiring) these.

(i) identifying related ability issues As mentioned in the above, $\dot{A}(x,\eta)$ is usually not an issue in itself, but would be an issue if $\dot{W}(x,\eta)$ has been raised as an issue.

- $?\dot{A}(x,\eta) \subset \dot{W}(x,\eta) \wedge \neg A(x,\eta)$

If you want η to be workable for x, and (yet) x is not (even) able to achieve η (i.e., there are no routines for achieving η), then suggest raising x's ability to achieve η as an issue.

(ii) identifying related workability issues.

- $\dot{W}(x,u) \subset \dot{V}(x,u)$

If you want routine u to be viable, then you also want it to be workable.

- $?\dot{W}(x,u) \subset \dot{W}(x,\eta) \wedge \mathcal{U}_x(u) \wedge purpose(u,\eta)$

If you want η to be workable, and there is a routine u in x's repertoire, and the purpose of u is to achieve η, then suggest raising the workability of u as an issue. Note that there can be more than one routine u that matches. The user can choose which to raise, or in what order to raise and address.

- $\dot{W}(x,\eta) \subset el(t,\eta) \wedge \dot{W}(x,t)$

If you want a task to be workable, you would want its elements to be workable.

Note that these axioms have counterparts in the analysis section. Here since we are trying to solve rather than to evaluate, the axioms tend to be in the "reverse direction". Goal node generation is downstream. Propagation of value is upstream. Some of these are mandatory while others are discretionary.

(iii) identifying related viability issues

- $?\dot{V}(x,u) \subset \dot{V}(x,\eta) \wedge \mathcal{U}_x(u) \wedge purpose(u,\eta)$

If you want η to be viable, and there is a routine u in x's repertoire, and the purpose of u is to achieve η, then suggest raising the viability of u as an issue. This is directly analogous to the workability case.

- $G(x,s) \subset \dot{V}(x,u) \wedge \mathcal{U}_x(u) \wedge how(u,t) \wedge softgoalFor(t,s)$

From evaluation axiom for viability, all softgoals can be raised without asking.

(iv) raising viability concerns across actor boundaries These are derived from the SD model, based on the degree of strength of a dependency – Open, Committed, or Critical.

- $?\dot{V}(x, \overleftarrow{D}(y, \eta)) \subset \quad \dot{V}(x, O\overrightarrow{D}(x, \eta)) \wedge \overleftarrow{D}(y, \eta)$

If the viability of x's open dependency on η is an issue, and y is an open dependee on η, then suggest raising as an issue the viability of y's offer as an open dependee.

- $\dot{V}(x, C\overrightarrow{D}(x, y, \eta)) \subset C\overrightarrow{D}(x, y, \eta)$

If there is committed dependency from x to y on η, then raise the viability of that dependency as an issue for x.

- $\dot{V}(x, C\overleftarrow{D}(y, x, \eta)) \subset \dot{V}(x, C\overrightarrow{D}(x, y, \eta))$

If the viability of x's committed dependency on y is an issue, raise y's being committed depended on by x as an issue.

- $\dot{V}(y, C\overleftarrow{D}(y, x, \eta)) \subset C\overleftarrow{D}(y, x, \eta)$

If y is committed depended on by x for η, raise the viability of that dependency as an issue for y.

- $\dot{V}(x, X\overrightarrow{D}(x, y, \eta)) \subset X\overrightarrow{D}(x, y, \eta)$

If x critically depends on y for η, raise the viability of that dependency as an issue for x.

- $$\begin{aligned}\dot{V}(x, X\overrightarrow{D}(x, y, \eta)) \supset\ & \dot{V}(x, C\overleftarrow{D}(y, x, \eta)) \\ & \wedge \forall \eta' \forall z (C\overrightarrow{D}(y, z, \eta') \supset \dot{V}(x, C\overrightarrow{D}(y, z, \eta'))) \\ & \wedge \forall \eta' \forall z (X\overrightarrow{D}(y, z, \eta') \supset \dot{V}(x, X\overrightarrow{D}(y, z, \eta')))\end{aligned}$$

If the viability of x's critical dependency on y for η is an issue, then raise the viability of y's being committed depended on by x as an issue, and for all of y's further committed and critical dependencies on other agents, raise the viability of those dependencies as issues for x (N.B. not (just) for y, because x is also vulnerable).

3.6.2.4 Settling Issues

The settling of issues are suggested when they have been adequately addressed (evaluate to desired values (e.g., the value "satisficed" in the framework of [Chung93])).

- $?settled(\dot{V}(x, \eta)) \subset \dot{V}(x, \eta) \wedge V(x, \eta)$
- $?settled(\dot{W}(x, \eta)) \subset \dot{W}(x, \eta) \wedge W(x, \eta)$
- $?settled(\dot{A}(x, \eta)) \subset \dot{A}(x, \eta) \wedge A(x, \eta)$

3.6.2.5 Questioning Assumptions and Justifying Them

Reducing assumptions to more basic assumptions.

- $?(\dot{B}(x, \alpha) \wedge \dot{B}(x, b')) \subset \dot{B}(x, b) \wedge \mathcal{J}_x(b, b', \alpha)$

If b is an assumption to be justified, and there is a belief reduction rule that reduces b to b' (i.e., b is believable if b' is believable), then raise the believability of b', as well as the application condition of the rule, as issues to be addressed.

Other beliefs that need to be raised as assumptions to be justified include:

(i) constraints among task components

- $\dot{B}(x, \kappa) \subset \dot{W}(x, t) \wedge cx(t, \kappa)$

If κ is a constraint in task t, and the workability of t is an issue, then raise the believability of κ as an issue.

(ii) rule applicability conditions

- $\dot{B}(x, \alpha) \subset \dot{W}(x, l) \wedge mel(l, \eta, u) \wedge \mathcal{U}_x(u) \wedge \mathcal{H}_x(\eta, u, \alpha)$

If the workability of the means-ends link l is an issue, then raise as an issue the believability of the applicability condition of the rule of which the means-ends link is an application.

(iii) opportunity aspect of a dependency

- $\dot{B}(x, \exists y(\overleftarrow{D}\ (y, \eta) \wedge \exists z(C(y, z, \eta) \supset W(z, \eta)))) \ \subset\ \dot{W}(x, \overrightarrow{D}_{opp}\ (x, \eta))$

If the workability of an open dependency is an issue for a depender x, then raise as an issue the believability of the existence of some dependee agent y who is an open dependee with the desired dependum η and whose commitment on η would lead to η being workable for the depender.

Similarly, for a committed dependency:

- $\dot{B}(x, \overleftarrow{D}\ (y, \eta) \wedge (C(y, x, \eta) \supset W(x, \eta))) \ \subset\ \dot{W}(x,\ C\overrightarrow{D}_{opp}\ (x, \eta))$

(iv) committed dependency

- $\dot{B}(x, C\overleftarrow{D}\ (y, x, \eta)) \subset \dot{W}(x, C\overrightarrow{D}\ (x, y, \eta))$

If the workability of a committed dependency is an issue for a depender, then raise as an issue the believability of a committed dependency from the dependee.

(v) workability commitment assumption

- $\dot{B}(A(y, \eta) \wedge (C(y, x, \eta) \supset W(y, \eta))) \subset \dot{W}(x, C\overrightarrow{D}\ (x, y, \eta))$

If the workability of a committed dependency is an issue for a depender, then raise as an issue the believability of the ability of the dependee to achieve the dependum, as well as its workability for the dependee, given the dependee's commitment to achieve it.

(vi) workability transfer assumption

- $\dot{B}(C(y, x, \eta) \wedge W(y, \eta) \supset W(x, \eta)) \subset \dot{W}(x, C\overrightarrow{D}\ (x, y, \eta))$

If the workability of a committed dependency is an issue for a depender, then raise as an issue the believability of the workability of the dependum for the depender, given that the dependee is committed to delivering the dependum, and that the dependum is workable for the dependee.

Finally, if the believability of b is raised as an issue, and the issue is believable, then suggest to accept the assumption (settle the issue).

- $?settled(\dot{B}(x, b)) \subset \dot{B}(x, b) \wedge B(x, b)$

3.7 Summary

In this chapter, we presented a model for describing the rationales behind processes (including existing processes and their proposed alternatives) through the use of task-decomposition and means-ends relationships among process elements. The model can be analyzed using the concepts of ability, workability, viability, and believability.

The model supports the design of processes through the raising the issues, the addressing of issues (with the help of means-ends rules where applicable), and the identification of cross-impacted issues. It also assists in the identification of assumptions and in their justification.

The model was embedded in the Telos conceptual modelling language. The modelling concepts were illustrated with examples from the health care domain. Axioms were used to provide a sharper characterization of the modelling concepts.

Chapter 4

Application I – Requirements Engineering[1]

4.1 Introduction

Requirements engineering (RE) is the branch of software engineering that deals with the early phase of software development, during which the wants and needs of customers for an intended software system are explored, understood, documented, and refined to the extent that a technical system can be developed. Recent work in this area have emphasized the need for an engineering approach, where models and languages, methods and tools are employed to assist in the requirements engineering effort.

The importance of the requirements phase has long been recognized in the software engineering literature. For example, it has been estimated that an error that is not identified and corrected in the requirements phase can cost a hundred times more to correct in subsequent phases [Boehm81]. Empirical studies of software development projects have also confirmed the crucial importance of domain knowledge and requirements analysis. In a study of seventeen large software projects, Curtis et al. [Curtis88] concluded that the three critical risk factors for project success were:

- the thin spread of domain knowledge
- changing requirements
- failures in coordination and communication

Despite this recognition, much research in software engineering have focused on later phases (design and implementation), and managers have been reluctant to allocate resources to the early phase. Recent trends, however, indicate a greater awareness about the need for requirements engineering, as evidenced by the establishment of requirements engineering conferences and research communities [ISRE93] [ICRE94].

Requirements Engineering Research. Much of requirements engineering research have taken as starting point the initial requirements statements, which express customer's wishes about what the system should do. Initial requirements are often ambiguous, incomplete, inconsistent, and usually expressed informally, such as in natural language text. The objective is to produce a requirements document that is suitable for developers to start developing a

[1] An application of an early version of the $i*$ framework to requirements engineering appeared in [Yu93a].

technical system. Many requirements languages and frameworks have been proposed for helping to make requirements precise, complete, and consistent. Most impose some degree of structure and formality (from box-and-arrow diagrams to logical formalisms).

Considerably less attention has been given to the activities that *precede* the formulation of the initial, prescriptive requirements. Earlier activities include those that consider how the intended system would meet the organizational goals of the system as embedded in the larger organizational environment. This would enable an understanding of how and why the requirements came about.

"Early Requirements". This earlier phase of the requirements process is just as important, if not more important, than that of refining initial requirements to a requirements specification, at least for the following reasons:

- System development involves a great many assumptions about the embedding environment and task domain. As discovered in empirical studies, poor understanding of the domain is a primary cause of project failure. To have a deep understanding about a domain, one needs to understand the interests and priorities and abilities of various actors and players, beyond a grasp of the domain concepts and facts.

- Users need help in coming up with initial requirements in the first place – even the informal ones. As technical systems increase in diversity and complexity, the number of technical alternatives and organizational configurations made possible by them constitute a vast space of options. A systematic framework is needed to help developers understand what users want and to help users understand what technical systems can do. Many systems that are technically sound have failed to address real needs.

- It is well known that changes to requirements is a major source of problem. Traceability is an important need in software engineering. Having an understanding of organizational issues (in some model) would allow software changes to be traced all the way to the originating source – the organizational changes that leads to requirements changes.

- Having well-organized bodies of organizational and strategic knowledge would allow such knowledge to be shared across domains at this high level, deepening understanding about relationships among domains. This would also facilitate the sharing and reuse of software (and other types of knowledge) across these domains.

What *i offers.** ***i**** offers a requirements modelling framework to support the "early phase" of requirements engineering. It is intended to assist in the understanding of the organizational environment (of some potential system), in the exploration of alternate system proposals and how they would fit into various work settings, and in the analysis of the impact of alternatives system arrangements on organizational participants.

- The modelling of the organizational environment in terms of intentional relationships provides a richer modelling of the environment, in comparison to most existing requirements frameworks, which offer modelling in terms of entities and activities. The SD model can express the types of freedom that actors have, as well as their strategic interests.

- The explicit modelling of the rationales that underlie process structures encourages a deeper understanding about *why* systems are (or planned to be) embedded in an organization in a certain way. This deeper understanding is also likely to lead to a more accurate description of the process structure (beyond the official descriptions of work procedures).

- The framework supports the analysis of the proposed systems and organizational configurations in relation to the strategic concerns of actors. Interdependencies among actors (including systems) are analyzed in terms of opportunity and vulnerability. Actors' routines are analyzed for workability, viability, and believability.

- The framework supports the systematic exploration of new system-and- environment alternatives, using a combination of functional and non-functional means-ends reasoning. Issues and stakeholders cross-impacted by proposed system alternatives are identified and addressed during this exploration, and need not be known beforehand, consistent with an "open world" perspective.

- The use of a knowledge-based approach facilitates the collection, organization, use and reuse of domain knowledge across cases and across domains. This knowledge might include strategic interests of actors in various domains, and how well different types of technical systems and features have been found to address these interests.

- The knowledge-based requirements model would map smoothly into knowledge-based frameworks for software design and implementations, speeding up software development and supporting software evolution to meet evolving requirements, and provide traceability of rationales and assumptions all the way from technical implementation decisions to strategic business policy decisions.

In the following two sections, we illustrate how the ***i****** framework can be used as a framework for the early phase in requirements engineering. Section 5.2 explains how the SD model is used to help deepen understanding during requirements modelling. Section 5.3 shows how the SR model is used to capture stakeholder issues and concerns, and how it assists in addressing them. Section 5.4 compares ***i****** with existing frameworks for requirements engineering.

A meeting scheduler example is used as the domain setting throughout this chapter. The domain of meeting scheduling has been adopted by a number of researchers in the Requirements Engineering community as a common ("benchmark") example for comparing RE techniques. The example has been adopted because it is small enough, but contains considerable range of issues for comparing how various frameworks would deal with them. The example used in this chapter is a simplified version of [VanLamsweerde93].

4.2 Modelling Strategic Dependencies in Requirements Engineering

In conventional approaches to information systems development, the requirements phase typically begins with an informal description of what the system is expected to do. The environment for which the system is targeted is usually described in terms of the activities that are expected to be performed (perhaps associated with certain agents), the entities that are produced and consumed (as inputs and outputs of activities), for example, as in an SADT model or DFD context diagram.

Consider a computer-based meeting scheduler for supporting the organization of meetings. A requirements statement might say that, for each meeting request, the meeting scheduler should try to determine a meeting date and location so that most of the intended participants will participate effectively. The system would find dates and locations that are as convenient as possible. The meeting initiator would ask all potential participants for information about their availability to meet during a date range, based on their personal agendas. This includes an exclusion set – dates on which a participant cannot attend the meeting, and a preference set – dates preferred by the participant for the meeting. The meeting scheduler comes up with a proposed date. The date must not be one of the exclusion dates, and should ideally belong to as many preference sets as possible. Participants would agree to a meeting date once an acceptable date has been found.

A number of schemes have been proposed to express this type of knowledge about system environment to various degrees of formality. The models are interpreted prescriptively, i.e., the system (here the meeting scheduler) and other agents (meeting initiator and participants) are supposed to conform to the model.

In contrast, the ***i****** framework supports a *descriptive* view for developing a deeper understanding about the organizational environment in which the proposed system is to be embedded, and is aimed at a phase in RE before the prescriptive requirements are arrived at. One aims to identify who will be affected (stakeholders), and how their strategic interests would be affected by changes in the work processes associated with the proposed system.

Using the Strategic Dependency model, one can first describe and analyze the relationships among actors in the organizational environment as it exists, before the proposed system is introduced. This can then be compared to the new configuration which includes the proposed system. The relationships can be analyzed in terms of opportunities and vulnerabilities.

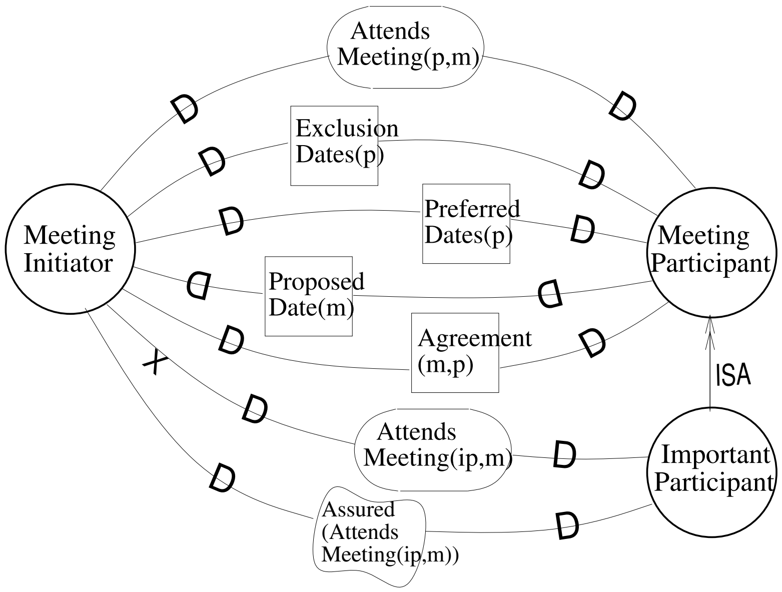

Figure 4.1: Strategic Dependency model for meeting scheduling, without computer-based scheduler

Figure 4.1 shows an SD model of meeting scheduling without computer-based scheduler support. The meeting initiator depends on participants to attend the meeting. To schedule meetings, the initiator depends on participants to provide information about their availability – in terms of a set of exclusion dates and preferred dates. (For simplicity, we do not separately consider time of day or location.) To arrive at an agreeable date, participants depend on the initiator for date proposals. Once proposed, the initiator depends on participants to indicate whether they agree with the date. For important participants, the meeting initiator depends critically on their attendance, and thus also on their assurance that they will attend.

Dependency types are used to differentiate among the kinds of relationships between depender and dependee, involving different types of freedom and constraint. The meeting initiator's dependency on participant's attendance at meeting (`AttendsMeeting(p,m)`) is a *goal dependency*. It is up to the participant how to attain that goal. An agreement on a proposed date `Agreement(m,p)` is modelled as a *resource dependency*. This means that the participant is expected only to give an agreement. If there is no agreement, it is the initiator who has to find other dates (do problem solving). For an important participant, the initiator critically depends on that participant's presence. The initiator wants the latter's attendance to be assured (`Assured[AttendsMeeting(p,m)]`). This is modelled as a *softgoal dependency*. It is up to the depender to decide what measures are enough for him to be assured, e.g., a telephone confirmation. These types of relationships cannot be expressed or distinguished in non-intentional models that are used in most existing requirements modelling frameworks.

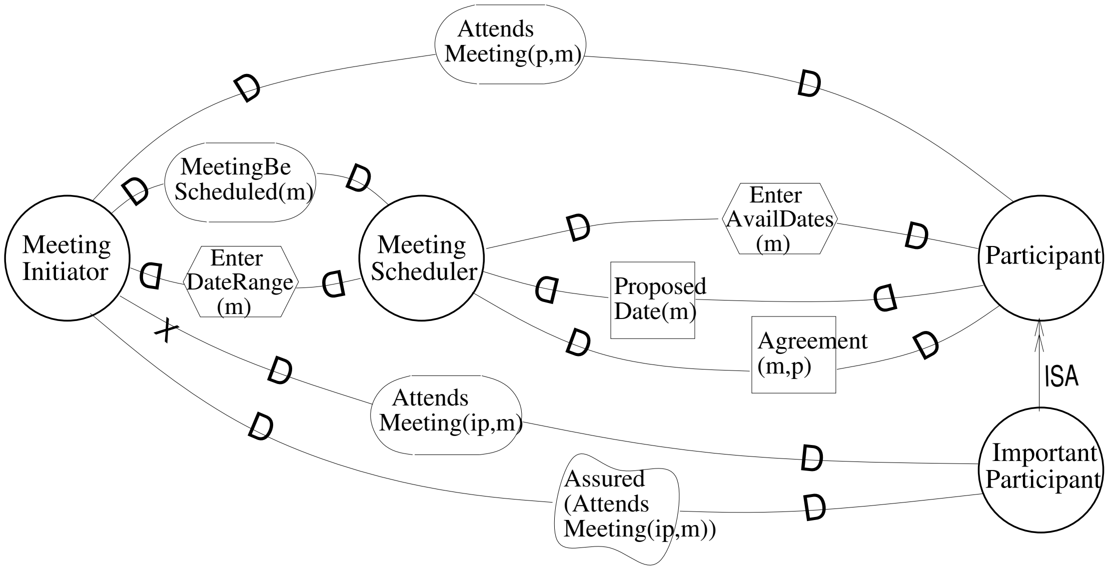

Figure 4.2: Strategic Dependency model for meeting scheduling with computer-based scheduler

Figure 4.2 shows an SD model of the meeting scheduling setting with a computer-based meeting scheduler. The meeting initiator delegates much of the work of meeting scheduling to the meeting scheduler. The initiator no longer needs to be bothered with collecting availability information from participants, or to obtain agreements about proposed dates from them. The meeting scheduler also determines what are the acceptable dates, given the availability information. The meeting initiator does not care how the scheduler does this, as longer as the acceptable dates are found. This is reflected in the *goal dependency* of `MeetingBeScheduled` from the initiator to the scheduler.

Note that it is still the meeting initiator who depends on participants to attend the meeting. Assurance from important participants that they will attend the meeting is not delegated to the scheduler, but retained as a dependency from meeting initiator to important participant.

The SD model models the meeting scheduling process in terms of intentional relationships among agents, instead of the flow of entities among activities. This allows analysis of opportunity and vulnerability. For example, the ability of a computer-based meeting scheduler to achieve the goal of `MeetingBeScheduled` represents an opportunity for the meeting initiator not to have to achieve this goal himself. On the other hand, the meeting initiator would become vulnerable to the failure of the meeting scheduler in achieving this goal.

4.3 Using Strategic Rationales in Requirements Engineering

The Strategic Rationale model assists in requirements engineering by allowing process elements and the rationales behind them to be expressed. During early requirements engineering, the SR model can be used to understand how systems are embedded in organizational actors' routines, to generate alternatives, and to model and support actors' reasoning about the alternatives.

Process Modelling. While the Strategic Dependency model provides an initial understanding of the intentional structure of an organizational environment in terms of external relationships between actors, the Strategic Rationale model provides a more detailed understanding by looking "inside" actors to see how processes are comprised of intentional elements, and how these elements contribute to the overall purposes of the routines.

For example, when a computer-based meeting scheduler is proposed, an initial understanding would include what a meeting initiator (a role, usually played by a human agent) has to deal with in order to organize a meeting. This understanding is important before deciding what capabilities and features a meeting scheduler should offer. This might include what aspects of meeting scheduling should be left to a human), or the even larger context within which meeting scheduling occurs – what types of meetings, why meetings are held, for example, meetings for reviewing code in a software development project.

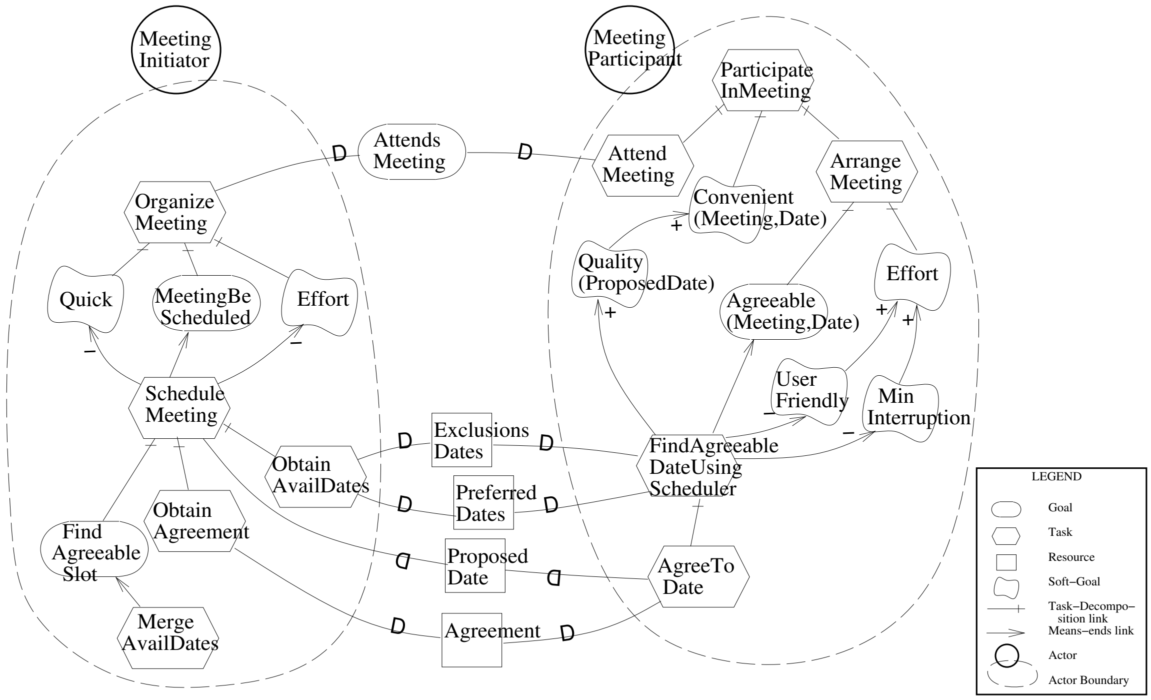

Figure 4.3: A Strategic Rationale model for meeting scheduling, before considering computer-based meeting scheduler

Consider the organizational environment in the meeting scheduling example, before computer-based systems are introduced. (Figure 4.3). To have a meeting, the meeting initiator depends on participants to attend (be present at) the meeting. That the meeting be scheduled is a *subgoal* of `OrganizeMeeting`. Other subgoals might include equipment be ordered, or that reminders be sent (not shown). The setting up of meetings should be done quickly and not involve inordinate amounts of effort. These are modelled as *softgoals*.

To schedule a meeting (without computer assistance), the meeting initiator obtains availability information from participants (exclusions dates and preferred dates). The initiator then finds a date (and time) slot that satisfies the availability information, and obtains agreements from participants to confirm their attendance.

Participants who are expected to participate in the meeting have to do their part in arranging for the meeting, and then to attend the meeting. For the participant, arranging the meeting consists primarily of arriving at an agreeable date. This requires them to supply availability information to the meeting initiator, and then to try to agree to the proposed dates. Participants want selected meeting times to be convenient, and want meeting arranging activities not to present too many interruptions.

In the SR model, *task-decomposition links* provide a hierarchical description of intentional elements that make up a routine. Each element in a task is needed for the success of the task. The *means-ends links* in the SR provides understanding about *why* an actor would engage in some tasks, pursue a goal, need a resource, or want a softgoal. From the softgoals, one can tell *why* one alternative may be chosen over others. (For simplicity, multiple alternatives are not shown in Figure 4.3.) For example, availability information in the form of exclusion sets and preferred sets are collected so as to minimize the number of rounds and thus to minimize interruption to participants.

Knowledge structuring can be used to collect knowledge about different types of meetings. The rationale patterns for different types of meetings may be variations of a generic meeting pattern. For example, to organize some meetings, there may be no need to inquire about the availability of participants. In others, availability information may be needed, by there is no need to confirm with participants.

Process Analysis. The SR model can be analyzed to determine whether an actor has the ability to accomplish something, whether those routines are workable and viable, and whether the actors beliefs about the above are well substantiated.

When a meeting initiator has a routine to organize a meeting, we say that he is *able* to organize a meeting. An actor who is able to organize one type of meeting (say, a project group meeting) is not necessarily able to organize another type of meeting (e.g., the annual general meeting for the corporation). One needs to know what subtask, subgoals, resources are required, and what softgoals are pertinent.

Given a routine, one can analyze it for *workability* and *viability*. Organizing meeting is workable if there is a workable routine for doing so. To determine workability, one needs to look at the workability of each element – for example, that the meeting initiator can obtaining availability information from participants, can find agreeable dates, and can obtain agreements from participants. If the workability of an element cannot be judged primitively by the actor, then it needs to be further reduced. If the subgoal `FindAgreeableSlot` is not primitively workable, it will need to be elaborated in terms of a particular way for achieving it. For example, one possible means for achieving it is to do an intersection of the availability information from all participants. If this task is judged to be workable, then the `FindAgreeableSlot` goal node would be workable. Tasks can be workable by way of external dependencies. The workability of `ObtainAvailDates` and `ObtainAgreement` are evaluated in terms of the workability of the commitment of meeting participant to provides availability information and agreement.

A routine that is workable is not necessarily viable. Although computing intersection of time slots by hand is possible, it is slow and error-prone. Potentially good slots may be missed.

When softgoals are not satisficed, the routine is not viable.

Note, however, that a routine which is not viable from one actor's perspective may be viable from another actor's perspective. For example, the existing way of arranging for meetings may be viable for participants, if the resulting meeting dates are convenient, and the meeting arrangement efforts do not involve too much interruption of work.

The assessment of workability and viability is based on many beliefs and assumptions. These can be provided as justifications for the assessment. The *believability* of the rationale network can be analyzed by checking the network of justifications for the beliefs. For example, the argument that "finding agreeable dates by merging available dates" is workable may be justified with the assertion that the meeting initiator has been doing it this way for years, and it works. The belief that meeting participants will supply availability information and agree to meeting dates may be justified by the belief that it is in their own interests to do so (e.g., programmers who want their code to pass a review).

Process Design (or Redesign). The SR model allows us to *raise* ability, workability, and viability as *issues* that need to be addressed. Using means-ends reasoning, these issues can be *addressed* systematically, resulting in new process configurations that are then to be evaluated and compared. Means-ends rules that encode knowhow in the domain can be used to assist in generating alternatives. Issues and stakeholders that are *cross-impacted* may be discovered during this process, and can be raised so that trade-offs can be made. Issues are *settled* when they are deemed to adequately addressed by stakeholders. Once settled, one can then proceed from the descriptive model of the $\boldsymbol{i}^*$ framework to a prescriptive model that would serve as the requirements specification for systems development.[2] Believability can also be raised as issues, so that assumptions would be justified.

In analyzing the SR model of Figure 4.3, it is found that the meeting initiator is dissatisfied with the amount of effort needed to schedule a meeting, and how quickly a meeting can be scheduled. These are raised as the issues `Quick[MeetingScheduling]` and `LowEffort [MeetingScheduling]`.

Since the meeting initiator's existing routine for scheduling meetings is deemed unviable, one would need to look for new routines. This is done by raising the meeting initiator's ability to schedule meetings as an issue (because `Issue(Viable(OrganizeMeeting))` being raised suggests the raising of `Issue(Able(OrganizeMeeting))`).

To address this issue, one could try to come up with solutions without special assistance, or one could look up *rules* that may be applicable. Suppose a rule is found whose *purpose* is `MeetingBeScheduled` and whose *how* attribute is `LetSchedulerScheduleMeeting`.

```
Class   CanLetSchedulerScheduleMeeting  IN Rule WITH
 purpose
     ms:   MeetingBeScheduled
 how
     lssm: LetSchedulerScheduleMeeting
 applicabilityCond:
     platform: $HasAppropriatePlatform(team,platform,scheduler)$
 END
```

This represents knowledge that the initiator has about software scheduler systems, their abilities, and their platform requirements. The rule helps discover that the meeting initiator can delegate the subgoal of meeting scheduling to the (computer-based) meeting scheduler. This constitutes a routine for the meeting initiator.

[2] This step is future work.

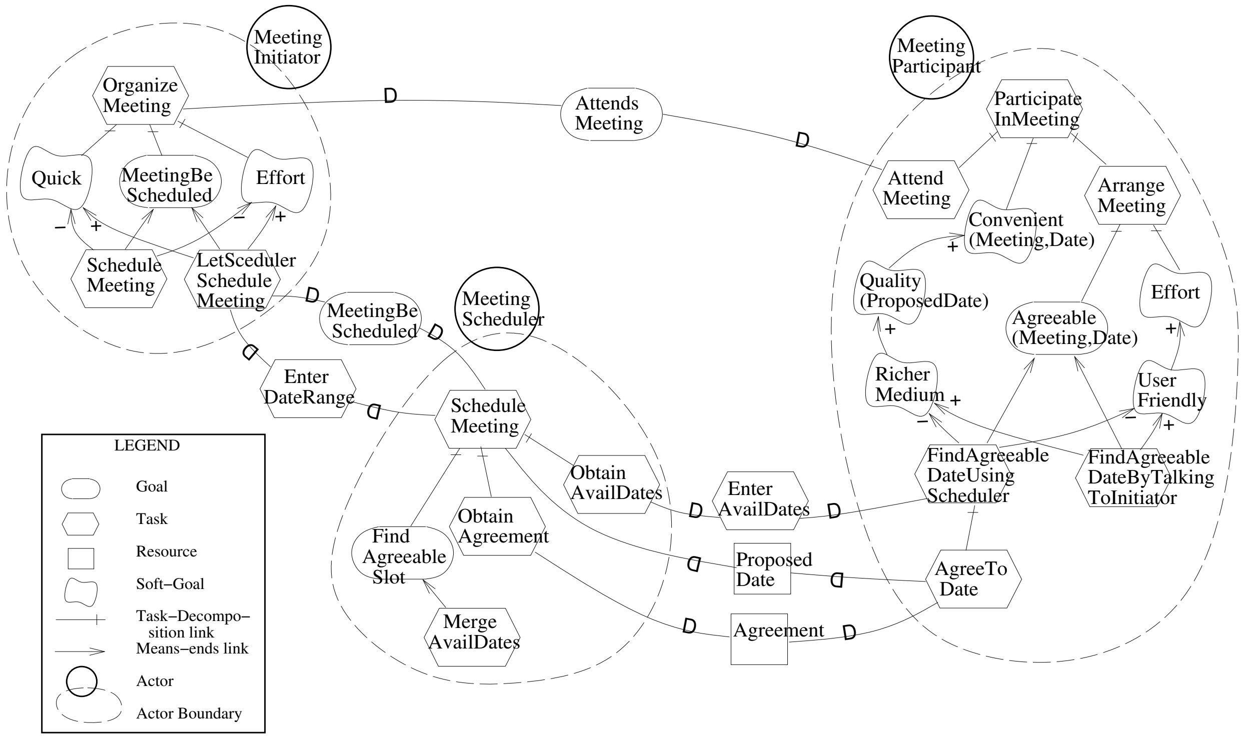

Figure 4.4: Strategic Rationale model for a computer-supported meeting scheduling configuration

Using a meeting scheduler, however, requires participants to enter availability information in a particular format. This is modelled as a *task dependency* on participants (an SD link). A routine that provides for this is sought in the participant. Again, rules may be used to assist in this search.

When new configurations are proposed, they may bring in additional issues. The new alternatives may have associated softgoals. The discovery of these softgoals can also be assisted with means-ends rules. For example, using computer-based meeting scheduling may be discovered to be negative in terms of medium richness and user-friendliness. These in turn have implications for the effort involved for the participant, and the quality of the proposed dates. These newly raised issues also need to be addressed.

Once new routines have been identified, they are analyzed for workability and viability. Further routines are searched for until workable and viable ones are found.

The unguided search for a match to "functional goals" do not necessarily end up with solutions that would meet the non-functional softgoals. A more controlled search is to make use of means-ends rules for the desired *softgoals*. For example, a softgoal rule might state, "To achieve `Quick[p]` (where `p` is some process), consider using computer-based systems", or "For `LowEffort[p]`, try using computer-based systems, but note the cross-impacts on user-friendliness and requisite training". These are called *methods* in [Chung93]. However, when such softgoal rules (for addressing non-functional goals) do not exist, one can still try goal reduction using the functional goals.

This process will lead to iterative exploration and discovery of alternatives. For meeting scheduling, one may come to explore using e-mail as the interface, to minimize effort for the participant. To minimize interruption, one way is to have a representative agent (e.g., a secretary) acting on behalf of a principal.

4.4 Discussion

In this section, we discuss how the $\boldsymbol{i}^*$ framework builds on and advances beyond existing Requirements Engineering (RE) frameworks. The $\boldsymbol{i}^*$ framework proposes a set of modelling, analysis and design concepts and techniques aimed at the "early" stage of requirements engineering, preceding but leading up to the *requirements document.* Much of RE research have focused on informal requirements statements as the starting point. There is little assistance, either conceptually or in the form of tools, to help users come up with initial requirements in the first place. In particular, formal modelling technqiues such as those developed in knowledge representation have been applied to later stages of requirements (e.g., [Greenspan84][Dubois86]) and to the rest of software development [Jarke92a]), but not to early requirements.

The $\boldsymbol{i}^*$ framework offers the perspective that the modelling concepts and techniques used for later phase requirements engineering are not necessarily appropriate for the early phase. A different ontology for describing and reasoning about a more open conception of the world (systems and their environments) is needed for early requirements. Despite the openness and "softer" nature of the knowledge involved, early requirements can still benefit from formal, knowledge-based concepts and techniques. The framework thus opens the way for a systematic approach to early phase RE, with tool support, compatible with and thus extending the scope of the knowledge-based approach to software engineering, to cover this previously *ad hoc*, unsupported phase of system development.

We discuss the differences in perspective between $\boldsymbol{i}^*$ and existing RE frameworks in terms of five areas.

(i) "Scriptiveness" – prescriptive versus descriptive models. Requirements specifications on a system are usually interpreted prescriptively. They state what a system is supposed to do. Requirements documents are often used in contractual settings – developers are obliged to design the systems in order to meet the specifications. To formalize requirements from this perspective, a number of requirements modelling languages have been developed to render the obligational aspects of requirements explicit, using, for example, deontic logic (e.g., [Dubois89] [Finkelstein87])

In early requirements, however, users and developers explore alternatives, before settling on any particular set of requirements. One would therefore like to reason about the implications of various alternatives, including the consequences of obligations *not* being met. This would then allow for considerations of human procedures or additional hardware/software systems/features to deal with them. This is hard to do in a prescriptive model which assumes that agents would behave according to the prescription. When models that do not have explicit representation of obligations are interpreted prescriptively, there is an implicit, blanket, unilateral obligation on all agents in the model (human or machine) to some unspecified agent outside the model (e.g., someone who might have the power to enforce the obligations.)

The $\boldsymbol{i}^*$ framework adopts a descriptive view. Agents make commitments to other agents, so there are multi-lateral relationships among agents (intentional dependencies) described explicitly in the model. One can reason about opportunity and vulnerability (implications of commitments not being met), and use analysis and design techniques to explore alternative configurations of relationships systematically in order to make use of opportunity and to mitigate vulnerability.

(ii) Completeness and consistency. A great deal of effort in RE is aimed at making requirements documents complete and consistent. These are clearly important in a requirements document because one does not want to underspecify – leaving out items that one wants to specify, or have requirements statements that contradict each other.

In early phase requirements, however, the information that one collects in order to eventually lead to a requirements document will most likely be very incomplete, and will contain conflicting statements, especially conflicting wants about proposed systems. This can be expected since many parties and stakeholders would have different perceptions and visions about the proposed systems. Inconsistencies provide useful clues during the modelling (e.g., to identify conflicting statements as articulating competing alternatives, or as a signal for the need to separate two roles).

Incompleteness can also be used to advantage. One does not want to be burdened with unnecessary details that are not important for this stage. Information that have no bearing on the choice of process alternatives (system or human) can be left out. The intentional, strategic concepts of $\boldsymbol{i}^*$ allow and support this.

Analysis based on the concept of workability allows details that are judged immaterial or inconsequential to be omitted. The concept of viability draws on the concept of satisficing as developed in [Mylopoulos92] and [Chung93]. Applying the stricter discipline of requirements modelling schemes that have been designed for late stage requirements to the early phase would be too restrictive and thus inappropriate.

(iii) The notion of process. Most requirements models treat a process as a series of activity steps. The primary abstraction mechanism is decomposition (e.g., as in structured analysis, e.g., SADT [Ross77] or DFDs [DeMarco79]).

In early requirements, it is important to acknowledge that, for many processes, the activity steps are not tightly constrained or known beforehand. It is up to the agents carrying out the process to elaborate at "run-time". On the other hand, although it is sometimes adequate to describe processes at a high level (coarse-grained), agents often do constrain each others behaviour to varying extents.

In the SD model, a process is described in terms of a network of intentional relationships between agents. The non-essential aspects of the process are hidden by using the agent abstraction – as "internal" actions and decisions that are only of concern to the agent, but not to outsiders. The essential aspects of the process are modelled as dependency relationships between particular agents, so that the impact of process step failure can be analyzed (in contrast to the blanket obligation implicit in most prescriptive process models). Different types of dependencies are used to express how agents constrain the behaviour of other agents. These features cannot be expressed using only the decomposition abstraction.

The SR model accommodates a more detailed conception of process, by allowing process elements and their rationales to be described, so that agents' reasoning about process alternatives can be explicitly represented. Even here, a high degree of incompleteness in describing the process can be tolerated. The means-ends hierarchy is used as an abstraction mechanism: means (intentional process elements) that do not contribute significantly to ends (other intentional process elements) can be omitted from the model, because of the use of the concepts of workability and satisficing.

(iv) The notion of agent. In most RE frameworks, the notion of agent is only weakly linked to that of process. Most typically, process steps (activities or actions) are mapped to agents (e.g., [Mylopoulos90] [Dardenne93]). Agents are treated as just another class of object, not a separate ontological category with a distinct semantics.

In ***i****, because an intentional view of process is adopted, the notion of process is closely tied to the notion of agent. The agent is the source of intentionality. Agents have goals, beliefs, abilities, and take actions in pursuit of their interests. A process, as described in an SD model, is a set of constraints on agents' otherwise autonomous and independent behaviour.

Tying processes to agents would seem to compromise the ability to describe processes independently of "who" is involved in performing them, in violation of the systems design principle of abstraction. However, in a descriptive model, we try to allow the model to reflect whatever degree of dependence or independence exists in the world being modelled, and not to impose normative principles as one would in prescriptive models. More abstract notions of agents such as *roles* have been explored at length in the social studies literature. The ***i**** framework thus uses the term *actor* to refer to the generic entity that have intentionality. The concepts of roles, positions, and agents are then defined as specializations of actor, reflecting different degrees of concreteness of agency.

The agent/position/role mappings are useful for modelling the grouping of the large number of dependencies (incoming and outgoing) that agents in the real world typically have. Agents play many roles and participate in many processes. ***i**** draws attention to the possible impact that agents' involvement in *other* processes might have on the process being studied. For example, during (early) requirements engineering for a meeting scheduler, a study of the meeting scheduling process only would not provide a very good basis for deciding on requirements, if the richer surrounding social context is not taken into account (e.g., meeting scheduling in an industrial project environment, as opposed to in a university department). ***i**** provides a framework for analyzing the influence of these other processes (the social context) in terms of dependencies from (or on) other roles and positions that impinge on the relevant agents.

(v) Goal-oriented requirements engineering. In recent RE research, a number of schemes have been proposed using a goal-oriented approach to arrive at requirements [Feather87] [Dubois89] [Fickas92] [Dardenne93] [Chung93] [Feather94]. The main idea is that systems should be viewed as fulfilling some higher goal in the larger environment. By starting with these higher goals, one can arrive at requirements by a (mostly) top-down goal reduction process.

The ***i**** framework is also predicated on a goal-oriented process, but differs from existing frameworks in this area in several important respects:

1. *Modelling is emphasized.* The ***i**** framework emphasizes modelling, i.e., its focus is on providing modelling features that allow phenomena in the world to be expressed in a model. It is assumed that the use of the model(s) would involve a great deal of user input and judgement throughout. The framework also provides *support* for analysis and design, but does not necessarily aim to provide the highest degree of automation. For example, the framework is intended to support and model *the agents' reasoning* about alternatives, not to automatically generate solutions and give a final recommendation to the agents.

2. *Process design goals need to be discovered.* Higher goals in the embedding environment of the intended system are not necessarily known beforehand. The early requirements process is more likely to require an in-depth understanding of *existing* processes and their rationales (hence "*re*-engineering"). Process redesign goals are often discovered (or clarified) during the modelling process (e.g., as a result of asking "why" questions, and doing analysis on workability and viability). Those goals that are selected for addressing are raised

as *issues*. Instead of being predominantly top-down, the RE process is likely to be up-and-down (elaborating the model along the means-ends hierarchy), and middle-out (from agents closely associated with the process to stakeholders who are indirectly impacted). As goals are being reduced, issues that are cross-impacted by proposed solutions may be raised, and further stakeholders may be identified. So the process is likely to be highly iterative and open-ended.

3. *Process design goals are only partially reduced.* Because processes are described intentionally, and agents are assumed to have problem solving ability (in the general case), the goal reduction process does not aim to reduce goals fully (unlike classical planning in AI, for example). Goals only need to be solved to the extent that agents can be found who can *further* reduce or solve the goals during process *execution*.

4. *Social actors are viewed as being strategic.* Agents in early requirements model are viewed as being strategic – their behaviour is affected by a confluence of many external relationships. During this part of RE, we support a richer analysis of the surrounding social environment, such as the mitigation of vulnerability through enforcement, assurance, and/or insurance. We defer concepts such as ensuring [Dardenne93] (imposition of tighter constraints on the systems being prescribed so as to be able to prove properties about the process) to a later stage in RE.

5. *Intentionality is distributed, both before and after process redesign.* Since actors are strategic, they typically would each have their *own* process design goals. We therefore do not assume that the multiplicity of stakeholder interests can be or need to be condensed into a set of global goals in order to drive the requirements process. Each actor pursues its own strategic interests by putting forth new process configurations that address its "issues", and evaluate alternatives with respect to those interests. We also do not need to assume that the redesigned process is optimal in some objective or global sense, or that it is arrived at rationally. The new process is usually a negotiated settlement reflecting the complex social realities in which the actors are embedded. It is possible that some actors succeed in advancing their strategic interests at the expense of others. What ***i****** offers is a framework for strategic actors to achieve a deeper understanding about the strategic relationships and issues facing them and hopefully enable them to pursue their interests more productively.

6. *The focus is on early requirements.* Much of the above differences stem from ***i******'s focus on early requirements, leading to a descriptive stance in modelling. Most of the goal-oriented RE frameworks have prescriptive models as the end-product of the goal-driven process. The ***i****** approach presupposes that an additional step would need to be taken in order to arrive at prescriptive requirements specifications. The benefit of separately considering early requirements is that concepts and techniques that are more suited to this stage (as outlined in this section) can be employed.

Chapter 5

Application II – Business Process Reengineering[1]

5.1 Introduction

The concept of business process reengineering (BPR) has been advocated as a way to achieve dramatic improvements in organizational performance by fundamentally re-designing the work organization as new information technology is introduced [Davenport90][Hammer90][Stewart92]. Although computer-based information systems are used extensively in business environments today, it has been argued that they often do not deliver significant benefits because they are simply used to automate existing (and often outdated) business practices. Business process reengineering proposes that one should question the appropriateness of the work process itself, with respect to today's business environment. To do this, one needs to keep asking Why? and What-if? questions about existing work practices. As businesses respond to competitive pressures, customer demands, and changing regulatory conditions, they are increasingly fundamentally rethinking the way they do business, and expecting computers to play key roles in innovative solutions [Venkatraman91][Keen91].

An example. An example that is often used to illustrate business process reengineering is that of goods acquisition. In most organizations, the person or department (the client) in need of an item would fill out a purchase requisition form, forward it to the purchasing department. A purchasing specialist would obtain price and delivery quotes from vendors, and place the order by mailing a purchase order to the selected vendor. Copies of the PO are sent to receiving and accounts payable. When the item arrives at receiving, it is checked against the copy of the PO at receiving. If the item was indeed what was ordered, it is sent on to the client. A receiving notice is sent to accounts payable. When accounts payable gets an invoice from the vendor, invoiced items are checked against purchase orders and receiving notices. If they match, payment is issued to the vendor.

This process involves a great deal of paperwork, and is slow and error-prone. For example, reconciliation of accounts may require sorting, filing, and searching for missing information, thus is not straight-forward, but involves problem solving.

As reported in [Hammer90], one company reengineered its goods acquisition process by following this line of reasoning. The traditional purchasing process was full of paperwork, errors, and delays. For small purchases, it was not uncommon for the purchasing process to

[1]This chapter is based in part on [Yu93c], [Yu94a], and [Yu94d].

cost more than the item. The company recognized that expert systems technology could be used to provide the knowhow and resource support for simple purchases. Now, except for large or strategic orders, company employees would order most items directly from pre-approved vendors through the system without the help of human purchasing agents.

The Ford Motor Company took a different approach. It reengineered its goods acquisition process by eliminating invoices. Instead of paying when an invoice is received, Ford now pays when it gets the goods. A large part of accounts payable work consists in reconciling disagreements among purchase orders, receiving documents, and invoices. Information technology could have been deployed to help investigate invoicing errors, and to automate document flow, but that would not have provided the radical improvement Ford had aimed for. By re-designing the business process, invoices, and hence invoicing errors, were eliminated altogether. The remaining reconciliation (between purchase orders and receiving reports), which is much simpler, could now be handled mostly by computer, which also generates the payment cheque.

The typical reengineering effort. The major activities in a typical reengineering effort would include (see, e.g., [Hammer93])

- identifying, delineating, and modelling the existing process
- analyzing it for deficiencies
- proposing new solutions (process design)
- implementing the new design, in terms of new technical systems and also new organizational (people) structures (roles and responsibilities).

A reengineering undertaking is fraught with risks and difficulties. Even though there has been many success stories with dramatic payoffs, BPR consultants frequently admit that only a small fraction of such efforts succeed [James93] [Karlgaard94]. The practice of reengineering is acknowledged to be more art than science, and results are often unpredictable. The area is thus rich with issues and problems for research.

Problems and issues. There are important research problems both on the technical side as well as on the people side. On the technical side, there are issues concerning process modelling, analysis and design.

- The ability to describe a process in a clear and sufficiently rich way is acknowledged as crucial to a reengineering effort. Currently, informal graphical flow charts (workflow models) are often used. These offer limited analytical capability and are not well related to technical system development methods. IDEF models (essentially SADT and ER models) are also used. These models are closer to technical system development methods, but do not capture organizational aspects well.

- The search for new process solutions is also informal and *ad hoc*. Guidelines, benchmarking, best practices are used in looking for opportunities for improvement. Examples of rules-of-thumb guidelines are "organize around outcomes" and "put decision where work is performed" [Hammer90] [Stewart92]. Benchmarks and guidelines help to some extent. However, since each organization is different, a systematic framework or methodology needs to be developed to help find solutions to address the particular circumstances [Keen91] [Davenport93].

- Since existing representational or modelling schemes are mostly informal (itemized text, box-and-arrow diagrams without clear semantics), the level of tool support is low. They are typically diagramming tools with little reasoning support. The link to information system development tools is also weak.

- Furthermore, the knowledge collected during reengineering cannot be reused easily, either for supporting future changes, or to benefit other reengineering efforts.

On the people side, the problems are not as clear-cut, but perhaps more serious and deep-seated (e.g., [Davenport94]). The opinion has often been voiced that organizational implementation ("change management") is the real difficulty in BPR. In comparison, coming up with new process designs is the easy part. However, this could well be an indication of a fundamental deficiency in the kinds of considerations that have been taken into account at the process analysis and design stage. It is conceivable that richer model capable of providing a deep enough understanding about the interests and concerns of the various stakeholders, to produce a design that addresses them more fully, can lead to a better chance of success in the implementation.

What *i offers.** ***i**** offers a framework for modelling, analyzing, and designing business process in terms of intentional, strategic actor relationships. The Strategic Dependency (SD) model describes a business organization in terms of the dependencies that actors have on each other in accomplishing their work. The Strategic Rationales (SR) model describes the reasoning that actors have about the different possible ways of organizing work, i.e., different configurations of Strategic Dependency networks. The framework can be used to assist participants and stakeholders in a business process to develop a deeper understanding about the existing process, and to systematically generate alternatives in order to arrive at new process designs that can better address business objectives and individual concerns.

i* offers a systematic approach to BPR based on a richer model of process structure, and of the rationales behind it, as follows:

- The modelling of a business process as a network of intentional relationships allows for the acknowledgement of the need for decision making (discretion, freedom, open-endedness) at particular points in the process, and who depends on whom for addressing those problems;

- The network of dependency relationships provides a systematic way of identifying stakeholders and their concerns;

- Explicit means-ends reasoning encourages and supports systematic search for process design alternatives;

- The representation of softgoals (with a qualitative reasoning framework) supports the identification of cross-impacted issues (and stakeholders), thus improving coverage of pertinent issues at the process design phase;

- The knowledge-based approach of ***i**** facilitates the collection, organization, use and reuse of knowledge about business processes, which can be drawn from case experiences and generic principles;

- This approach to business process modelling, analysis, and design is compatible with the knowledge-based approach to software engineering and information systems development, enabling a fairly direct link to software development to support business processes (e.g., by using the mapping framework of DAIDA [Jarke92a]).

- The explicit modelling of business processes and their rationales facilitates on-going process evolution, and especially facilitates the software evolution of the systems aimed at supporting these processes;

- By encouraging an understanding of strategic issues at the process design stage, including the implications of various technical system alternatives to stakeholders, the framework could potentially lead to process designs that can achieve earlier buy-in by participants, and hopefully high rates of success in implementation.

5.2 Modelling Strategic Dependencies in Business Processes

A business process is most often modelled as a network of flows of work products from one work unit (e.g., a department or a person) to another. For example, a typical workflow for acquiring goods in a business organization might be as represented in Figure 5.1. A more detailed model would show activities performed within each unit, with intermediate products as inputs and outputs of activities. Workflow models show the entities and activities involved in a work process, but not the reasons for their existence or relatedness. They also do not convey the types of problem solving that may be involved in carrying out the work.

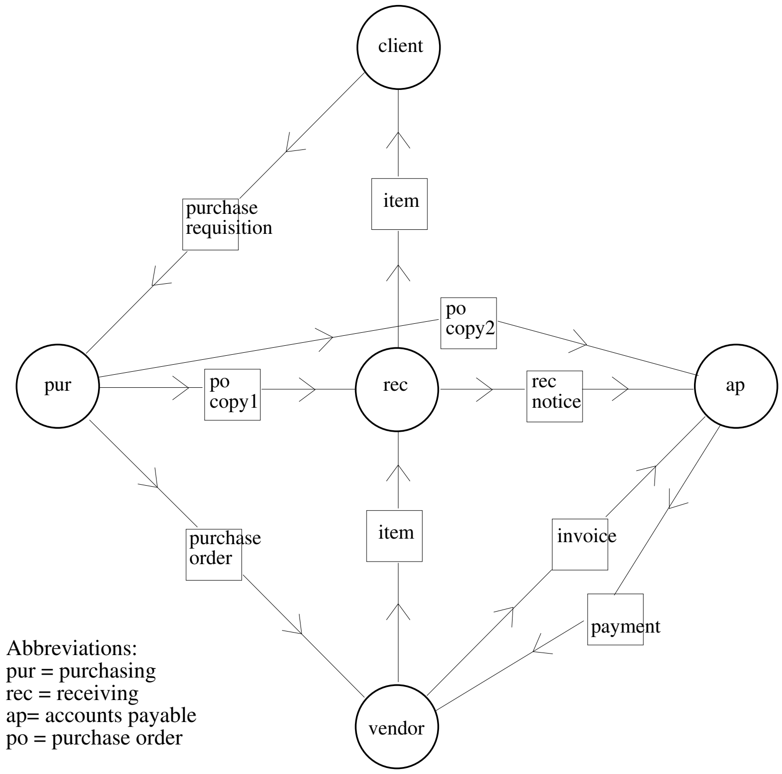

Figure 5.1: A "work-flow" model of a goods acquisition process

Figure 5.2 shows a Strategic Dependency model for a goods acquisition process. In the modelling of Strategic Dependencies, we assume that business processes, unlike processes that are executed by machines, exist in social organizational settings. Organizations are made up of social *actors* who have goals and interests, which they pursue through a network of relationships with other actors. A richer model of a business process should therefore include not only how work products (entities) progress from process step to process step (activities), but also how the actors performing these steps relate to each other *intentionally*, i.e., in terms of concepts such as goal, belief, ability, and commitment. When an organization seeks new ways for organizing work, actors who have goals and interests are likely to evaluate these proposals *strategically*, e.g., in terms of potential opportunities and threats. A model for supporting business process

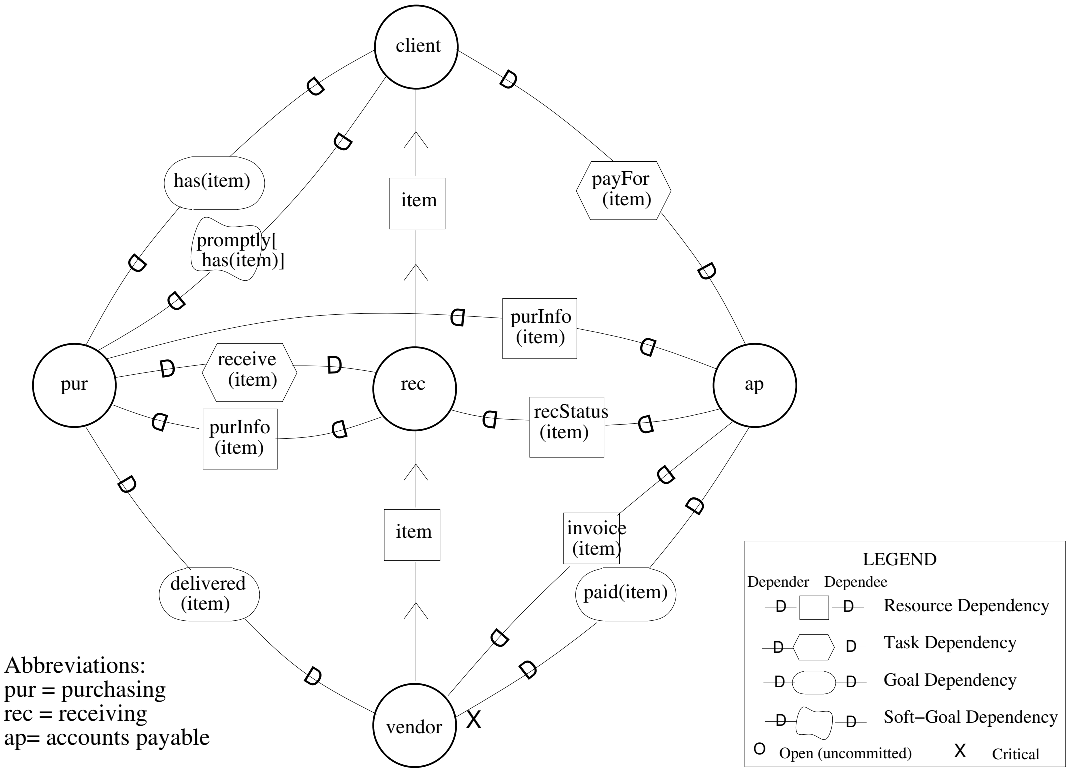

Figure 5.2: Strategic Dependency model of a goods acquisition process

reengineering should be able to express and reason about these types of intentional and strategic actor relationships.

In the SD model of Figure 5.2, the client depends on purchasing to have an item ordered. Purchasing depends on the vendor to have the item delivered, and on Receiving to receive the item. Receiving depends on purchasing information about the item. To issue payment to the vendor, Accounts Payable depends on purchasing information from Purchasing, receiving status from Receiving, and the invoice from the vendor.

The four types of dependencies serve to convey the types of work relationships that exist between actors. Since the client just wants to have the item ordered, but does not care how the purchasing specialist does it (e.g., by obtaining price quotes and choosing among suppliers), the dependency from the client to Purchasing is appropriately modelled as a *goal dependency*. Purchasing, in turn, just wants the vendor to have the item delivered, but does not care what mode of transportation is used, etc.

Purchasing's dependency on Receiving is a *task dependency* because Purchasing relies on Receiving to follow procedures such as: accept only if the item was ordered. Similarly, the client wants Accounts Payable to pay only if the item was ordered and has been received. A task dependency specifies *how* the task is to be performed, but not *why*. The depender makes the decisions. The depender's goals are not given to the dependee.

Accounting's dependencies for information from Purchasing, Receiving, and the vendor before it issues payment are modelled as *resource dependencies.* A resource is usually the finished product of some process. It is assumed that there are no open issues or decisions to be addressed.[2]

Note that resource dependency is different from a non-intentional flow. In Figure 5.2, flow of item from vendor to Receiving and from Receiving to client are modelled as non-intentional flows. This means Receiving does not care whether the item is received. An item not arriving is not viewed as a failure in Receiving. The client does not hold Receiving (directly) accountable for the non-delivery of an item. Contrast this with Accounts Payable, where if the needed resources are not available, a failure to issue payment would result.

The client's desire to have item promptly ordered is modelled as as *softgoal dependency.* There is no clear-cut, a priori criteria for what is meant by "prompt". Purchasing may have different ways for achieving promptness. It is the client who decides which is prompt enough.

The relationships in an SD model are intentional. They embody concepts of success and failure in what an agent does. Because success or failure of one agent is affected by success or failure in other agents, these relationships are also strategic. By depending on another actor for a dependum, an actor is *able* to achieve goals that it was not able to do without the dependency, or not as easily or as well. At the same time, the depender becomes *vulnerable* [Malone87]. If the dependee fails to deliver the dependum, the depender would be adversely affected in its ability to achieve its goals.

By following the chains of dependencies, one could explore the expanded possibilities that are open to an actor. From a vulnerability viewpoint, one could also use the model to determine how an actor could be affected adversely by its dependencies. We have argued that this type of intentional model is important for understanding an organization design. We now illustrate this with some examples from business process reengineering.

One can obtain some understanding about *why* processes are configured in a particular way. Suppose one asks: Why do we need Purchasing in the goods acquisition process? The workflow model of Figure 5.1 could only indicate that purchasing is there to "process" purchase requisition forms. The model is equally unhelpful for answering what would happen if Purchasing were bypassed ("obliterated"). It offers little help in identifying alternatives to having a purchasing department.

In contrast, from a Strategic Dependency model, one could tell who depends on whom, and for what. In Figure 5.2, the client depends on purchasing for meeting the goal of having an item. Purchasing in turn depends on the vendor to meet the goal of having the item delivered, and on the Receiving department to perform the procedurally defined task of receiving the item. At each point in a chain of dependencies, one can infer from the model how the actor's goal-seeking behaviour may be enhanced or restricted, based on the type and strength of the dependencies that it has. It is this deeper knowledge that is necessary to help judge potential targets for obliteration.

To answer the question: "What if the Purchasing department is removed from the goods acquisition process?", we observe from the model that the client depends on Purchasing to achieve the goal of having an item, and is therefore vulnerable with respect to this same goal. The client's ability is enhanced through this dependency because the goal can be achieved even if the client does not have the knowhow or the resources to pursue it on his own. To bypass Purchasing, the client would have to acquire the knowhow and have the needed resources (e.g.,

[2] In a more detailed analysis of accounts payable work [Suchman83], it was found that there is considerable problem solving even in apparently routine clerical work – for example, when an overdue notice is received, or when some documents are found to be missing. The dependency types of the SD model can be used to indicate who is expected to do what problem solving.

time and effort) to doing purchasing on his own.

The real-world reengineering examples cited in the introductory section of this chapter can be clarified using the SD model. (The SD model is used to described each process configuration, while the SR model can be used to describe the reasoning about alternative configurations.) In one of the companies, it was recognized that expert systems technology could be used to provide the knowhow and resource support for simple purchases. Clients do not necessarily have to depend on human purchasing specialists to order items. Thus a system was set up so that most items can be ordered directly from pre-approved vendors through the system without the help of human purchasing agents.

At the Ford Motor Company, the goods acquisition process was reengineered by eliminating invoices. The invoice is recognized as only a means to an end. An alternate, and much more effective, means to the same end was found. Instead of paying when invoiced, Ford now pays when the goods are received. This greatly simplifies the goods acquisition process. With conventional methods that focus on workflow analysis (Figure 5.1) and no explicit support for answering Why? and What-if? questions, one is more likely to end up with the less effective, "automation" approaches, which merely computerize existing (and often outdated) processes. (This has been referred to as "paving the cow-path" [Hammer90]).

The following is a sample representation in Telos of the dependencies associated with Purchasing.

```
Class PurchasingSpecialist IN ActorClass
   WITH
  goalDepended, commits
    ord: ItemBeOrdered(i:Item)
          WITH dependee
                 cl: Client
          END
  softgoalDepended, commits
    ordp: ItemBeOrderedPromptly(i:Item)
          WITH dependee
                 cl: Client
          END
  goalDepends, committed
    del: ItemBeDelivered(i:Item)
          WITH dependee
                 vdr: Vendor
          END
  taskDepends, committed
    rcv: ReceiveItem
          WITH dependee
                 rcvg: Receiving
          END
  resourceDepended, commits
    pi: PurchasingInfo
          WITH dependee
                 rcvg: Receiving
                 ap: AccountsPayable
          END
 END
```

A number of rule-of-thumb principles have been proposed to guide reengineering efforts [Hammer90]. We use two of these principles to further illustrate how the Strategic Dependency model can help bring out the distinctive features of different organizational configurations by making their intentional structures explicit.

i) "Organize around outcomes, not tasks". One common way to organize work is to group similar tasks into units with a specialized function, such as order entry, credit checking, assembly, or shipping. The problem with this structure is that while workflow passes from unit to unit, no one is responsible for the application from end to end. While each person is accountable to a supervisor in his/her own functional unit, problems that arise in between units tend to fall through the cracks (e.g., files can be misplaced, delayed, or become lost in transit). One insurance company (Mutual Benefit Life) reengineered its policy application process into a configuration in which a single person acts as a case manager and handles a customer's application from beginning to end. Using computer support, the case manager performs all the tasks associated with the application. For difficult cases, she would seek help from specialist consultants, but only for advice. The decisions remain with the case manager.

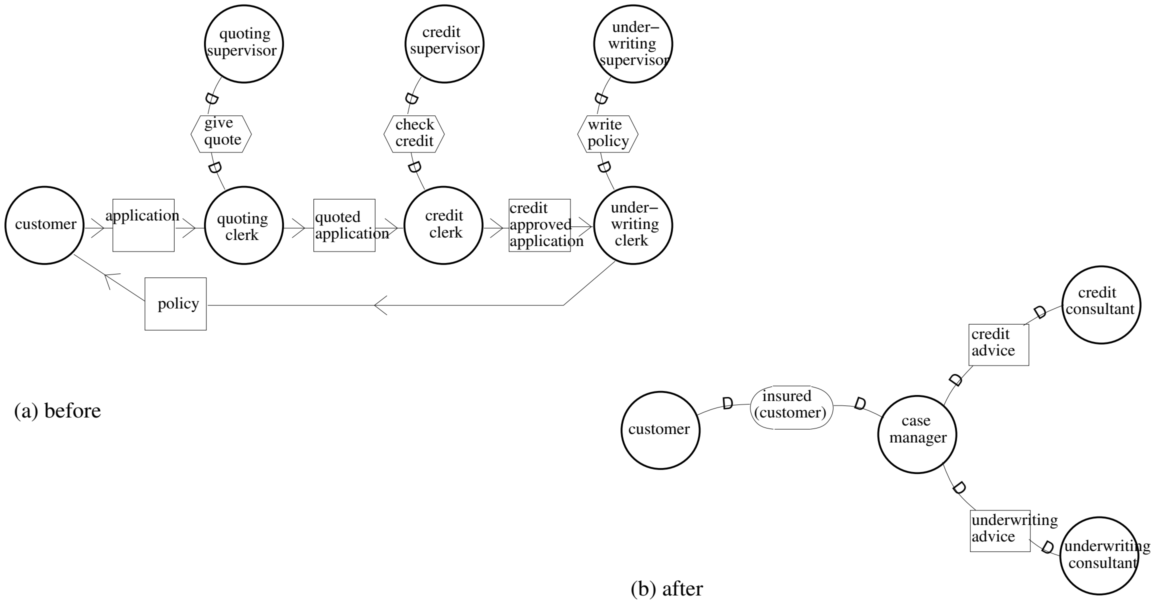

Figure 5.3: "Organize around outcomes, not tasks"

The Strategic Dependency model of Figure 5.3(a) reflects the traditional functional organization. Each processing unit (the clerks) is a dependee in a *task dependency*, but are only related to each other by non-intentional workflow. In the reengineered configuration (Figure 5.3(b)), there is a single *goal dependency* from the customer to the case manager. The case manager depends on the consultants' advice as a resource, since the case manager is the one who makes the decisions and takes action.

ii) "Put the decision point where the work is performed, and build control into the process". In hierarchically structured organizations, decisions are made in the higher levels, while the ensuing tasks are executed by the lower levels. This type of structure is often plagued with problems of delay, error, and miscommunication. The reengineering principle recommends to allow the person performing the work to make the decision. Computer networks and shared databases can be used to enable one to access information and knowhow so that one person can encompass a much broader scope of work. The case manager in the insurance company

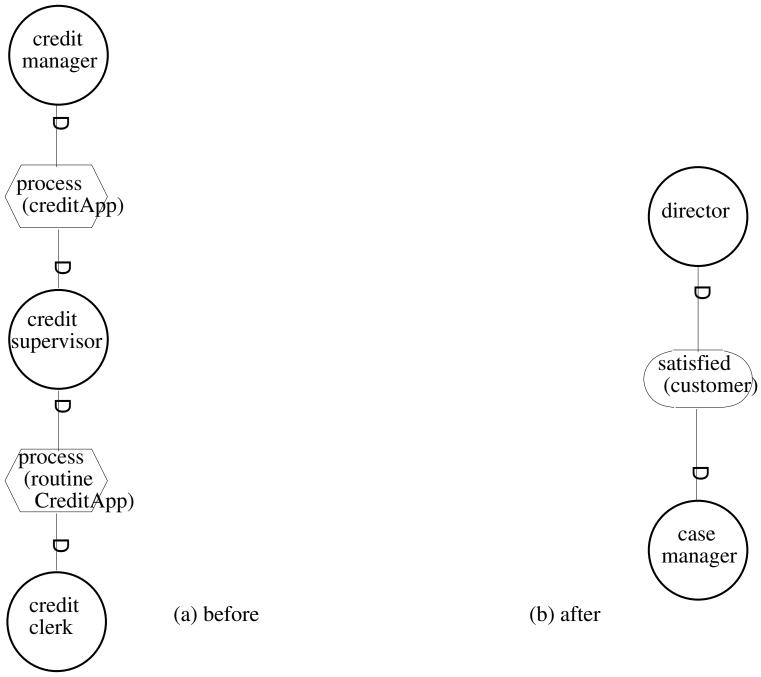

Figure 5.4: "Put the decision point where the work is performed, and build control into the process."

exemplifies this principle. She can best decide how to meet customer needs because she is closer to the customer.

Using a Strategic Dependency model, the traditional hierarchical delegation chain may be modelled as a string of *task dependencies* (Figure 5.4(a)). Under the reengineered configuration, the relationship between the case manager and her superior is modelled as a *goal dependency* (Figure 5.4(b)). She has the freedom to make decisions regarding how to meet the goal. Control is built-in because it is the outcome that matters to the depender, not the detailed activities of the dependee.

These examples help illustrate the need to understand a work organization at an intentional level. Without the deeper knowledge about intentional structure, one could not easily break away from current practice to a new conceptualization of the work process – one of the central ideas of business process reengineering.

5.3 Using Strategic Rationales in Business Process Reengineering

The Strategic Rationale model of the i^* framework provides an intentional description of a process in terms of process elements and the rationales behind them. In this section, we illustrate how the SR model can support business process reengineering by:

1. providing a deeper understanding of a current process (compared to conventional, non-intentional models) by supporting the modelling of relationships that convey the "why" and the "how",

2. facilitating the search for new and innovate alternatives to an existing business process, and

3. assisting in the evaluation of alternatives, with respect to the interests and concerns of stakeholders.

5.3.1 Deepening Understanding by Asking Why and How

In the Strategic Rationales model, we model the intentional relationships within an actor, so that we can describe and support actors' reasoning about processes.

In most modelling schemes used for modelling business processes, process elements are understood as activity steps. Process steps are connected by flows of entities (informational or physical objects) into and out of them (e.g., workflow models or SADT/IDEF0 models). In contrast, process elements in the SR model are intentional, i.e., they are inherently goal- (result-, outcome-) oriented, so that there is a notion of success and failure. Implicit in this is the premise that actors may have some freedom on how to accomplish what is described in an intentional process element.

Process elements are connected by intentional relationships. A means-ends relationship links an end to one (of possibly several) means for accomplishing it. By going "up" a means-ends hierarchy of links, one can pursue the reasons *why* a process element is needed. Conversely, the different ways (the *hows*) in which an actor can accomplishing something can be determined by traversing down the means-ends hierarchy.

The task decomposition links allow a task to be described as a set of inter-related intentional elements (not just sub-steps interconnected by input /output flows, as in decomposition hierarchies in SADT, for example). The intentional elements are elements that are needed for the task to succeed. They can be a combination of subgoals, subtasks, resources, and softgoals.

Because the SR model is intended for strategic reasoning, typically only elements that are considered strategically significant are included (i.e., those that would have an impact on stakeholder interests). The description in an SR model is therefore usually inadequate for operational use (execution) because of its incompleteness.

Figure 5.5 shows an SR model for a portion of a goods acquisition process from the client's viewpoint. To accomplish the goal of having an item, one can lease it, buy it, or borrow it. If one were to buy the item, one needs to have the budget for it, have it ordered, received, and paid for, and the ordering should be done promptly. To have an item ordered, one can order by phone, by sending a purchasing order, or by letting a purchasing specialist handle it.

The term *routine* is used to refer to a hierarchy of successive decompositions and means-ends reductions which includes only one alternative at each choice point. For example, buying an item by having a purchasing specialist order it is one routine for achieving the goal of having an item. Another routine might involve borrowing it through some particular channel. At the actor boundary, intentional elements of a routine connect up with dependency links in an SD model.

5.3.2 Identifying Alternatives by Generating New Means to Ends

Since a Strategic Dependency model has explicit representation of goals, one is led to realize that there is more than one way to do things, and to look for alternatives. The intentional representation of processes provided by the SR model facilitates the discovery of alternatives.

Advocates of business process reengineering have pointed out that organizations often follow rules that are outdated, e.g. "to order an item, send out a purchase order", or "pay for an item

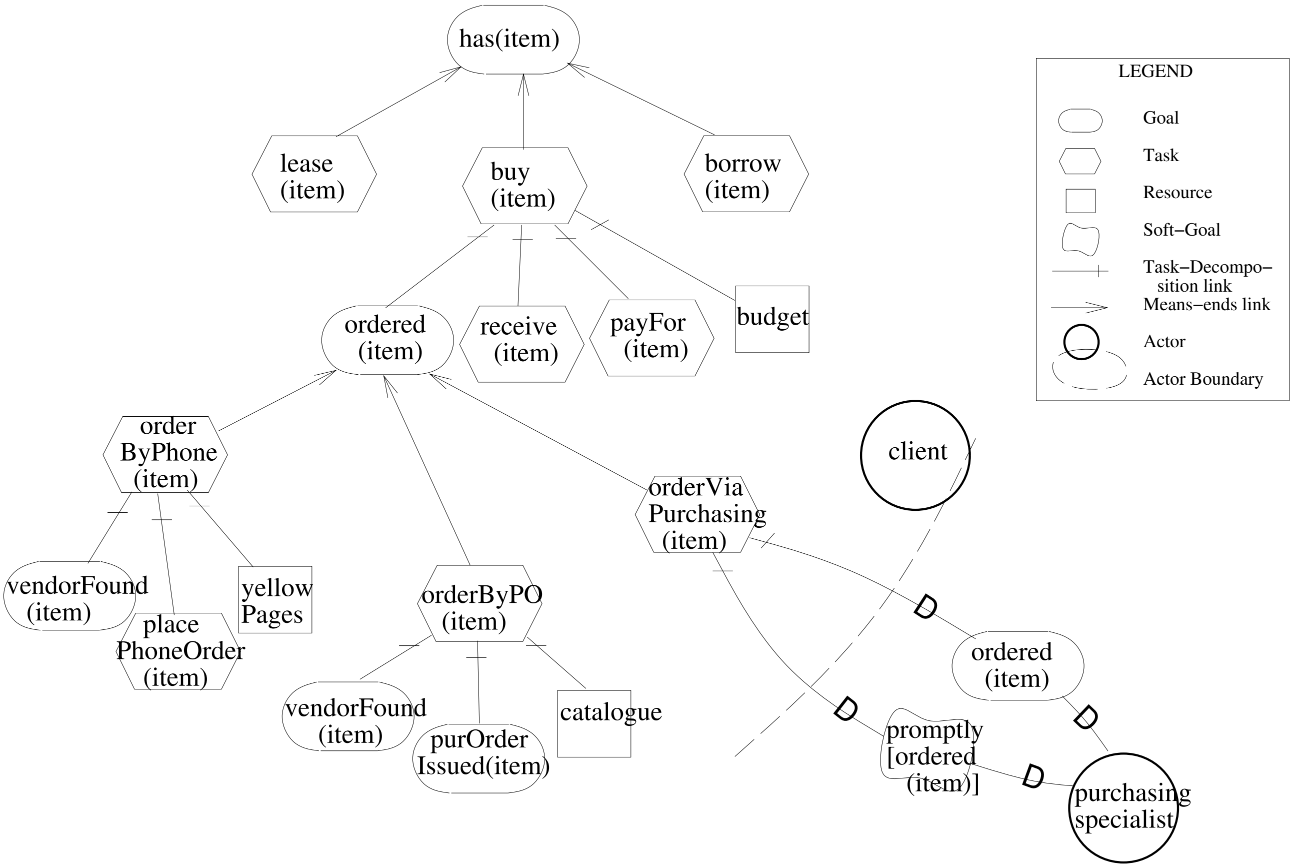

Figure 5.5: A Strategic Rationales model showing alternative ways of accomplishing "having an item"

only when an invoice is received". In this context, a rule has the connotation that it is the only way to do things. One is not aware there are other ways to do things because the goal is not explicit. But if the goals are explicit, then a rule could easily be seen as *one* way of achieving the goal. A rule is a generic link from means to ends. There can be other rules that provide other means to lead to the same ends.

An example of rule representation is as follows.

```
Class CanOrderByExpertSystem IN Rule
 WITH
  purpose
    ord: ItemBeOrdered(i:Item)
  how
    es: OrderViaPurchasingExpertSystem
  applicabilityCondition
    expertSystemCanHandle:  $ SimplePurchase(ord) and LowQuantity(ord) $
 END

Class OrderViaPurchasingExpertSystem IN TaskClass
 WITH
  goalDep
    esord: ItemBeOrdered(i:Item)
      WITH dependee
```

```
            pes: PurchasingExpertSystem
      END
 END
```

The SR model provides concepts that support the systematic exploration of alternatives. When one wishes to explore different ways for accomplishing something, one *raises* it as an *issue*. For example, one could raise as an issue whether a client could make a purchase using his budget allocation for next year (supposing the budget for this year is exhausted). To *address* this issue, the client would try to find a routine (a composition of elements) that accomplishes the desired result. This may include preparing a statement explaining the situation, and submitting it to appropriate authorities for approval. This routine needs to be *workable* – that he can write a statement, and there are people who are in positions to give such approvals. The routine also needs to be *viable* for various stakeholders – for the client, that it does not take too longer to prepare a statement, and that it does not take too longer to approve; for the accounting department and the controller, that it does not violate the intent of budgeting and control too much.

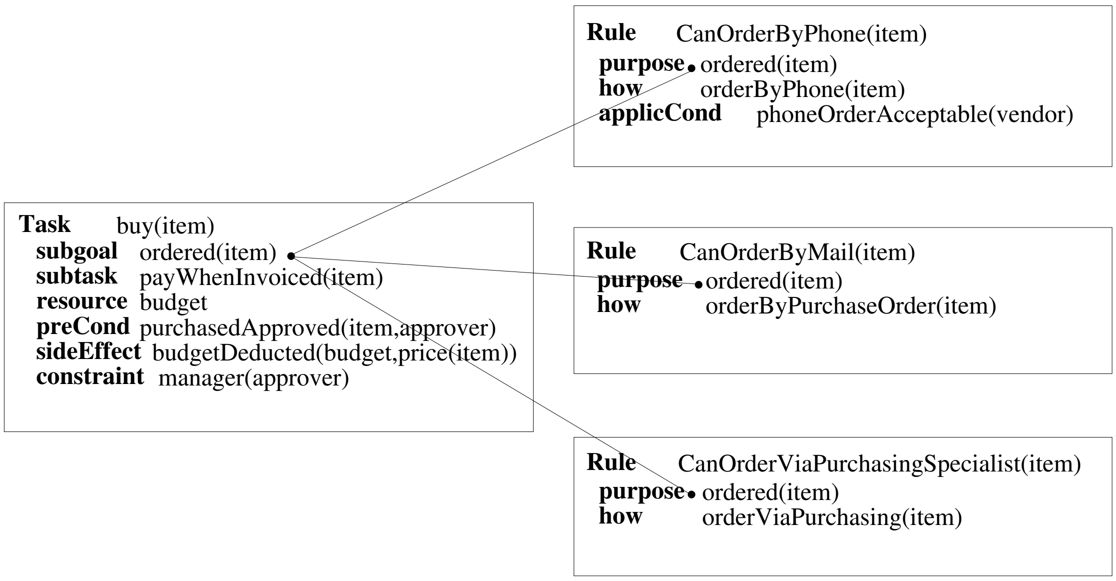

Figure 5.6: Rule matching (syntax simplified)

The identification of alternatives is facilitated by the use of rules. Consider the example of ordering an item. The different ways of ordering expressed as means-ends links on Figure 5.5 can be seen as applications of generic means-ends relationships that are potentially applicable in other contexts. There may be several rules all with `ItemBeOrdered` as the "purpose", but offering different "hows".

To discover the different ways that one can accomplish the subgoal `Ordered(Item)` in the task `Buy(Item)`, we search for rules that have `Ordered(Item)` as its "purpose". The "how" attribute of these rules point to tasks that are the different ways. (Figure 5.6).

Because of the explicit representation of intentional elements and rules, one could explore new possibilities more systematically, by generating and evaluating alternatives, as well as by coming up with new rules. One way in which new rules can arise is when new technology becomes available offering new abilities.

For example, the new rule indicating that expert systems technology has the capability to provide the knowhow and resource for simple purchases might appear as a fourth option in the example of Figure 5.6. One company has indeed reengineered its purchasing process in this way [Hammer90], as cited in the first section of this chapter.

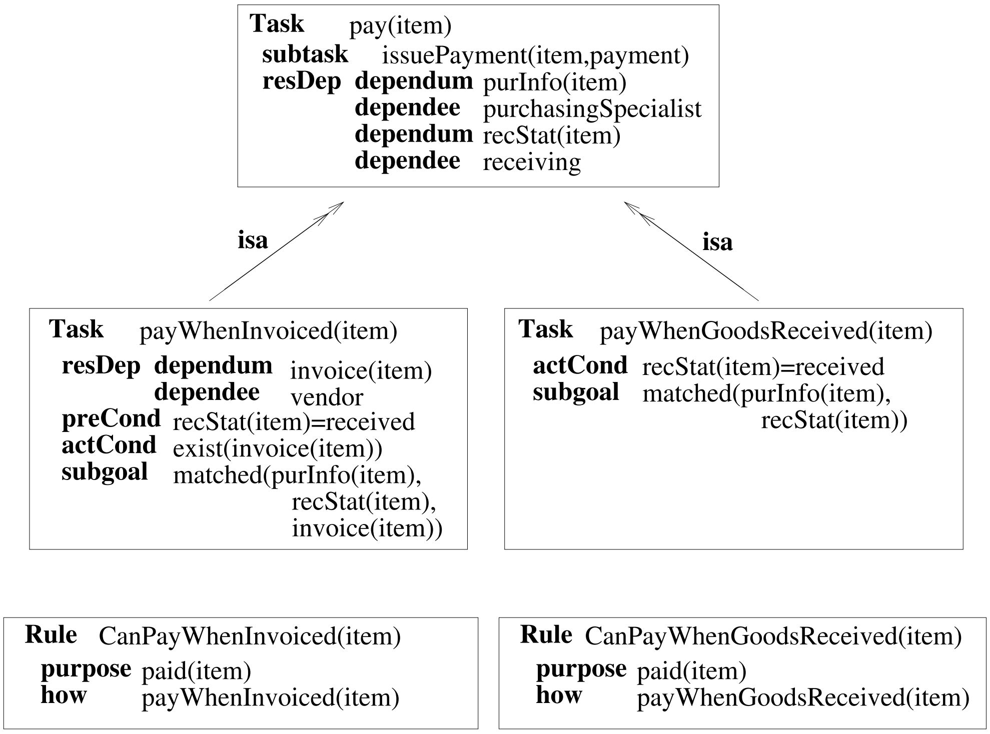

Figure 5.7: Two payment tasks (and corresponding rules) as specializations of a generic payment task (syntax simplified)

The search for new rules can be done systematically along an IS-A hierarchy. For example, the approach taken by the Ford Motor Company in its effort to reengineer its goods acquisition process (as described in [Hammer90]) may be represented as in Figure 5.7. A more general version of the rule is first sought, then another specialization is found which achieves the same goal. The current rule for payment is recognized as one way to achieve the goal of "paid(item)". A new rule which eliminates invoices is found. The new rule eliminates the resource dependency on invoice, and simplifies the subgoal (from matching three items to matching two items). It turns out that this is a much simpler operation and can be accomplished mostly by computer.

5.3.3 Exploring and Evaluating Impacts of Alternatives

The reengineering literature tends to emphasize the benefits of radically new ways of doing work. However, when new alternatives are proposed, one must also consider its implications on many other factors. The SR model facilitates the identification of cross-impacts with other issues by the use of multiple means-ends links to softgoals. Means-ends rules can be used in reverse (given means, identify the ends) to find out what other goals are affected when adopting a new alternative. Such links may be traced to other affected actors (stakeholders) through the SD model.

For example, when looking for new ways to make payment, the original process design objectives may have been a faster process and the reduction of errors, in order to reduce costs. However, these objective have cross impacts on other issues such as cash flow, and financial control and accountability. These cross-impacts are identified using rules. The identified issues are then raised and addressed by the various stakeholders. (Figure 5.8).

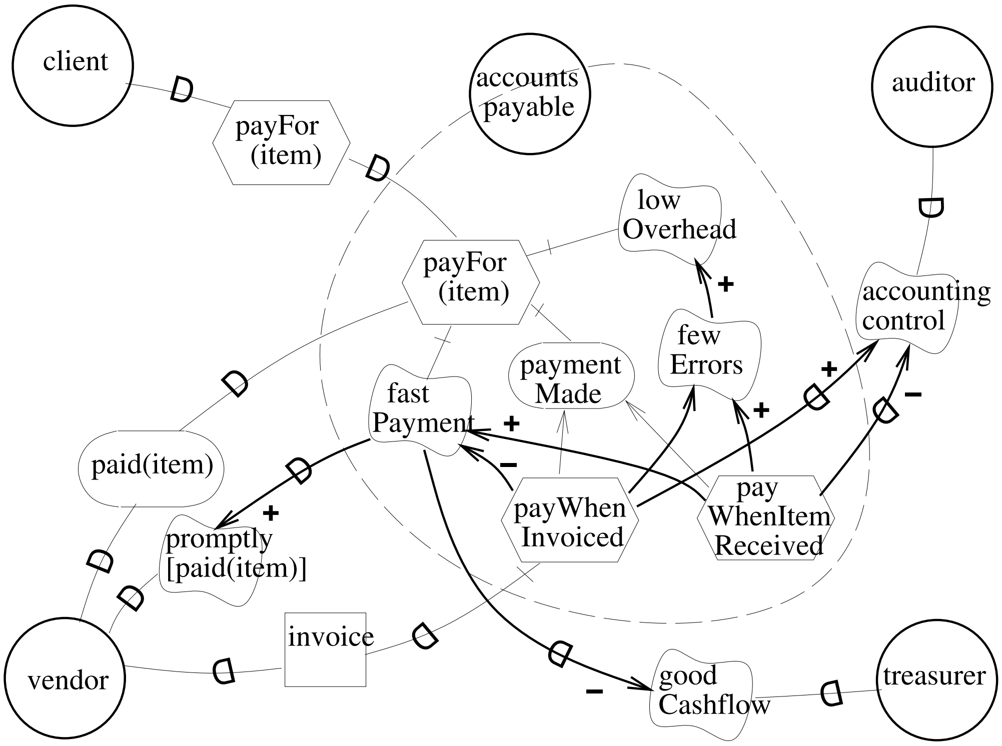

Figure 5.8: Identifying cross-impacts using softgoals

Figure 5.9 shows a simplified scenario of the use of the ***i**** framework in a business process reengineering effort. From an SD model of the existing process, one fleshes out the internal means-ends linkages to connect external dependency relationships. Alternatives are generated and explored. Cross-impacted issues and stakeholders are identified. These steps are iterated over and the results evaluated. A new process design is arrived at when issues have been sufficiently addressed and have become settled.

5.4 Discussion

The ***i**** framework improves on the state-of-the-art in business process reengineering in several respects:

(i) A richer process model based on intentional concepts. In the BPR literature, there are frequent hints at goal-oriented concepts and means-ends reasoning, such as the need to "understand why", the primacy of the "value to the customer", and "organizing work around outcomes". It could be argued that intentional concepts and means-ends reasoning are central intuitions underlying BPR. Despite these underlying intuitions, the models commonly used in BPR are primarily those used in conventional systems analysis, i.e., activity and workflow models. These cannot convey intentional concepts nor do they support goal-oriented reasoning.

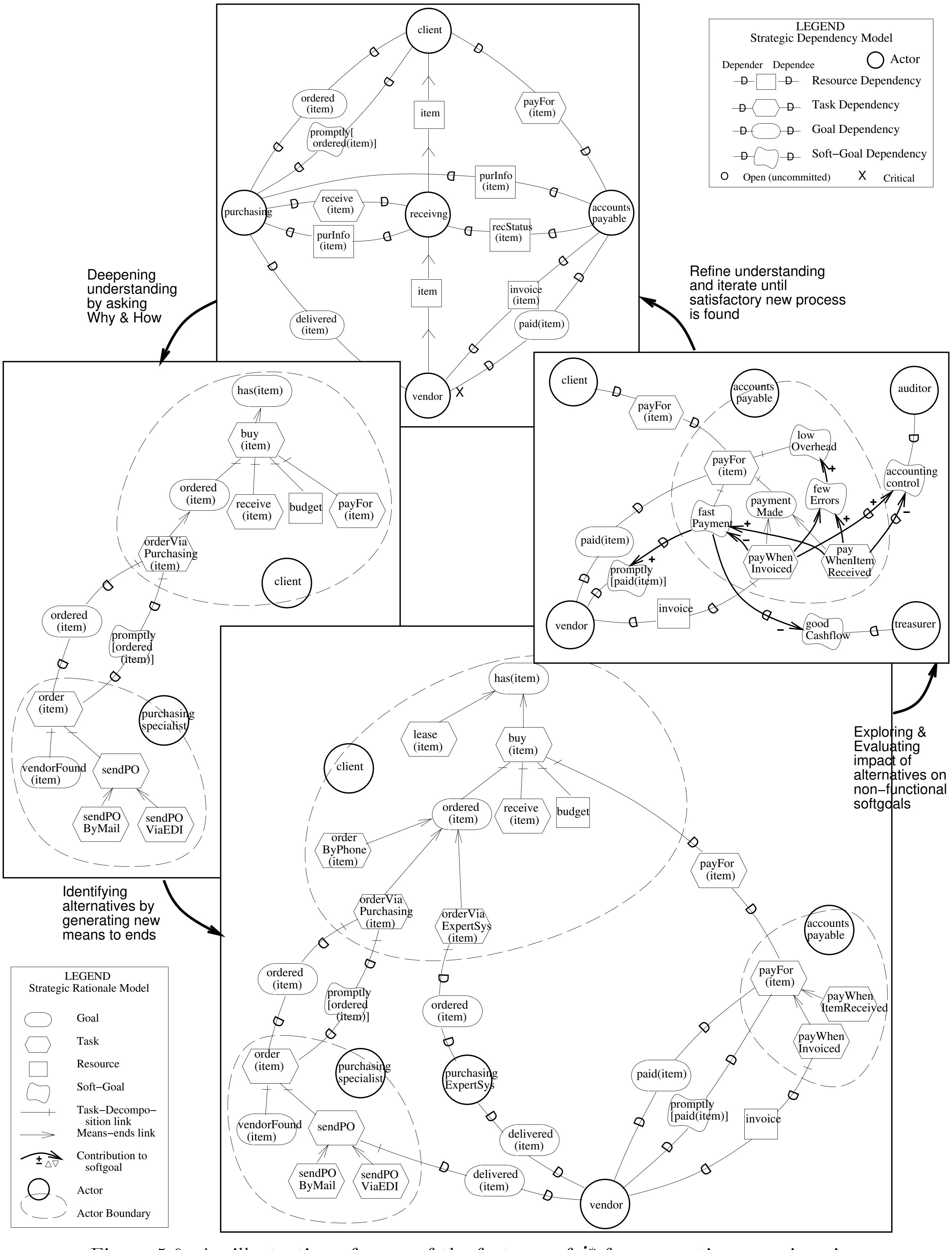

Figure 5.9: An illustration of some of the features of i^* for supporting reengineering

The $\boldsymbol{i}^*$ framework offers a model for describing processes (SD) that is inherently intentional, and a model for expressing and reasoning about means-ends relationships in processes (SR), thus providing explicit support for some of the central concepts in BPR. Furthermore, modelling a process in terms of intentional relationships between actors provides an abstraction that allows detailed actions to be left open, acknowledging the inherent freedom of actors.

One model proposed for BPR that has an implicit intentional connotation is the Action Workflow model of Medina-Mora et al. [Medina-Mora92]. In this model, each process "step" consists of a four-phased loop between a customer and a performer. The four phases are proposal, agreement, performance, and satisfaction. This pairing of customer and performer may be compared to the depender–dependee relationship in the Strategic Dependency model, and the notion of satisfaction suggests an intentional dimension. However, the model is informal and does not have an intentional semantics. The patterns of customer–performer relationships are not analyzed for strategic implications. In $\boldsymbol{i}^*$, an intentional relationship need not be accompanied by a flow or any explicit action between the two actors.

(ii) Systematic search for process alternatives. It is commonly acknowledged that current BPR practice is primarily *ad hoc*. There is no systematic method for arriving at new process solutions. Practitioners are guided only by rules-of-thumb, anecdotal accounts of success stories, or benchmarking of similar processes in other organizations. Since solutions that work for one organization do not necessarily work for another, practitioners have admitted that a majority of BPR efforts fail.

The intentional models of $\boldsymbol{i}^*$ support systematic search for alternatives through means-ends reasoning and hierarchical decomposition of tasks into their intentional elements. Generic means-ends knowledge is encoded as rules with applicability conditions. A knowledge-based approach further helps to organize knowledge along dimensions of classification, generalization, aggregation and time, allowing relevant knowledge to be brought to bear during process design.

The approach of Malone et al ([Malone93][Lee93]) uses representations of processes organized along the generalization/specialization dimension to help process designers come up with solutions. Goals are used primarily as evaluation criteria. Different types of dependencies among *activities* are viewed as coordination mechanisms, not strategic relationships. (The types of dependencies mentioned include shared resources, producer/consumer relationships, simultaneous constraints, and task/subtask.) There is no intentional semantics for actors nor reasoning about strategic implications.

(iii) Systematic evaluation of process alternatives with respect to stakeholder interests. In the BPR literature, it is often said that the real challenge in a BPR effort lies in the implementation phase ("change management" in introducing the new process). It is relatively easy to come up with new process designs, but it is much harder to make them work, i.e., to get all parties to agree to the new process (achieving "buy-in"). This situation would seem to suggest that stakeholder concerns are often inadequately identified and analyzed during process design, so that the chances of success are seriously compromised. The lack of systematic frameworks for modelling and analyzing the strategic interests of stakeholders perhaps contributes to this unfortunate situation in the state-of-the-practice.

In the $\boldsymbol{i}^*$ framework, the SD model supports the systematic identification of stakeholders and their interests and concerns. The SR model supports the systematic evaluation of alternatives through the concepts of ability, workability, viability, and believability. "Softer" issues can also be dealt with when reasoning about strategic interests, allowing for qualitative reasoning, identifying correlated issues, and making tradeoffs. These aspects of the framework draw on a framework originally developed for dealing with non-functional requirements in software engineering.

(iv) Connecting strategic business reasoning with information systems development. Current models in BPR are mostly informal, so it is difficult to relate business reasoning to information systems decisions. Yet successful BPR often rely on rapid development of information systems to support the new process. The formal representations used in $\boldsymbol{i^*}$ to support reasoning about business processes are compatible with knowledge-based software development techniques (e.g., [Jarke92a]). New business initiatives can be mapped quickly into software requirements, and to designs and implementations. More importantly, incremental business process innovations can be rapidly supported by changes in technical systems. For information system developers, a systematic understanding about business issues and rationales would provide a deeper basis for software evolution and reuse. [Gruninger94] outlines a BPR approach also using formal modelling and reasoning, but does not deal with the dimension of strategic relationships among actors.

Chapter 6

Application III – Organizational Impacts Analysis[1]

6.1 Introduction

Although information systems are usually designed to achieve some organizational objectives, such as improving productivity or service, the actual outcome of introducing a computer-based system in an organizational setting is often different from the one intended or anticipated [Lyytinen87]. Many systems fail to achieve their intended effects, or even have serious negative impacts [Lucas81][Grudin88][Neumann94]. The interplay of forces among organizational actors and technical systems is the subject of study in the area of Organizational Impacts Analysis.

The study of organizational impact needs to be concerned with many types of issues, including the infrastructure needed to get work done [Kling82] [Suchman83] [Gasser87] [Clement90], power and politics [Keen81] [Markus83], managerial control [Clement86], privacy [Clement94], organizational culture and structure [Orlikowski92] and others. An understanding of these issues regarding a particular organizational settings would have important implications for the design and eventual success or failure of computer systems introduced into the setting.

Although this is an important area of research, it faces a number of difficult challenges:

- In attempting to analyze a given organizational setting, there are a large number of issues to be considered. There are often many players (stakeholders), each with a number of concerns. Each organizational setting has a great many features. Currently these are described using narrative text. Reasoning is presented in argumentative prose. While the richness of organizational setting often can only be conveyed fully by such rich media as unstructured text (or even video), the information and knowledge thus conveyed is hard to organize and manipulate. Analysts can easily be overwhelmed by the amount of detail involved ([Kling87, p. 345]).

- Understanding an organizational setting often require a number of perspectives. Given the complexity of issues within one perspective, it is difficult to bring multiple perspectives together to draw conclusions from their combined insights.

- The insights gained from organizational impact studies are hard to apply to the design of systems. There is apparently too great a gap between organization analysis and system

[1]This chapter is an abridged and updated version of [Yu93b].

analysis – both in the subject matter (organizational actors and their issues and concerns versus system boxes with inputs and outputs) and in the way they are expressed (unstructured text versus structured charts or more rigorous formalisms).

- Unlike in the design of mechanistic systems, the knowledge used to design organizations are often soft and tentative, with many mitigating conditions. The knowledge from organizational impact studies may be taxonomic distinctions or weakly predictive or prescriptive.

What i^* offers. To address the above issues, $\boldsymbol{i^*}$ can be used to analyze organizations and the impact of computing systems by taking an approach as follows:

- The underlying knowledge-based approach in $\boldsymbol{i^*}$ can be used to organize and relate the large amount of knowledge that may be involved in understanding an organizational environment. The knowledge structuring dimensions of classification, generalization, aggregation, and time will serve to provide structure, accommodating different levels of granularity.

- The key concept of intentional, strategic actor in $\boldsymbol{i^*}$ provide a natural focal point for modelling organizational issues and concerns, and also allow for the modelling of processes as a rich network of relationships among actors.

- $\boldsymbol{i^*}$ combines the formal and the informal, allowing concepts such as power and control, or ease of use, to be dealt with in the framework, without necessarily requiring precise definitions.

- $\boldsymbol{i^*}$ assumes a highly interactive modelling, analysis, and design approach, so that weak knowledge can be suggested to help in the interpretation of situations, or in the selection of design alternatives, with the analyst/designer making the necessary judgements.

- The formal representation of $\boldsymbol{i^*}$ on the one hand, and its use of organizational concepts on the other, serve to bring organizational analysis and system analysis closer together. Organization analysis and design can become an integral part of the system development cycle.

In this chapter, we use an example from the organizational impact analysis literature to illustrate some of the issues that can arise when a computer system is introduced in an organization, and the kinds of impact that it can have. We show how the Strategic Dependency model can highlight the differences in work relationships (1) before the system was introduced, (2) in the intended organizational usage environment, and (3) in the actual usage setting that eventually emerged. The Strategic Rationale model provides a way for presenting and analyzing the organizational actors' reasoning about these sets of relationships.

6.2 Modelling Strategic Dependencies in Organizational Impact Analysis

The example setting concerns the introduction of a design tool called Trillium into the machine interface design community of a large American corporation [Blomberg86]. Trillium is a computer-based design environment used in the creation of the layout and logic of interaction for Control/Display interfaces on machines such as copiers and printers. It allowed designers to build prototypes for user interfaces, so that they can experiment with different designs before committing to an implementation on the target machine (the product) [Henderson86].

[Blomberg86] described how the new technology altered the patterns of work relationships in the organization. The organization evolved in a way that resulted from an interplay of many factors, and did not follow the straight forward path that was originally intended. Further, the way the tool was used in the organization impacted the demands and requirements on the tool itself, and influenced its evolution. [Blomberg86] described how two different groups evolved along different paths in response to the introduction of Trillium. We will only use one of these to illustrate our framework.

In this section, we show how the SD model can be used to describe the different social configurations in which the Trillium tool was embedded. The salient features of these configurations become apparent through the use of the SD model. The impact of the introduction of this computer-based technology is shown in terms of how it altered and rearranged work relationships, including how work roles are associated with positions, and the types of dependencies among them.

Designers were responsible for coming up with the user interface (UI) of the product, including the look and feel and the behaviour. They were typically trained as industrial designers or human factors psychologists and had little or no computer programming background. Before Trillium was introduced, designers used pencil and paper to specify the look and feel of the interface, and used flow charts to specify the behaviour of the interface.

These designs were given to programmers who would then develop the software to implement the interface according to the specifications. Since the paper descriptions were often imprecise and informal, designers expected programmers to consult them for clarifications during the implementation process.

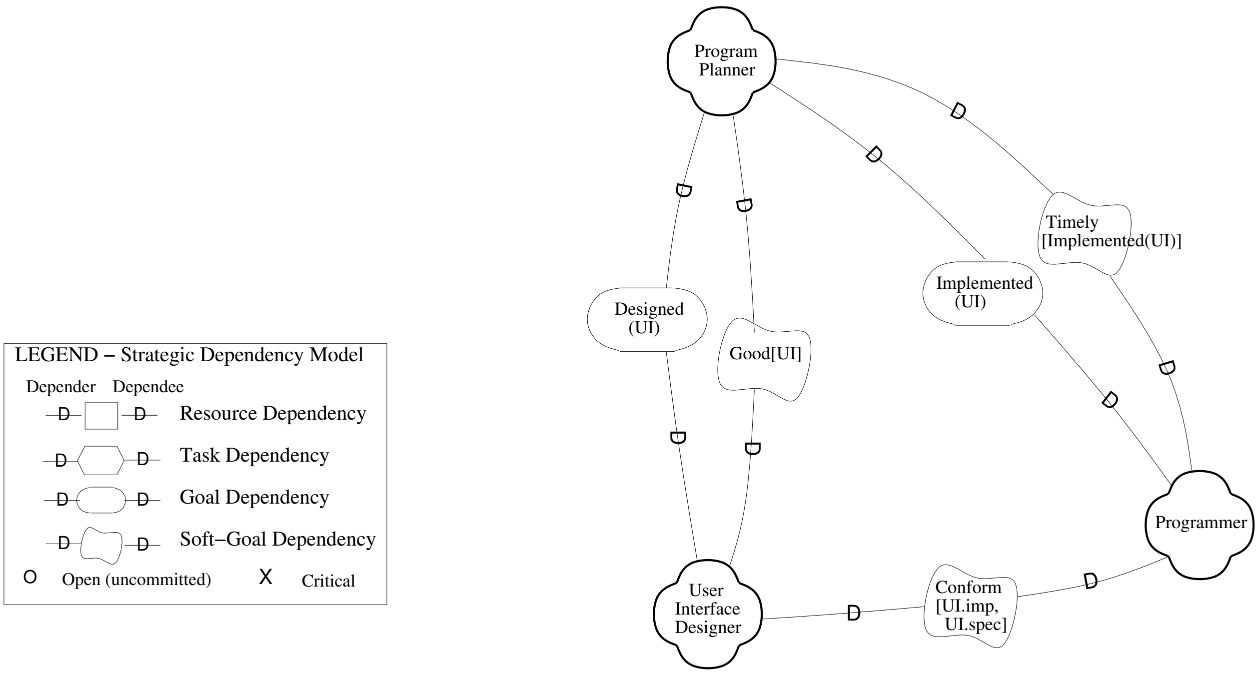

Figure 6.1: A Strategic Dependency model of the initial configuration (before Trillium)

We represent the pre-Trillium configuration of work relationships in a Strategic Dependency model in Figure 6.1. Program planners depended on designers to design the user interface (UI) for a new product (e.g., a copier), and on the programmers to produce implementations that would run on the target machine. These are modelled as *goal dependencies* since the dependees

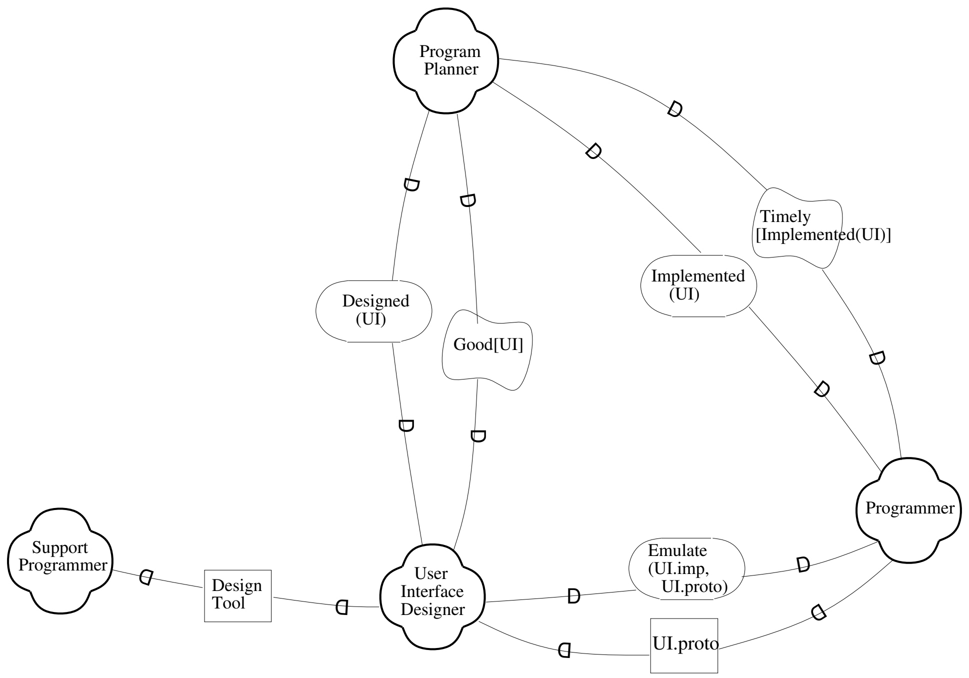

Figure 6.2: A Strategic Dependency model of the intended configuration (after Trillium)

were expected to use their knowhow freely to achieve the goals – namely, that there will be a UI design, and an implementation of it. There are also accompanying softgoals – that the UI design be a good one, and that the implementation be completed on time for product introduction.

The nature of the relationship between designers and programmers turned out to be of crucial interest in this study. Since designers were committed to producing good user interfaces, they were dependent on programmers to bring their design concepts into concrete realization. If designers were able to describe their designs precisely (e.g. as in a formal notation), they could adopt a *goal dependency* relationship with the programmers. The dependum would be something like `Meet(UI.imp, UI.spec)`, where there are reasonably clear-cut definitions and criteria for deciding whether an implementation meets the specifications.

Since the existing methods of specification consisted of informal diagrams and descriptions, the specifications were often incomplete and open to interpretation. Designers expected programmers to consult them during the implementation process, until they were sufficiently satisfied that the implementation meets the specifications. This type of relationship is modelled as a *softgoal dependency* in the SD model – `Conform[UI.imp, UI.spec]`. (The choice of terms such as "meet", "conform", and "emulate" is only for mnemonic purposes; what really matters is the dependency type.)

With the introduction of Trillium, it was hoped that designers would be able to do their own implementations to some extent, producing a working prototype. The prototype would be a more precise representation of the intended design. Programmers would then reproduce the behaviour and look of the user interface on the target machine. This would reduce or eliminate the need for programmers to iteratively clarify the designers' intended conception of the design. Also, the design and implementation cycle could be shortened since design concepts could be tested on the prototype before they were implemented on the target machine.

The SD model of this intended configuration is shown in Figure 6.2. The main feature to note is that the relationship between designers and programmers is now a *goal dependency*.

The idea was to use the Trillium tool to build a prototype that would embody the working behaviour of the UI design. The programmers would then be asked to make the implementation produce the same behaviour as the prototype. Designers would become users of the Trillium tool, via a *resource dependency* on the software engineers who designed and supported the tool (the "supporter" position).

When Trillium was introduced, the work relationships that emerged was different from the intended configuration. It turned out that the designers found the Trillium tool to be too hard to use in their work, even though the tool was designed with users with no computer background in mind. Designers also did not have the incentive to spend a great deal of time to learn and to try to obtain the desired effects from the Trillium tool. After several months, some designers were still unable to complete their design tasks using the tool.

The actual organizational configuration that evolved involved the creation of a new position called a "Trillium implementor". The experienced designers would go back to doing design on pencil and paper. They would discuss their designs with Trillium implementors, who would prototype them using the Trillium tool. Trillium implementors were persons who were trained as designers but had become specialists in using the Trillium tool. They were either designers with a special aptitude for computers, or were people hired specifically to use Trillium to implement interfaces.

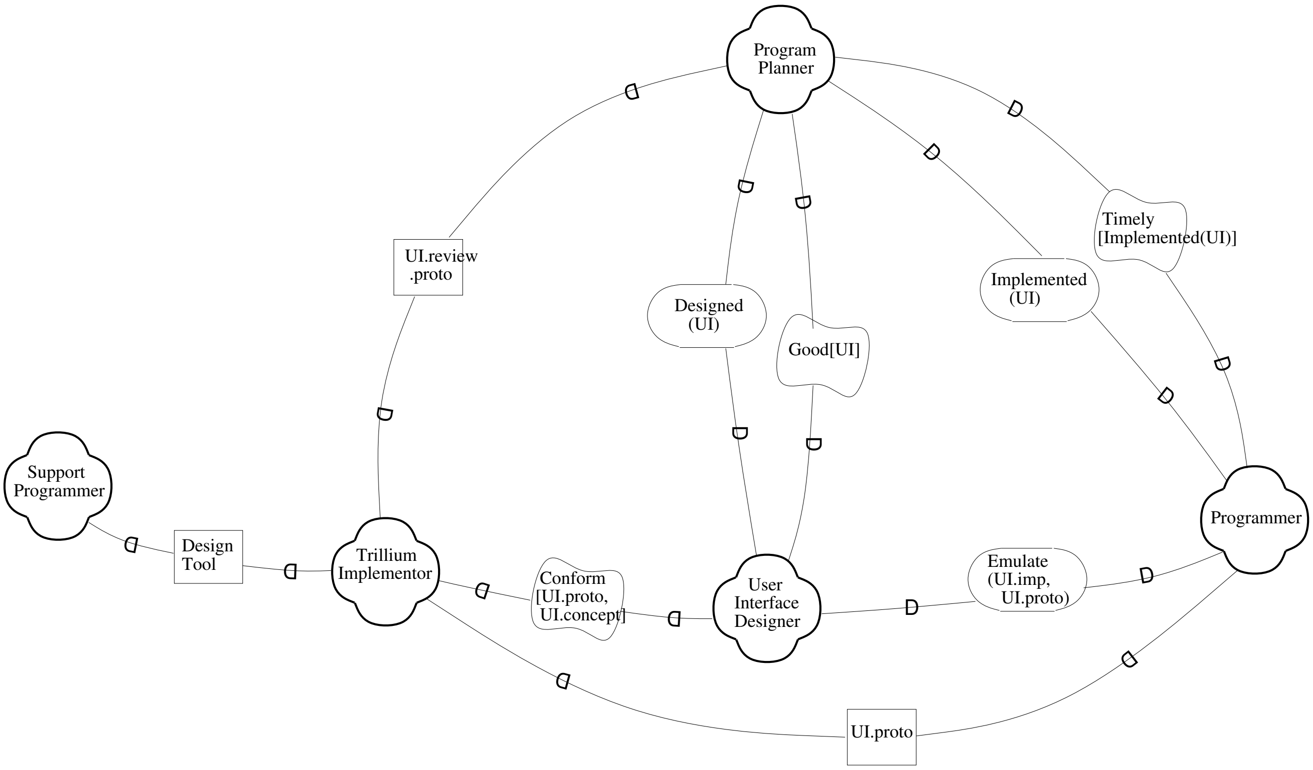

Figure 6.3: A Strategic Dependency model of the emergent configuration (with Trillium Implementors)

The SD model for this emergent configuration is shown in Figure 6.3. Designers depend on Trillium implementors to build a prototype that conforms to their design concepts. Since Trillium implementors were knowledgeable about interface design, the *softgoal dependency* relationship works well between them. Programmers now take the prototype from Trillium implementors (instead of directly from the designers), but their job is still to produce a target machine implementation that accurately reflect the designers' design concepts (a *goal dependency*).

Support programmers who developed and maintained the Trillium tool now served the Trillium implementors, instead of serving the designers directly. This configuration works better because Trillium implementors are more adept in computer use, and did not require as much support.

The emergence of the position of Trillium implementor created an *opportunity* for program planners. Once Trillium implementors were *able* to produce prototypes, the program planners also wanted to use them for the purpose of project reviews. The prototypes were used to gain support for the product, and to argue for allocation of more resources to the user interface. This is modelled as a *resource dependency* from program planners to Trillium implementors.

From the description in [Blomberg86], one could recognize many of the key concepts offered by the ***i**** modelling framework. *Dependence* is one of the key concepts used to explain many features of the organization. Organizational members depended on each other for achieving goals such as producing good user interfaces on time. Dependence is understood *intentionally* in that goal achievement would be affected if the dependencies did not hold up.

The concepts of *roles, positions,* and *agents* also provide useful distinctions in understanding how the organization responded to the new technology. Before Trillium was introduced, the organization consisted of three (types of) positions – program planners, designers, and programmers. The designer position was occupied by individuals (agents) with training in industrial design or human factors psychology, while the programmer position was occupied by computing professionals. A major part of the organizational impact of Trillium was how it altered and rearranged work roles that are associated with positions, and the types of dependencies among them.

6.3 Modelling Strategic Rationales Behind Organizational Change

The Strategic Dependency models in the preceding section do not show the actors' reasoning that led the organization to evolve in the way it did. We now re-express (portions of) the reasoning given in [Blomberg86] in a Strategic Rationale model.

A major task of program planners was to oversee the development of the UI of the product. The planners did the planning and reviewing themselves, but delegated the designing and implementing to designers and programmers respectively, through *goal dependencies*.

For a designer, the task of designing consisted of arriving at a design, then getting the design implemented. These are modelled as two subgoals of the main task of design. There are different ways for achieving these subgoals. The crucial differences among the three configurations described in the preceding section were in how the design was conveyed to programmers for implementation. These different ways evaluated differently with respect to the softgoals. An important softgoal is that the UI be good. Factors that contributed to a good UI in the final product included ease of experimentation during the design stage, ease of communication between designer and implementor, and designers having control over the implementation process.

In the initial configuration (before Trillium), the designs were implemented based on the pencil-and-paper specifications supplied to programmers. Programmers were supposed to consult with designers during implementation to clarify their understanding of the specification. In this configuration, designers had fairly good control over the design. However, since the programmers did not have good appreciation of UI design concepts, communication between designer and programmers were difficult.

In fact, programmers often did not consult with designers but went ahead to add their own design decisions. Also, by appealing to their timeliness commitment to the program planners, the programmers were able to make it difficult for designers to make changes to the UI design once the programmers were given the specification. In effect, the programmers were operating on a *goal dependency* relationship with the designers, not the *softgoal dependency* relationship that the designers expected. The programmers were free to make decisions about how the final interface looked like, usurping the control that the designers needed to have to ensure their conception of a good design. The programmers succeeded in enjoying a powerful position with respect to the designers (the "+" link from `Meet` to `Power`), with the opposite effect on the designers' side.

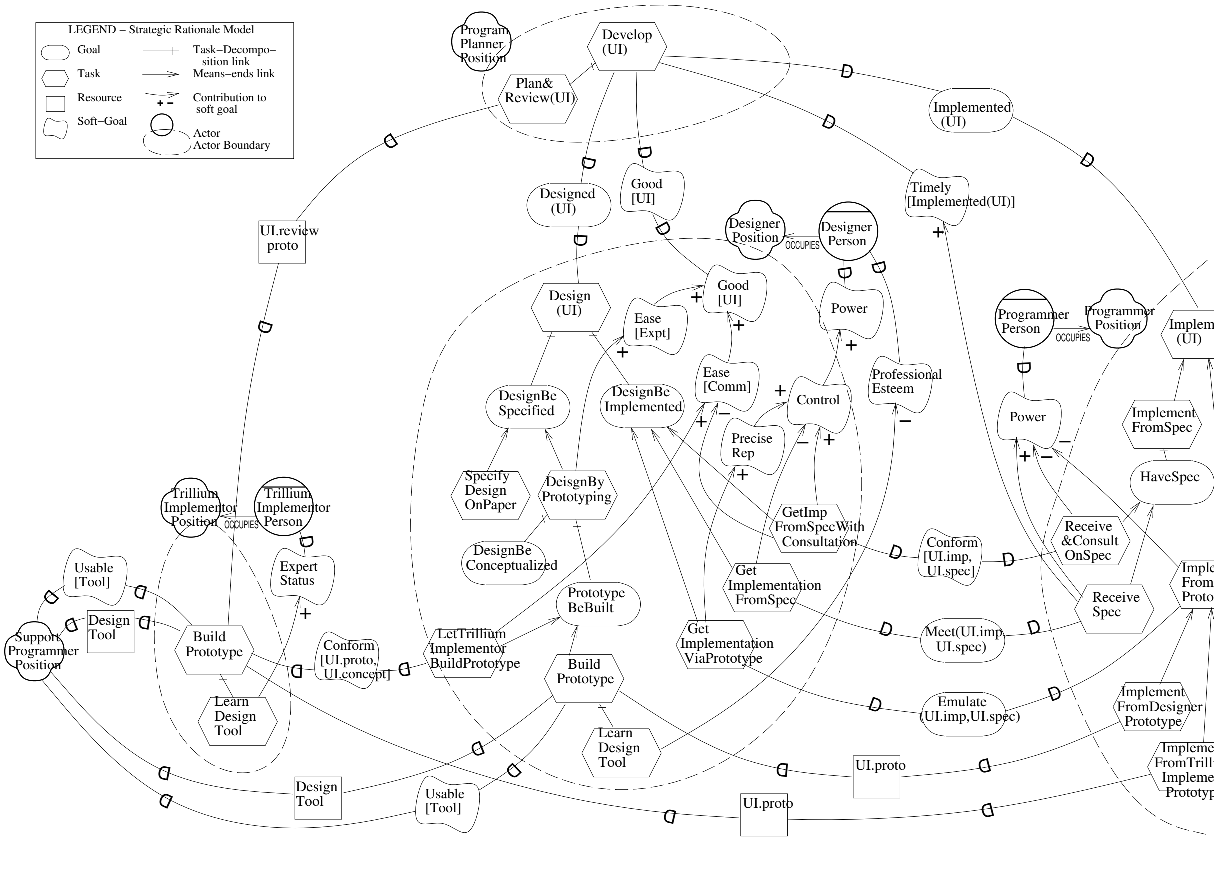

Figure 6.4: A Strategic Rationale model for the Trillium example

Trillium was intended to make experimentation with different designs faster and easier, thus contributing to better designs (and also shorter UI development cycle time, not shown in diagram to reduce clutter.) Also, the designers would be able to give the programmers a much more precise representation of what they wanted, in the form of a prototype that would demonstrate the working behaviour of the UI design. The communication problem between designers and programmers could be circumvented. Given this more precise representation, the designers would be satisfied with a *goal dependency* relationship with the programmers (`Emulate(UI.imp,UI.proto)`) since it would be clear whether a finished implementation was correctly emulating the behaviour embodied in the working prototype. This would give the designers the control that they wanted, and would have the added benefit of shifting power from the programmers back to the designers.

However, when the designers started trying to do design using Trillium (`DesignByPrototyping`), they found out that the usability of the tool was unacceptable. Further, they felt that learning to use the tool conflicted ("–" link) with their professional self-image of what designers were supposed to spend their time on, and distracted them from their proper role of doing actual designing.

The solution that finally emerged was one that involved an alternative way to meet the goal `PrototypeBeBuilt`. This was to have some members of the team become specialized expert users of the Trillium tool, so that the designers could "`LetTrilliumImplementorsBuildPrototype`". Since Trillium implementors had UI design background, it was easy to convey design concepts to them ("+" for `Ease[Communication]`). A *softgoal dependency* between designers and Trillium implementors allowed design concepts to be tried out and refined iteratively on the prototype.

Trillium implementors themselves were not averse to learning to become proficient in using Trillium. Since their primary role was to use Trillium to produce prototypes, they perceived efforts spent learning to use Trillium as furthering their professional expertise ("+" link) and were able to overcome the usability problem that the designers had.

Once the prototyping process became viable with the team approach, program planners began to use the prototypes for reviewing and evaluating UI designs at an earlier point in the development process. This also allowed designers to make convincing arguments about innovative design concepts, and were able to argue for better allocation of resources to their efforts, further improving their power within the organization (not shown in model). By being dependees to program planners as well as to designers, Trillium implementors were able to consolidate their positions in the organization.

6.4 Discussion

The $\boldsymbol{i}^*$ framework represents an initial attempt to bring organization modelling and analysis within the scope of conceptual modelling and knowledge representation techniques. In this section, we discuss some of the strengths and limitations of the $\boldsymbol{i}^*$ approach for organization analysis.

(i) Limitations to modelling. A model necessarily highlights certain aspects of the real world, while omitting others. It offers a specialized vocabulary for expressing and reasoning about a selected domain, but has built-in assumptions and blind spots. The $\boldsymbol{i}^*$ models do not offer the same richness as some of the existings methods for describing organizations (e.g., narrative text, video clips), and do not aim to replace them. Instead, $\boldsymbol{i}^*$ offers a more concise form of expression more suited to drawing certain pertinent conclusions (e.g., opportunity and vulnerability, how and why), and for easier connection to the information system development process (through a compatible set of underlying knowledge structuring primitives).

(ii) The choice of core concepts. A variety of concepts have been used by organization analysts for analyzing organizations, drawing on diverse disciplines such as psychology, sociology, political science, economics, and others (e.g., as surveyed in [Morgan86][Scott87]). In $\boldsymbol{i}^*$, a minimal set of concepts is chosen to provide the basis for the framework, so that a wide range of perspectives and theories on organizational behaviour can be accommodated. This ontology extends the usual ontology of conceptual modelling (e.g., RML [Greenspan84]) of entity, activity and assertion, primarily by adding the concept of intentional actor. Actors have intentional dependencies on each other. Entities, activities, and assertions are treated intentionally (becoming resources, tasks, and goals), and there are means-ends relationships among them. The semantics for these concepts allows reasoning support to be extended beyond those offered in non-intentional conceptual modelling frameworks.

More specialized organizational modelling concepts which may be specific to certain theoretical perspectives are *modelled* by the modeller. For example, power can be characterized as a softgoal. The semantics of such concepts are captured partly in the rules that are associated with the concept. In the Trillium example, one manifestation of the notion of power is captured in the rule: For a designer to have more power in the project team, one way is to have greater control over the process of how the design gets implemented. It should be noted that the framework allows different actors to have different interpretations of the concept of power.

(iii) The nature of organizational actors. The notion of actor in $\boldsymbol{i}^*$ also aims to be general with respect to various theoretical perspectives on organizational behaviour. The SD model only indicates external relationships among actors, but does not impose any particular built-in constraints on actor behaviour. For example, there is no inherent assumption that actors are rational in a strict sense; and actors can even violate their commitments. A depender therefore needs to use its own judgement as to how much assurance, enforcement, insurance, etc., to counter its own vulnerability.

In the SR model, even though means-ends relationships are explicitly modelled, the actor still is not necessarily rational in a strong sense, because the model allows satisficing, as well as conflicting goals, which are open to judgement. Narrower notions of actor can be obtained by imposing restrictions on the model. One version of rational actor would be one whose decisions strictly conform to the set of rules known to it. Another version would require all alternatives to be exhaustively evaluated to obtain optimal decisions.

For a "cultural" perspective on organizational behaviour, where actors take certain behavioural patterns for granted, means-ends rules are used only in a condition-action mode. Actors do not ask "why". They do not attempt to substitute another means for an end even if alternate means are available. The same set of rules are shared among the entire cultural group of actors. For an a-political perspective, actors inherit the same set of "organizational goals" and do not have private goals (and thus there are no competing interests among actors).[2]

Although the analysis in [Blomberg86] was not explicitly multi-perspective, the issues and concerns that were raised included key concepts that would be emphasized by several major analytical perspectives (e.g. as discussed in [Kling82]). Program planners' desire for better quality, faster development cycles would be dominant issues from an a-political, rational perspective. Designers and programmers were quick to see, from a political perspective, the implications of the new technology on control and distribution of power in the organization. The meaning and perception of the new technology as something incompatible with the proper role of a designer could be interpreted from a symbolic interactionist perspective. Finally, the program planners' use of the prototypes as a resource to reduce project management uncertainty would fit well into a structural contingency perspective.

(iv) The knowledge-based approach. The use of a knowledge-based approach in the framework provides a systematic way to first identify and display the organizational configuration alternatives (the SD models), and then arrange the issues and concerns of actors into a network of arguments (the SR model). Issues from various perspectives interact with each other, through positive and negative contributions, leading to adoption of an alternative when actors felt that their issues and concerns would be adequately addressed by pursuing that alternative.

[2] For overviews and discussions of the various theoretical perspectives on organizational behaviour, see e.g., [Morgan86][Scott87], and in the organizational impact of computing literature, see e.g., [Kling82].

Chapter 7

Application IV - Software Process Modelling[1]

7.1 Introduction

A software process refers to the set of tools, methods, and practices used to produce a software product [Humphrey89]. Historically, software development have largely been product-centred. Recently, many researchers and practitioners have refocused their efforts on the process dimension of software engineering (e.g., [ISPW93] [ICSP93] [ICSE93]). At the core of most of these efforts is some way of *modelling* a software process.

Software process models have been proposed to address a variety of needs, e.g., to improve understanding, to facilitate communication or management, or to support and sometimes automate process enactment [Curtis92]. Most of these models aim to express *what* steps a process consists of, or *how* they are to be performed. However, in order to improve or redesign a process, we often need to have a deeper understanding about the process – an understanding that reveals the "whys" behind the "whats" and the "hows".

Typically, process performers need models that detail the "hows", process managers prefer models that highlight the "whats", while process engineers charged with improving and *redesigning* processes need models that explicitly deal with the "whys" [2]. The need for different types of software process models for different purposes may be compared to the need for different languages to represent software *products* at different levels – requirements (providing the "why"), design (specifying the "what"), and implementation (giving the "how") (e.g., [Jarke92a]).

The need to capture design rationales behind software *products* is well recognized (e.g., [Potts88]). However, to address *process* rationale, we need to face up to the distributed, organizational nature of processes. Because software processes are carried out by many parties or individuals, the "whys" for a process are typically not dictated by some process engineer, but reflect the complex social relationships among process participants. When considering different options for improvement, software engineers, managers and other stakeholders in the organization need to understand how each option would affect their daily work, and their pursuit of project and personal goals. This deeper understanding would help them choose process design options that meet their needs and interests.

[1] This chapter is based in part on [Yu94c].

[2] We follow [Madhavji91] in distinguishing these three classes of users of software process models.

What i^* **offers.** The Strategic Dependency model of the $\boldsymbol{i}^*$ framework provides a richer model of software processes in terms of intentional relationships among software process participants. The Strategic Rationale model describes the reasoning behind a software process, providing a systematic framework for designing a software process to meet project or organization objectives.

- $\boldsymbol{i}^*$ offers a deeper understanding about software processes because it views processes as involving intentional, strategic actors. Participants in a software project depend on each other to accomplish their work. A deeper understanding needs to include implications of these dependencies, so as to understand why the project team goes about developing software in that particular way.
- The descriptive model is intended to capture the software processes as they are in an organization, rather than as prescribed. The intentional modelling avoids casting processes into rigid steps, allowing openness and acknowledging the need for dealing with unanticipated problems as they arise.
- The framework supports the tailoring of software processes to the particular needs of a project or organization. Analysis and design concepts in the framework allow issues to be identified, evaluated, and tradeoffs to be made.
- The framework assists software projects and organizations to identify the types of tools and development environments that can best support their software processes, thus serving as a requirements engineering framework for software development environments (SDEs).

7.2 Modelling Strategic Dependencies in Software Processes

In the *Strategic Dependency* model, we assume that participants in software processes are organizational actors who need to cope with problems cooperatively on an on-going basis. How actors make use of, and constrain, each others' problem solving activity is therefore an important aspect of a software process that needs to be modelled and reasoned about. Actors depend on each other for *goals* to be achieved, *tasks* to be performed, and *resources* to be furnished. By modelling the structure of these *intentional dependencies* among actors, we provide a higher level characterization of a software process.

Figure 7.1 shows a Strategic Dependency model for a hypothetical (and simplistic) software engineering project organization. A customer depends on a project manager to have a system developed. The project manager in turn depends on a designer, a programmer, and a tester to do the technical work and be on schedule. Technical team members depend on each other for intermediate work products such as the design, code, and test results. The manager is also depended on by his boss for avoiding project overrun, and by the quality assurance manager for the system to be maintainable. The user depends on the project manager for a user-friendly and high performance system.

The *goal dependency* relationships between the project manager and his staff means that it is up to members to decide how to do their job. The customer does not care how the system is developed. It is the outcome that matters.

The programmer depends on the tester to test a module via a *task dependency* by specifying a test plan. If the project manager were to indicate the technical steps for each team member to carry out, then the manager would be relating to his staff by task dependencies.

The general manager's dependency on the customer for payment, the tester's dependency on the programmer for code, and the project manager's dependency on his technical staff for notification of task completion, are modelled as *resource dependencies*.

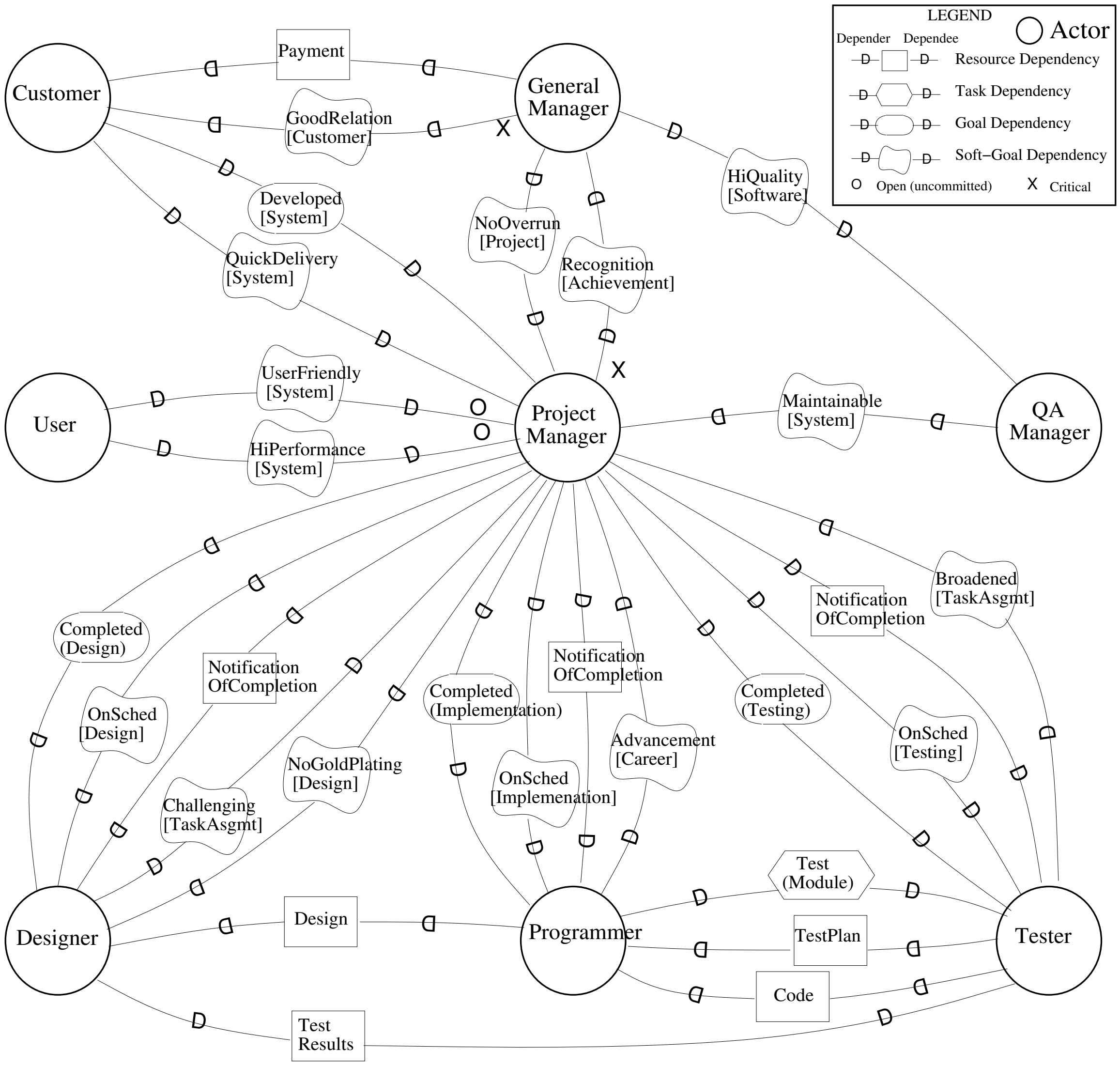

Figure 7.1: A Strategic Dependency model of a simple software project organization

The project manager's dependency on his boss for recognition can be achieved in many different ways. What constitutes sufficient recognition needs to be worked out between the two, and is ultimately decided by the depender. Treating "no project overrun" and "on schedule" as *softgoals* indicates that these are not evaluated as binary yes/no assertions. A (hard) goal dependency would be used if there is a sharp cutoff, e.g., if the product must be delivered either by a promised date, or not delivered at all.

The four types of dependencies differentiate how the depender and dependee relate to each other in terms of their freedom in solving problems and achieving goals.

The three degrees of strength also offers useful distinctions. In Figure 7.1, the general manager's *critical dependency* on having good relations with the customer would lead him to be concerned about whether the project manager can (and will) develop a system and deliver it quickly to the customer, and whether technical team members will also do their part.

By providing these richer characterizations of a software process, the Strategic Dependency model allows an analyst to explore opportunities open to actors by matching wants against abilities, to identify vulnerabilities of actors arising from their dependencies, and to recognize channels by which actors can mitigate their vulnerabilities, such as mechanisms for *enforcing* a commitment, *assuring* its success, and *insuring* against failure. The ability to assess these broader implications help differentiate among alternatives in efforts to design or redesign software processes.

The software process modelling domain offers rich examples for distinguish among roles, positions, and agents. Figure 7.2 shows an example of an Strategic Dependency model of a software

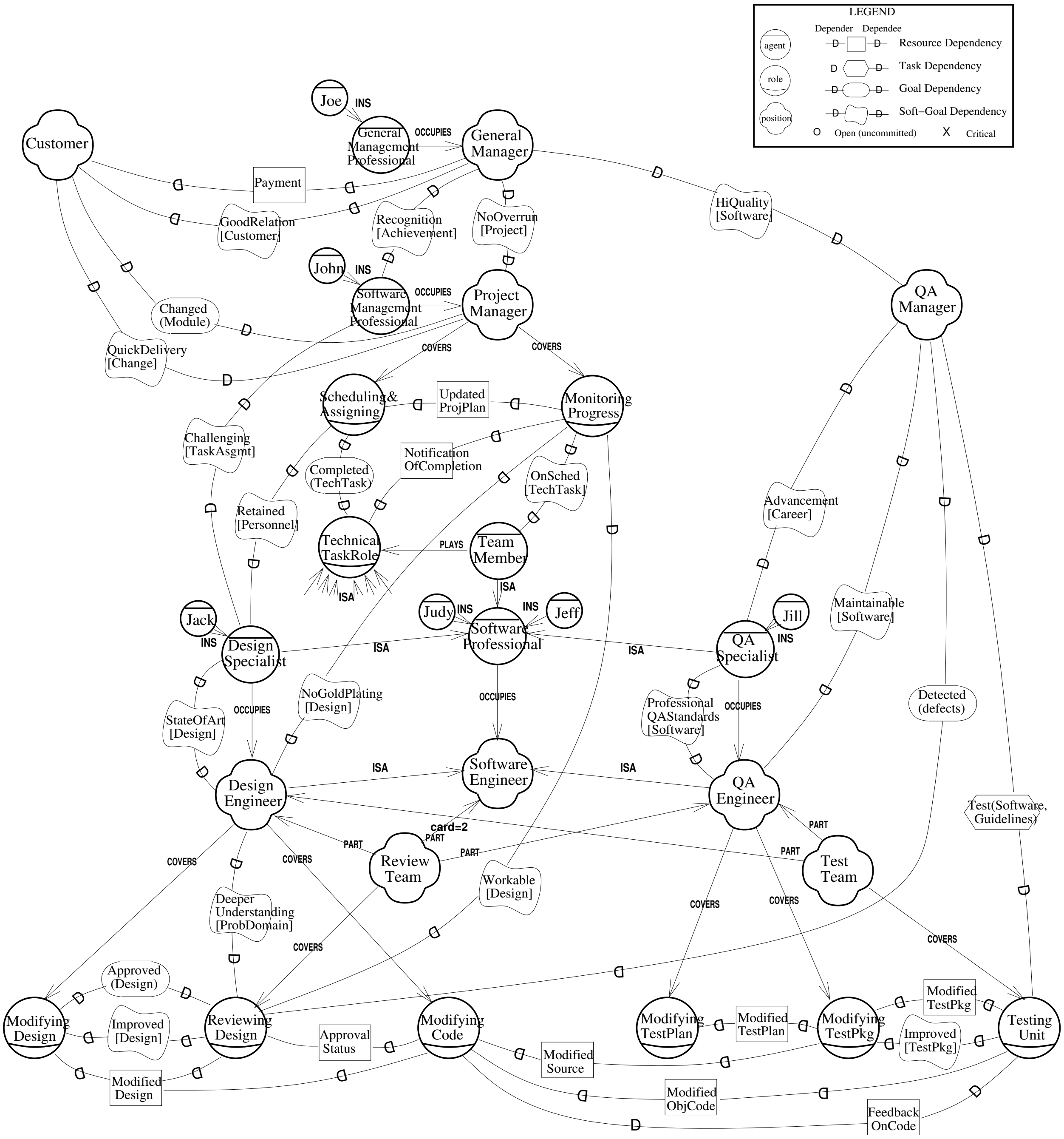

Figure 7.2: A Strategic Dependency model with roles, positions, and agents (adapted from ISPW-6/7 benchmark example)

engineering process organization with agents, roles, and positions. It is an adaptation of the ISPW-6/7 benchmark example [Kellner91]. The organization includes a project manager, design engineers and quality assurance engineers. The example setting includes six technical activities (from "Modifying Design" to "Unit Testing") and two management activities ("Scheduling And Assigning Tasks" and "Monitoring Progress") pertaining to the development and testing efforts required to respond to a change request.

The intentional structure of this organization closely resembles the one in Figure 7.1. Separating out the concepts of roles, positions, and agents gives a finer grouping of dependencies, so that one could identify more precisely how one dependency might lead to other dependencies. For example, the role "Monitoring Project Progress" depends on progress reports from team members, regardless of who is doing the monitoring.

An *agent* is an actor with concrete, physical manifestations, such as a human individual. An agent has dependencies that apply regardless of what roles he/she/it happens to be playing. For example, if John, the project manager desires recognition from his boss, John wants the credit to go towards his personal self, not to the position of project manager (which he hopes to be filled by someone else upon his own promotion), nor to any of the abstract roles that John plays (e.g. "Monitoring Progress"). We use the term *agent* instead of person for generality, so that it can be used to refer to human as well as artificial (hardware/software) agents.

A *position* is intermediate in abstraction between a role and an agent. It is a set of roles typically assigned jointly to one agent. For example, the position of project manager *covers* the two roles of "Scheduling And Assigning Tasks", and "Monitoring Progress". We say that an agent *occupies* a position.

The dependency structure in Figure 7.2 can be understood in terms of three main systems. One set of dependencies can be traced to the customer's goal dependency to have a module changed. This leads to the project manager's dependency to have each portion of the project completed by the respective technical roles (shown at the bottom of the figure), and also to the dependencies among the technical roles. An "IS-A" construct representing conceptual generalization/ specialization is used to simplify the presentation (near centre of figure).

A second system can be traced to the general manager's dependency on the project manager for no project overrun. This leads the project manager to depend on team members to be on schedule, and to notify completion. A third system can be traced to the general manager's dependency on the QA manager for high quality on software produced. This leads the latter's dependencies on the Reviewing and Testing roles. The remaining dependencies can be traced to the need for viability of the dependencies in the main systems.

There can be dependencies from an agent to the position that it occupies. "Design Specialist" is a class of agents, each of whom having a dependency on the position that it occupies – namely "Design Engineer", for achieving the goal that designs produced be state-of-the-art. If the goal is not met, an agent may seek another position.

Roles, positions, and agents can each have subparts. 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.

Process Analysis. A software process model that captures actors' motivations, intents, and rationales provides a better basis for an analyst to explore the broader implications of a process. Because software engineering activities involve uncertainty, actors need to be flexible enough to respond to contingent situations, and be prepared for setbacks. In acknowledging actors' freedoms and constraints, the Strategic Dependency model permits richer types of analysis than conventional, non-intentional models. The formal representation of the SD model allows computational tools to be developed to support analysis. In this section we suggest some types of analyses by considering two important aspects of intentional dependency – the enabling aspect and the vulnerability aspect.

By enlisting the help of dependees, a depender expands opportunities, and can achieve what would otherwise be unachievable. The customer in the example of Figure 7.1 is able to have a system developed, by depending on the project manager, even if the customer has no ability to develop the system himself. The project manager does not have ability to development the system all by himself. He is enabled through dependencies on his technical team. Given an SD model of a software process, one could ask: What new relationships among actors are possible? By matching the open dependencies from dependers and dependees, one can explore opportunities that are open. Classification and generalization hierarchies facilitate the matching of dependums.

The "down-side" of a dependency for a depender is that the depender becomes vulnerable to the failure of the dependency. A depender would be concerned about the viability of a dependency. Various mechanisms can contribute to fortifying a dependency and to mitigate vulnerability. In analyzing an SD model for viability of dependencies, we look for mechanisms such as *enforcement, assurance,* and *insurance.*

A commitment is *enforceable* if there is some way for the depender to cause some goal of the dependee to fail, e.g., if there is a reciprocal dependency. In Figure 7.1, each of the technical team members have dependencies on the project manager. These dependencies make the manager's dependency on team members enforceable. The customer's dependencies on the project manager are not directly enforceable, since there are no reciprocal dependencies. However, the general manager depends on the customer for payment and for good customer relations; and the project manager depends on the general manager for recognition. The customer's dependencies are therefore indirectly enforceable through the general manager. Each leg of indirectness introduces uncertainty and may weaken enforceability. The lack of dependencies from the project manager to the end-user (as opposed to the paying customer), either direct or indirect, would suggest that the user's dependencies on the manager are unenforceable. Figure 7.2 contains more examples of enforcement mechanisms. We note that enforcement loops often go through agents, since it is ultimately agents (especially human agents) who are vulnerable, not abstract roles or positions.

Another way to analyze viability of a dependency is to look for mechanisms for *assuring* commitment. Assurance means that there is some evidence that the dependee will deliver, apart from the dependee's claim. For example, knowing that fulfilling the commitment is in the dependee's own interest would be an assurance. In the example of Figure 7.2, the professional standards and pride of QA specialists provide some assurance to the QA manager that his desire for maintainable software would be met. Unlike in enforcement-based measures, an assurance mechanism does not allow the depender to take action that can cause the dependee to correct its behaviour.

If a conflict of interest is detected, it would contribute negatively to the assurance of a dependency. In Figure 7.2, the project manager depends on the design engineer not to add fancies features beyond the customer's requirements ("No Gold-Plating"). However, design specialists occupying the position of design engineer prefer to do state-of-the-art designs. This is negative assurance that the manager's no gold-plating dependency would be met. An analyst can use the SD model to analyze alignment of interests or conflicts of interests among various combinations of roles, agents, and positions.

Insurance mechanisms reduce the vulnerability of a depender by reducing the degree of dependence on a particular dependee. A depender can improve the chances of a dependum being achieved by having more than one dependee for the same dependum (or parts thereof). Including two software engineers from some other team to do design reviewing provides some *insurance* against failure by the development team to detect their own defects (in addition to addressing the problem of bias). Another type of insurance is the provision of extra resources to enable remedial or recovery action upon failure of the original dependency. Purchasing an insurance policy from an insurer is an example of this type. In contrast to enforcement or assurance, insurance measures can be taken on the depender side without involving the original dependee.

Measures for dealing with vulnerability are often taken in combination. A weekly status report might be used by the general manager to assure no project overrun, and as a basis for deciding whether and when enforcement action is necessary.

By analyzing the opportunities and vulnerabilities of actors, and the provisions that actors make to deal with vulnerabilities, an analyst can gain a fuller understanding of the "whys" behind a software process. Questions such as "Why do we need design reviews?", "Why does the review team have this membership composition?" and "Why does the general manager want weekly status reports?" can be answered more fully. An SD model provides the conceptual framework and the basis for analytical tools.

7.3 Using Strategic Rationales in Software Process (Re-)Design

The Strategic Rationale model of the i^* framework can be used to describe a software process in more detail, in terms of process elements and how they relate to each other via task decomposition and means-ends relationships. New process configurations can be explored and evaluated, in the same manner as described in Chapter 5 in the reengineering of business processes. In this section, we focus on the evaluation of process alternatives with respect to some of the softgoals that might be relevant in designing software processes.

Figure 7.3 shows four alternative arrangements for accomplishing testing in a hypothetical software organization. For simplicity, we omit the (functional) means-ends relationships linking the four alternatives, highlighting the non-functional process design goals (softgoals). The figure shows two major branches, with the left branch consisting of three alternatives. All four alternatives meet the functional goal of `Completed(Testing)` (not shown), but are differentiated with regard to how well they meet non-functional process design goals (shown in the centre of the figure).

The relationship between the designing role and the testing role (and hence between designer and tester) are different in the four alternatives. In the *task dependency* option (left-hand side, top), the designer tells the tester to follow a detailed test plan. This has the advantage of fast turnaround, but the disadvantage that no one other than the designer is really subjecting the software to test (negative contribution to the process design goal that the testing and designing roles should be independent).

In the *goal dependency* alternative (left, middle), the tester is given freedom on how to test, thus achieving some degree of independence. But testing would likely take longer to complete, and it would not be making use of knowledge about the design to focus testing on potential weak spots (negative contribution to `WeightedEffort[Testing, Design]`). A *softgoal dependency* has the advantage of making testing a cooperative venture, fostering team spirit, and contributing to the tester's learning about design.

Looking at the relationship between the project manager and technical team members, we see that the team is rewarded as a unit (on the left-hand branch), fostering a strong team spirit.

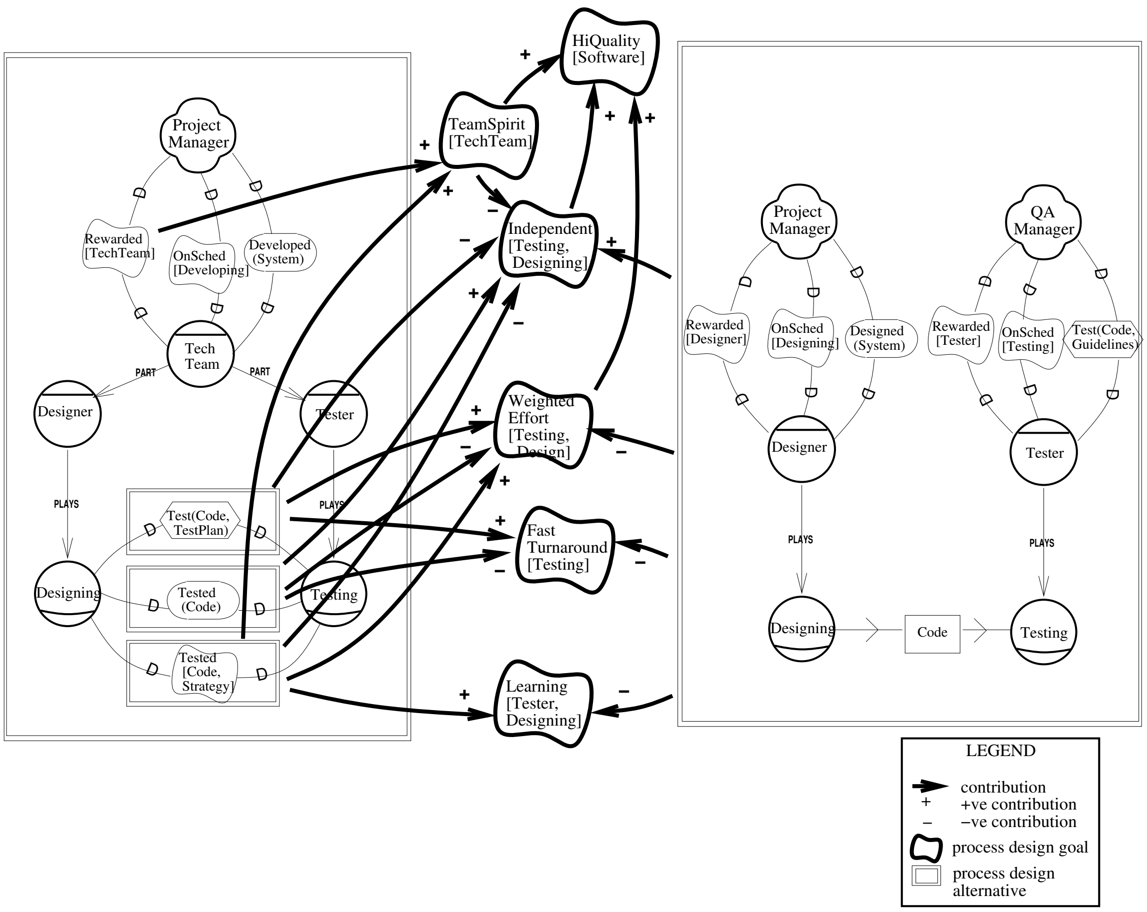

Figure 7.3: Four process design alternatives and their qualitative evaluation with respect to process design goals

This, however, makes all three alternatives on this branch weak with respect to independence of testing.

On the right-hand side, the designer and tester are rewarded separately for their efforts. Each have dependencies from their respective managers, who have no immediate dependencies between them. This alternative is good for achieving independence between testing and designing, but is negative for fast turnaround, design-weighted testing, and for the tester's learning about design.

The non-intentional nature of the flow of `Code` from designer to tester – as opposed to a *resource dependency* – indicates that the tester does not have goals that would be affected if the code is not received. This might be the case if the QA manager does not consider the responsibility of testing to begin until code is received. This is in contrast to the team situation on the left, in which the tester would be motivated to assure prompt completion of design, so that testing could begin on time, in order to assure reward for the whole team, of which the tester is part.

In studying a process and seeking ways to improve it, one could typically come up with many alternatives, possibly involving changes to human procedures as well as selecting among a variety of features in support environments. These alternatives need to be evaluated against many criteria including project objectives (such as quality and productivity) as well as personal concerns (such as reward structures and career paths). The process of designing software processes could be greatly facilitated by providing tools that can help process engineers to systematically pursue design goals by generating and analyzing alternatives, and to make tradeoffs.

7.4 Discussion

The $\boldsymbol{i}^*$ framework differs from most existing software process models in objective and consequently in features. We discuss these differences in this section.

(i) Understanding and redesigning software processes. Most models that have been proposed in the software process modelling research area are non-intentional models that focus on activities and input-output flow. More flexible formalisms include models with rules and triggers, and extensions of Petri nets (e.g., [Deiters90] [Bandinelli93]). These may be viewed as providing the "how", to better support or automate process enactment. The intentional model proposed in this paper focuses on the "why", in order to support reasoning about process improvement and redesign.

By viewing processes as relationships among intentional strategic actors, the $\boldsymbol{i}^*$ models provide deeper insights into existing processes and into potential implications of new proposed processes.

(ii) Software processes are organizational processes too. Software development has been, and still is, an extremely people-intensive process. Much of the processes modelled in "software process" models are carried out in large part by humans, even though they may be supported by automated tools at various points in the process. Most existing process models, however, treat human processes much like machine processes, with little support for dealing with the social, organizational dimension of software development.

The SD model embodies a distributed conception of intentionality, acknowledging that human actors have motivations, beliefs, and abilities. The intentional dimension is represented as relationships between actors, with dependency chains propagating in all directions, criss-crossing the organization in the form of a network. The SD model accommodates uncertainty in organizational environments, and acknowledges actors' flexibility in coping with uncertainty. Intentional dependencies reflect actors' expectations on each others' otherwise unpredictable behaviour. Expectations are not always met, so that *analyses* of enforcement, assurance, and insurance are of interest. The concepts of role, position, and agent reflect how organizations group and manage complex patterns of social relationships.

(iii) Formality, Granularity, and Scriptiveness. In a survey of process modelling, Curtis et al. [Curtis92] have identified formality, granularity, and scriptiveness as important issues for software process modelling research. The Strategic Dependency model is formal without being deterministic. Intentional concepts are used to model actors' expectations about each other's behaviour, and their provisions for unmet expectations. Acknowledging the inherent freedom of actors, analyses on the model focus on issues such as opportunities and risks. An intentional model places limits around an actor's behaviour, in terms of what is expected to be achieved, without explicitly specifying detailed process steps. This avoids the granularity dilemma encountered in non-intentional models, where a coarse-grained description is likely to underspecify by allowing too much freedom, while a fine-grained description tends to overspecify by including process characteristics that are circumstantial rather than essential. The SD model is primarily intended to be used in a descriptive mode. A prescriptive or proscriptive model which only specifies officially sanctioned or prohibited actions would not allow us to reason about the potential impacts of actors' *violations* of expectations and commitments, which we have illustrated in section 4.

(iv) *i serves as requirements engineering framework for software development environments and CASE tools.** The contributions that software development environments and CASE tools and environment make to a project or organization can be more clearly understood using intentional models. Decisions to select, customize, or develop environments and tools can be made systematically in the larger context of software process (re)design, based on explicit process design goals, and taking into account the abilities and strategic interests of various software process participants as stakeholders.

From the perspective of our framework, portions of software process models that are enacted by machine belong *inside* one (or more) of the "agents". The focus of the SD model is on external relationships between agents (human or otherwise). For computer-based agents, the SD model serves as a *requirements* level model. Further constraints are needed to reduce the requirements to a design specification, and from there to an implementation – expressed in a non-intentional representation such as procedural or Petri net formalisms [Jarke92a]. Computer-based agents with planning and problem-solving ability (e.g., [Huff88]), will require less reduction to reach an implementation.

Chapter 8

Conclusions

8.1 Summary of Results

The main result of this research is a framework for modelling processes, and for analyzing and redesigning them. The framework focuses on the type of processes found in organizational settings where people, often supported by computer-based systems, work together towards some end. The framework embodies and formalizes a notion of process that is richer than those presumed in existing process models, by allowing a greater openness in the behaviour and relationships among the agents being modelled, and by viewing them strategically, as social actors with potentially common or conflicting interests and motivations.

The modelling framework consists of two components, the Strategic Dependency model (SD) and the Strategic Rationale (SR) model. Each model has been presented in terms of:

- a set of modelling concepts, including, for the Strategic Dependency model (SD): *actors* (as dependers and dependees), types of dependency, according to the type of the dependum – *goal, task, resource*, and *soft-goal*; strength of dependency – *open, committed*, and *critical*; for the Strategic Rationale model (SR): *task-decomposition links*, *means-ends links and rules*, and *routines*; concepts for process analysis: *ability, workability, viability*, and *believability*; concepts for process redesign: the *raising, addressing, correlating*, and *settling* of issues.

- a semantics for the modelling concepts. These are axiomatically characterized, building on agent modelling work in artificial intelligence (for SD), and on means-ends reasoning and design rationale and support frameworks (for SR).

- representational constructs for embedding these modelling concepts into a conceptual modelling language, Telos.

- an illustration of the use of the models in process analysis and design, using simplified examples from the domain of health care and health care reform.

The utility of the framework was demonstrated in four research areas. Comparisons were made to existing modelling techniques in each of these areas.

- In the Chapter on requirements engineering, the framework was demonstrated to offer a more appropriate set of concepts and techniques than existing frameworks for dealing with "early requirements". A common example domain used among researchers in this area – that of meeting scheduling – was used to facilitate comparison with other approaches to requirements engineering.

- In the Chapter on business process reengineering, the benefits of using a more formal modelling framework and a more systematic approach to reengineering was demonstrated. The need for the more expressive, intentional type of process model (compared to conventional workflow or activity models) was shown. The widely cited example of the reengineering of the goods acquisition process at the Ford Motor Company was used to illustrate the framework.

- In the Chapter on analyzing organizational impacts of computing, it was demonstrated that the rich organizational issues that are usually presented in discursive text can be given a more formal and systematic treatment using the framework. A case study from the literature was re-expressed using the framework.

- In the Chapter on software process modelling, the framework was shown to provide a deeper understanding of software processes beyond existing models in that area, and to support process analysis and design. A benchmark example widely used for comparing software process models served to illustrate the application of the framework to this area.

8.2 Contributions

This work offers a novel perspective on the modelling, analysis, and design of complex organizational processes. The foundation of the perspective lies in the adoption of an intentional, strategic view of actors in organizational settings. Processes are viewed as networks of intentional dependencies among actors. Process reengineering is viewed as the search for new patterns of dependency relationships to advance actors' strategic interests. This perspective is embodied in a set of modelling concepts, which are made precise by axiomatic characterization. Knowledge representation and reasoning techniques are used to provide a systematic, engineering approach to process modelling, analysis and design.

The $\boldsymbol{i}^*$ framework advances the state of the art in process modelling and reengineering in several respects:

- It acknowledges the richer dimension of processes involving humans and machines, and renders these richer issues addressable within a computationally supportable, engineering framework.

- It extends and complements the notion of process as traditionally studied in computer science, by proposing a shift in ontological and epistemological assumptions – towards a more open, strategic perspective (as further detailed under contributions to requirements engineering)

- It extends conceptual modelling (e.g., [Brodie84]) with the ontology of intentional, strategic agents, thus broadening the scope of phenomena that is addressable by formal conceptual modelling, knowledge representation techniques. The original focus on relationships among entities (e.g., in the Entity-Relationships approach to data modelling [Chen76]) is broadened to include strategic relationships among intentional actors [Yu94d].

- It brings together concepts and techniques from several areas within and outside of computer science to produce a new conceptual framework:

  – from theories of organization – concepts of dependency, commitment, means-ends hierarchies, and strategic relationships, to form the basic modelling concepts in $\boldsymbol{i}^*$;

  - from artificial intelligence – agent modelling techniques using intentional operators, adapted to provide agent abstraction in the SD model;
  - from artificial intelligence – concepts and representations from problem-solving and planning, adapted for modelling means-ends reasoning in the SR model;
  - from software engineering – design rationale and design support frameworks, especially a framework for dealing with non-functional requirements, adapted for dealing with soft concepts in dependency relationships and as issues in reengineering

- Finally, the framework suggests process modelling and reengineering as a *class* of problems common to a number of areas – problems involving intentional strategic actors operating in a multi-agent distributed organizational setting, where actors operate with some degree of freedom but under social constraints, where they may try to, but cannot fully predict or control each others' behaviour. Areas in computing to which the framework may be applied include, beyond the four illustrated in this thesis, computer-supported cooperative work, safety-critical systems, and cooperative information system architectures. The framework is also of potential interest to areas outside of computer science, such as policy and strategy analysis and formulation in business and other arenas.

This work also contributes to each of four areas of application:

Requirements engineering (RE)

- A relatively unaddressed area within requirements engineering – called "early requirements" in this work – is identified as deserving attention. Its importance and relation to the rest of requirements engineering and software development are pointed out.

- Existing RE techniques are found to be inadequate or inappropriate for early requirements engineering, primarily due to ontological and epistemological assumptions that are too restrictive:

  - Most requirements models are prescriptive, while a descriptive view is more appropriate for early requirements.
  - Most requirements frameworks strive for completeness and consistency, whereas incompleteness and inconsistencies can be used to advantage in early requirements.
  - Conventional requirements impose obligations on agents, and assume they will comply. In early requirements, one needs to reason about the implications of noncompliance, assuming that agents are ultimately autonomous, unpredictable and uncontrollable.
  - Processes are typically viewed as actions or activity steps in requirements models. In early requirements, detailed process steps need to be left open to acknowledge the inherent freedom (and ability) of agents to deal with the unexpected.
  - Current goal-oriented RE frameworks tend to assume systems and their environments can be designed from scratch and from global goals. More realistic environments often involve multiple stakeholders with competing interests negotiating over the redesign of existing processes.

- Modelling features and analysis and design concepts more suited to early phase requirements engineering are offered in the ***i**** framework. The application of the framework to requirements engineering was illustrated with an common example used in that research community.

Business process reengineering (BPR)

- Goal-directedness is identified as an intuitive underpinning of the concept of process in BPR – as borne out by the themes of "understanding why", "needs of the customer", and "organizing around outcomes" in the BPR literature.
- Despite the underlying goal-directed intuitions, the models commonly used in BPR – activity and workflow models – are non-intentional and do not support goal-oriented reasoning. The i^* framework offers a model for describing processes (SD) that is inherently intentional, and a model for expressing and reasoning about means-ends relationships in processes (SR), thus providing explicit support for some of the central concepts in BPR.
- By embodying the intentional modelling concepts within a knowledge-based representational and reasoning framework, i^* supports a systematic approach for identifying process alternatives, overcoming the shortcomings of current *ad hoc* approaches which rely on rules-of-thumb or anecdotal success stories.
- It is suggested that the widely acknowledged difficulty of the implementation phase in BPR ("change management" in introducing the new process) might well be a symptom of inadequate attention to the key issues of stakeholder interests during the process design phase, which in turn may be indicative of a lack of conceptual frameworks and tools to support understanding and reasoning about these deeper issues. By focusing the modelling and reasoning around intentional actors, the i^* framework brings the strategic dimension of BPR to the fore. Stakeholders and their concerns can be identified systematically. A framework for modelling and reasoning about softgoals – originally developed for dealing with non-functional requirements in software engineering – was adapted for dealing with the "softer" issues when reasoning about strategic interests, allowing for qualitative reasoning, identifying correlated issues, and making tradeoffs.
- The formal representations used in an i^* -supported BPR effort would make business knowledge and rationales readily accessible to software development efforts, thus enabling new business process initiatives to be rapidly and smoothly supported by information system implementations. For information system developers, a systematic understanding about business issues and rationales would provide a deeper basis for software evolution and reuse.

Organizational Impacts Analysis

- Research in the area have primarily relied on informal reasoning, with results presented predominantly in a discursive textual format. The i^* framework offers a knowledge-based approach for organizing the large amounts of knowledge and for assisting in argumentation in organization analysis.
- The framework accommodates formal as well as semi-formal representations, allowing soft concepts such as power, control, esteem, ease of communication, etc., to be treated within a knowledge representation framework. These types of concepts are important for capturing some of the richness in organizational analysis.
- The analysis of organizational impact is usually done after the fact, thus having little impact on system design. By extending the knowledge-based software engineering (KBSE)

paradigm to encompass organization analysis, the $\boldsymbol{i^*}$ framework facilitates the incorporation of organization analysis into the system development cycle as one of its integral parts, providing input to technical design. Insights from the cumulative body of organizational impact research can be used to guide system design systematically.

Software process modelling

- Software process modelling research has tended to focus on models that support the *execution* of processes. Despite the wide-spread interest in "software process improvement", there are few models that offer an ontology more suited to supporting the *understanding and redesign* of software processes. This work demonstrates that an ontology based on intentional, strategic actors can offer a deeper understanding of software processes. The framework opens the way for a systematic, engineering approach to software process modelling, analysis and redesign, supplementing the high-level normative guidelines that guide software process improvement efforts today.

- The intentional models of $\boldsymbol{i^*}$ allow the human, organizational dimension of software processes to be explicitly modelled and reasoned about. The concept of softgoals, for instance, allows issues such as work incentives and team spirit to be included in process models and as process design goals. This provides a bridge between two important sides to software engineering – the technical side and the people, management side.

- The $\boldsymbol{i^*}$ framework also serves as a requirements engineering framework for software development environments and CASE tools. The contributions that these tools and environment make to a project or organization can be more clearly understood using intentional models. Decisions to select, customize, or develop environments and tools can be made systematically in the larger context of software process (re)design, based on explicit process design goals, and taking into account the abilities and strategic interests of various software process participants as stakeholders.

8.3 Future Directions

This work represents a first step at incorporating a notion of intentional, strategic actor into a conceptual modelling framework. A number of further steps need to be taken in order for the full potential of the framework to be realized. At the same time, the framework opens up a number of avenues for research, leading to potential applications both within and outside computer science.

Testing the framework in large, realistic applications

Although the $\boldsymbol{i^*}$ framework has been illustrated with examples from a number of application areas, the practical utility of the framework remains to be tested in full-blooded real-life scenarios. The examples in this thesis were presented as illustrations of the framework concepts, but were not intended to bring out the full complexity of their respective domains. To test the practicality and scalability of the framework, particular organizational situations in several domains would have to be studied in greater detail, including realistic judgements about the degree of completeness (coverage) of the model with respect to the actual phenomena.

An independent research group has reported on some early experiences in using the Strategic Dependency model to characterize and assess a large-scale software maintenance organization

[Briand94]. The model was found to be "very useful in capturing the important properties of the organizational context of the maintenance process, and aided in the understanding of the flaws found in this process." The report also suggested a number of opportunities for extending and improving the SD model.

Further tests are needed to confirm the usefulness of the framework in a broader range of applications, and to gather more data about the strengths and weaknesses of the framework features.

Methodologies for using *i**

Methodologies are needed to guide the usage of the framework in particular contexts. A methodology would specify who should do what and when (including when to stop modelling). It should also indicate how to validate the contents of a model, e.g., by checking whether the various nodes and links indeed reflect the reality, or cross-checking among different actors perceptions of reality. A methodology needs to take into account the skills and background of the types of personnel involved in each activity. For example, a methodology used by in-house process engineers in a software development organization using ***i**** to design software processes may need to be quite different from one used by externally hired consultants assisting management and system professionals in an insurance company to improve their customer service processes. Where ***i**** is used as an early requirements framework, the methodology for its use should be reconciled with the system development methodologies for the rest of the system development process. Roles and processes need to be defined. Indeed, the ***i**** framework could well be used to help develop methodologies for using ***i**** under various contexts.

Linking the framework to systems development

This research has so far focused on the development of the framework itself. For incorporation into the larger context of systems development, the connection between ***i**** and other stages of systems development need to be elaborated (e.g., [Jarke92a]). The specific connections to system design and implementation phase techniques (e.g., [Chung93] [Nixon93,94]) need to be worked out, to form a comprehensive, integrated environment to support system development [Mylopoulos91]. Work is under way to link ***i****, as an early requirements framework, to ALBERT [Dubois94], an agent-oriented requirements specification language. The two types of requirements models complement each other, with ***i**** supporting reasoning about the "whys" in requirements, and ALBERT specifying the "whats".

Tools to support *i**

To benefit from the formal knowledge representation of the framework, appropriate computer-based tools are needed. These could be built on top of existing knowledge base management facilities such as those supporting the Telos language (e.g., [Jarke92b]). Tool support is especially needed for dealing with the large amounts of knowledge typically involved in modelling real-life applications. Knowledge management facilities such as versions and change management, and design support facilities such as design replay would be desirable. For better usability, tools supporting ***i**** will need:

- graphical user interfaces – to automatically display formally represented knowledge (e.g., as stored in the underlying knowledge base management system) in a visual format (e.g., similar to the graphical notations used in this thesis for the SD and SR models). Capability to input, update, and selectively view the models through the graphical representation would further improve usability (e.g., by using or adapting graphical query languages such as GraphLog and visualization systems such as Hy+ [Consens94]).

- efficient algorithms – Certain operations on the model may benefit from the availability of specialized algorithms. A study of the computational complexity of the operations on the model would offer insights into potential tradeoffs between expressiveness and tractability in the modelling features of the framework.

Building up libraries of strategic knowledge

In using a knowledge-based approach, much greater leverage can be gained by reusing existing knowledge. In the case of ***i****, such knowledge would be in terms of classes of actors (agents, roles, and positions) and relationships involving resources, tasks, goals, and softgoals. There are also rules and routines. Generic knowledge may have accumulated as domain expertise. Case-specific knowledge may have been collected over time as the framework is applied to real cases. These knowledge can be brought to bear on subsequent applications of the framework. The accumulation and maintenance of such knowledge in "libraries" is therefore important for the ***i**** framework to achieve its full potential. For example, to help deploy information technology in organizations (as in requirements engineering and business process reengineering), it would be helpful to have libraries of knowledge about:

- classes of organization usage settings, types of actors and processes, with their functional and non-functional requirements (e.g., speed, quality of service),

- classes of information technologies (differentiated types of information systems, groupware, personal productivity software, communication media, etc.,) and their capabilities (functional and non-functional),

- knowhow for bridging the two (means-ends rules and routines, also organized along knowledge structuring hierarchies) about what requirements can be met by what capabilities, and what other issues (and stakeholders) might be cross-impacted.

The development of these knowledge libraries could eventually be done by consultants assisting organizations to reengineer their processes, or technical system vendors and developers aiming to present how their offerings can be used to meet organizational needs. Initially, however, additional research needs to be done to demonstrate the benefits of such libraries. Some related efforts include the Process Handbook project [Malone93], and the Advisor-based Architecture for Enterprise Engineering project [Gruninger94]. The ability to interchange knowledge with these other projects would also be a highly desirable research objective.

One specialized domain area for the application of ***i**** is software development organization and their supporting technologies (software development environments, CASE tools, integrated project support environments, etc.), as discussed in Chapter 7. Convenient starting points for generic knowledge about this domain include the Software Engineering Institutes Capability Maturity Model and ISO-9000 software quality standards (although the knowledge is essentially prescriptive).

Refinements and extensions of the modelling framework

During the development of the framework, a number of issues were recognized as deserving further consideration and exploration, as refinements or extensions to the framework:

1. Elaboration on the agent-role-position relationships, and the interactions with the knowledge structuring dimensions of classification, generalization, and aggregation, e.g.,
   - the inheritance of intentional relationships across classes of actors,
   - intentional relationships between an aggregate actor and its constituents.
2. Further integration of softgoals and their reasoning with the rest of the framework, e.g., the use of the actor abstraction as a scoping mechanism to set boundaries for the propagation of issues and their evaluation.
3. Support for modelling actors' reasoning about other actors' strategic options and decisions. This may require a more elaborate context mechanism in the underlying conceptual modelling framework, drawing on research in multiple viewpoints, e.g., [Nuseibeh93]. The $\boldsymbol{i}^*$ framework does not assume that actors have perfect knowledge about intentions (of other actors or even one's own). A context mechanism would allow multiple hypotheses to be pursued and reasoned about.
4. Additional types of relationships beyond the basic dependencies:
   - other types of dependencies:
     - informational versus non-informational resources,
     - knowhow as an resource, i.e., actors acquiring means-ends knowledge during a process.[1]
   - other types of relationships: It is not clear whether other types of intentional relationships can be expressed in terms of dependencies, or require special treatment, e.g., relationships involving authority and trust. These concepts are important for modelling many types of organizational environments, including those where security is a concern.
5. Actors who learn or acquire knowledge during a process. There are processes in which actors change their knowledge base upon which their decisions are based. A simple version of this could be modelled as knowhow dependency.
6. Elaboration on the time dimension, including:
   - creation and termination of actors, e.g.,
     - of role instances, as in the handling of a customer file,
     - of agent instances, as in hiring and firing,
     - of position classes, as in managerial processes which create new kinds of positions (essentially a meta-level process).
   - addition of a planning component to the framework. The current framework provides representations for configurations of strategic relationships, but not the transitional steps for moving towards them from an existing configuration. Very often, decisions regarding what new configuration to adopt involve reasoning about how to get to the new configuration from the existing one (e.g., the relative difficulty in transitioning to alternative new configurations). The ability to represent and reason about strategic plans would significantly augment the power of the framework.

A number of extensions and improvements have also been proposed by Briand et al. in [Briand94].

[1] [Clement90] described how a group of office workers depended on each other for knowledge about how to use word processing and other desktop computing facilities in order to carry out their daily work.

Broader applications

The framework can potentially be applied to a number of other areas, beyond the four illustrated in the application chapters in this thesis. Within the computing disciplines itself, the framework may be applied to:

- the area of safety-critical systems, where vulnerability and methods of mitigation of vulnerability would be of interest,
- the area of secure systems, where the motivations and behaviours of strategic (often antagonistic) actors are of interest,
- the architectural design of networks of interacting independently-purposive software systems, (e.g., cooperative information systems [CoopIS94]). As software systems are increasingly networked into heterogeneous environments, the other systems that they interact with are often developed by different organizations, at different times ("legacy systems"), and to serve diverse, often competing interests. For example, in the increasingly networked systems serving a bank, there would be systems representing the many departments and divisions in a bank interacting with each other and with systems representing clients and other financial institutions. On public commercial information networks ("information highways"), there would be software agents representing service providers (e.g., video-on-demand, shop-at-home), network carriers and administrators (e.g., switching, billing), information brokers and other intermediaries, and end-users. An intentional, strategic view of the relationships among these interacting software agents would offer a richer analytical and design framework than existing architectural frameworks that do not make intentional relationships explicit. Similarly, an organizational perspective on interacting systems could be of interest to distributed AI (e.g., as argued in [Fox81] [Hewitt84] and [Gasser91].)

By providing modelling and reasoning features to deal with strategic relationships among actors, the framework opens up the possibility of using a conceptual modelling, knowledge-based approach to provide computational support to broader application areas, including, e.g.:

- business strategy – The analysis and formulation of business strategic relationships can be assisted by the systematic, knowledge-based approach of $\boldsymbol{i}^*$. The types of actors of interest are customers, suppliers, and competitors. The applicability of the framework to actors at this aggregate level needs to be investigated.
- policy analysis – Similarly, the analysis and formulation of public policy, such as health care reform, or environmental policy also involve searching for alternative patterns of relationships among intentional strategic actors, exploration implications, and seeking solutions that adequately address the concerns of multiple stakeholders. The extension of the $\boldsymbol{i}^*$ framework to these broader application domains would require considerably more research.

Bibliography

[Attewell84] P. Attewell and J. Rule, Computing and Organizations: What We Know and What We Don't Know, *Communications of the ACM*, 27(12), Dec. 1984, pp. 1184-1192.

[Bandinelli93] S. Bandinelli and A. Fuggetta, Computational Reflection in Software Process Modelling: the SLANG Approach, *Proc. 15th International Conference on Software Engineering*, 1993, pp. 144-154.

[Barber82] G. R. Barber, *Office Semantics*, Ph.D. Dissertation, Dept. of Elec. Eng. and Comp. Sci., M.I.T., 1982.

[Becker60] H. S. Becker, Notes on the Concept of Commitment, *American Journal of Sociology*, vol 66, July 1960, pp. 32-40.

[Blackwell93] G. Blackwell, Re-engineering the Claims Process – Case Study: Alberta's Workers' Compensation Board, *IT Magazine*, Jan. 1993, pp. 14-18.

[Blomberg86] J. L. Blomberg, The Variable Impact of Computer Technologies on the Organization of Work Activities, *Proceedings of the Conference on Computer-Supported Cooperative Work*, pp. 35-42, 1986, also in *Computer-Supported Cooperative Work – A Book of Readings*, I. Greif, ed., pp. 771-781.

[Boehm81] B. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

[Borgida85] A. Borgida, S. Greenspan, J. Mylopoulos, Knowledge Representation as the Basis for Requirements Specifications, *IEEE Computer*, April 1985, pp. 82-91.

[Bracchi84] G. Bracchi and B. Pernici, The Design Requirements of Office Systems, *ACM Trans. Office Information Systems,* 2(2) April 1984, pp. 151-170.

[Briand94] L. Briand, W. L. Melo, C. Seaman, and V. Basili, *Characterizing and Assessing a Large-Scale Software Maintenance Organization,* Technical Report CS-TR-3354, Computer Science Department, University of Maryland, 1994.

[Brodie84] M. L. Brodie, J. Mylopoulos, J. W. Schmidt, eds., *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, Springer-Verlag, 1984.

[Bubenko80] J. A. Bubenko, Information Modeling in the Context of System Development, *Proc. IFIP,* pp. 395-411, 1980.

[Bubenko91] J. A. Bubenko, Next Generation Information Systems: an Organizational Perspective *Intl. Workshop on Development of Intelligent Information Systems*, Niagara-on-the-Lake, Ontario, Canada, April 21-23, 1991, pp. 22-31.

[Bubenko93] J. A. Bubenko, Extending the Scope of Information Modeling, *Proc. 4th Int. Workshop on the Deductive Approach to Information Systems and Databases,* Lloret-Costa Brava, Catalonia, Sept. 20-22, 1993, pp. 73-98.

[Castelfranchi92] C. Castelfranchi, M. Miceli, and A. Cesta, Dependence Relations Among Autonomous Agents, *Decentralized A.I. - 3 (Proc. 3rd European Workshop on Modeling Autonomous Agents in a Multi-Agent World)*, Elsevier, 1992.

[Chen76] P. P. Chen, The Entity-Relationship Model – Toward a Unified View of Data, *ACM Trans. Database Sys.*, vol. 1, no. 1, pp. 9-38, 1976.

[Chung91] L. Chung, Representation and Utilization of Non-Functional Requirements for Information System Design. *Advanced Information Systems Engineering, Proc. 3rd Intl. Conf. CAiSE, Trondheim, Norway*, R. Anderson etal. (eds.), Springer-Verlag, pp. 5-30, 1991.

[Chung93] K. L. Chung, *Representing and Using Non-Functional Requirements for Information System Development: A Process-Oriented Approach*, Ph.D. Thesis, also Tech. Rpt. DKBS-TR-93-1, Dept. of Comp. Sci., Univ. of Toronto, June 1993.

[Chung94a] K. L. Chung, B. A. Nixon, E. Yu, Using Quality Requirements to Drive Software Development, *ICSE-16 Workshop on Research Issues in the Intersection Between Software Engineering and Artificial Intelligence*, International Conference on Software Engineering, Sorrento, Italy, May 16-20, 1994.

[Chung94b] K. L. Chung, B. A. Nixon, E. Yu, Using Quality Requirements to Systematically Develop Quality Software, *Fourth International Conference on Software Quality: Capitalizing on Software Quality,* October 3–5, 1994.

[Clement87] A. Clement, C. C. Gotlieb, Evolution of an Organization Interface: The New Business Department at a Large Insurance Firm, *Trans. Office Information Systems*, pp. 328–339, 1987.

[Clement90] A. Clement and D. Parsons, Work Group Knowledge Requirements for Desktop Computing, *Proc. 23rd Hawaii International Conference on System Sciences*, Kailua-Kona, Hawaii, pp. 84–93, Jan. 1990.

[Clement94] A. Clement, Considering Privacy in the Development of Multi-Media Communications, *Computer Supported Cooperative Work (CSCW)* 2, 1994, Kluwer Academic Publishers, pp. 67-88.

[Coad90] P. Coad and E. Yourdon, *Object-Oriented Analysis*, Yourdon Press, Prentice-Hall, 1990.

[Cohen90] P. R. Cohen and H. J. Levesque, Intention is Choice with Commitment, *Artificial Intelligence,* 42 (3), 1990.

[Conklin88] J. Conklin and M. L. Begeman, gIBIS: A Hypertext Tool for Explanatory Policy Discussions, *ACM Transactions on Office Information Systems*, 6(4), pp. 303-331, 1988.

[Conklin91] E. J. Conklin and K. C. Burgess-Yakemovic, A process-oriented approach to design rationale, *Human-Computer Interaction*, 6(3-4), 1991.

[Consens94] M. P. Consens, F. Ch. Eigler, M. Z. Hasan, A. O. Mendelzon, E. G. Noik, A. G. Ryman, and D. Vista, Architecture and applications of the Hy+ visualization system, *IBM Systems Journal*, 33(3), pp. 458–476, 1994,

[CoopIS94] Proceedings of the Second International Conference on Cooperative Information Systems (CoopIS-94), M. Brodie, M. Jarke, and M. Papazoglou, eds., Toronto, Canada, May 17-20, 1994.

[Croft88] W. B. Croft and L. S. Lefkowitz, A Goal-Based Representation of Office Work, *Office Knowledge: Representation, Management, and Utilization,* W. Lamersdorf (ed.), Elsevier, 1988, pp. 99-124.

[Crosby79] P. R. Crosby, *Quality is Free*, MacGraw-Hill, 1979.

[Crozier64] M. Crozier, *The Bureaucratic Phenomenon*, London: Tavistock, 1964.

[Curtis88] B. Curtis, H. Krasner and N. Iscoe, A Field Study of the Software Design Process for Large Systems, *Communications of the ACM*, 31(11), pp. 1268-1287, 1988.

[Curtis92] W. Curtis, M. I. Kellner and J. Over, Process Modelling, *Comm. ACM*, 35 (9), 1992, pp. 75-90.

[Dardenne93] A. Dardenne, A. van Lamsweerde and S. Fickas, Goal-Directed Requirements Acquisition, *Science of Computer Programming*, 20, pp. 3-50, 1993.

[Davenport90] T. Davenport, J. Short, The New Industrial Engineering: Information Technology and Business Process Redesign, *Sloan Management Review*, Summer 1990, pp. 11-28.

[Davenport93] T. H. Davenport, *Process Innovation: Reengineering Work Through Information Technology*, Harvard Business School Press, Boston, Mass., 1993.

[Davenport94] T. H. Davenport, Saving IT's Soul: Human-Centered Information Management, *Harvard Business Review*, pp. 119–131, March-April 1994.

[Deiters90] W. Deiters and V. Gruhn, Managing Software Processes in the Environment MELMAC, *Proc. 4th International Symposium on Practical Software Development Environments*, Irvine, 1990, SIGSOFT Notes 15(6), pp. 193-205.

[deJong89] P. de Jong, *Ubik: A Framework for the Development of Distributed Organizations*, Ph.D. Dissertation, Dept. of Elec. Eng. and Comp. Sci., M.I.T., 1989.

[DeMarco78] T. DeMarco, *Structured Analysis and System Specification*, Yourdon Press: N.Y.

[Dubois86] E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaert and A. Rifaut, A Knowledge Representation Language for Requirements Engineering, *Proc. IEEE*, 74 (10), pp. 1431 –1444, Oct. 1986.

[Dubois89] E. Dubois, A Logic of Action for Supporting Goal-Oriented Elaborations of Requirements, *Proc. 5th International Workshop on Software Specification and Design*, Pittsburgh, PA, 1989, pp. 160-168.

[Dubois92] E. Dubois, Ph. Du Bois and A. Rifaut, Elaborating, Structuring and Expressing Formal Requirements of Composite Systems, *Proc. Fourth Conf. Advanced Info. Sys. Eng.*, Manchester, U.K., May 12-15, 1992.

[Dubois94] E. Dubois, Ph. Du Bois, F. Dubru, and M. Petit, Agent-Oriented Requirements Engineering – A Case Study Using the ALBERT Language, *Proc. 4th International Working Conference on Dynamic Modelling and Information Systems – DYNMOD'94*, Noordwijkerhout, the Netherlands, Sept. 1994.

[Emerson62] R. M. Emerson, Power-Dependence Relations, *American Sociological Review*, v. 27, February 1962, pp. 31-40.

[Feather87] M. S. Feather, Language Support for the Specification and Development of Composite Systems, *ACM Trans. Prog. Lang. and Sys.* 9, 2, April 1987, pp. 198-234.

[Feather94] M. S. Feather, Composite System Design, *ICSE-16 Workshop on Research Issues in the Intersection Between Software Engineering and Artificial Intelligence*, International Conference on Software Engineering, Sorrento, Italy, May 16-20, 1994.

[Fickas92] S. Fickas and R. Helm, Knowledge Representation and Reasoning in the Design of Composite Systems, *IEEE Trans. Soft. Eng.*, 18, 6, June 1992, pp. 470-482.

[Fikes80] R. E. Fikes and D. A. Henderson, On Supporting the Use of Procedures in Office Work, *Proceedings of American Association for Artificial Intelligence*, pp. 202-207, 1980.

[Finkelstein87] A. Finkelstein and C. Potts, Building Formal Specifications Using "Structured Common Sense", *Proc. 4th International Workshop on Software Specification and Design*, Monterey CA, April 1987, pp. 108-119.

[Fox81] M. S. Fox, An Organizational View of Distributed Systems, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1), 1981, pp. 70-80.

[Gasser86] L. Gasser, The Integration of Computing and Routine Work, *Trans. Office Info. Sys.*, vol. 4, no. 3, July 1986, pp. 205-225.

[Gasser91] L. Gasser, Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics, *Artif. Intell.,* 47, Jan. 1991, pp. 107-138.

[Gerson76] E. M. Gerson, On "Quality of Life", *American Sociological Review*, 41, Oct. 1976, pp. 793-806.

[Greenspan84] S. J. Greenspan, *Requirements Modelling: A Knowledge Representation Approach to Software Requirements Definition,* Ph. D. Thesis, Dept. of Comp. Sci., Univ. of Toronto, 1984.

[Greenspan94] S. J. Greenspan, J. Mylopoulos, A. Borgida, On Formal Requirements Modeling Languages: RML Revisited, (invited plenary talk), *Proc. 16th Int. Conf. Software Engineering*, May 16-21 1994, Sorrento, Italy, pp. 135-147.

[Grudin88] J. Grudin, Why CSCW Applications Fail: Problems in the Design and Evaln of Organizational Interfaces, *Proc. Conference on Computer-Supported Cooperative Work* 1988, pp. 85-93.

[Gruninger94] M. Gruninger and M. S. Fox, An Advisor-based Architecture for Enterprise Engineering, *AAAI-94 Workshop on Artificial Intelligence in Business Process Reengineering*, Seattle, Washington, August 1994, Working Notes, pp. 1-8.

[**Hammer90**] M. Hammer, Reengineering Work: Don't Automate, Obliterate, *Harvard Business Review*, July-August 1990, pp. 104-112.

[**Hammer93**] M. Hammer and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*, HarperBusiness, 1993.

[**Hauser88**] J. R. Hauser, D. Clausing, The House of Quality, *Harvard Business Review*, May-June 1988, pp. 63-73.

[**Henderson86**] D. A. Henderson, Trillium: A Knowledge-based Design Environment for Control/Display Interfaces, *Proc. CHI '86 Human Factors in Computing Systems*, M. Mantei and P. Orbeton, eds., April 1986, pp. 221-227.

[**Hewitt84**] C. Hewitt and P. de Jong, Open Systems, *On Conceptual Modelling*, M. Brodie, J. Mylopoulos and J. Schmidt, eds., Springer-Verlag, 1984, pp. 147-164.

[**Hewitt86**] C. Hewitt, Offices Are Open Systems, *Trans. Office Information Systems*, vol. 4, no. 3, July 1986, pp. 271-287.

[**Hickson71**] D. J. Hickson, C. R. Hinings, C. A. Lee, R. E. Schneck, and J. M. Pennings, A Strategic Contingencies Theory of Intra-Organizational Power, *Administrative Science Quarterly*, 16, 1971, pp 216-229.

[**Huff88**] K. E. Huff and V. R. Lesser, A plan-based intelligent assistant that supports the software development process, *Proc. Third Software Engineering Symp. on Practical Software Development Environments. Software Eng. Not. 13*, 5, 1989, pp. 97-106.

[**Humphrey89**] W. Humphrey, *Managing the Software Process*, Addison-Wesley, Reading, Mas., 1989.

[**ICRE94**] *Proc. First International Conference on Requirements Engineering*, Colorado Springs, 1994.

[**ICSP93**] *Proc. 2nd International Conference on the Software Process*, Berlin, Germnay, Feb. 1993.

[**ICSE93**] *Proc. 15th International Conference on Software Engineering*, Baltimore, May 1993.

[**ISPW93**] *Proc. 8th International Software Process Workshop*, 1993.

[**ISRE93**] *Proc. First International Symposium on Requirements Engineering*, San Diego, 1993.

[**Jackson83**] M. Jackson, *System Development*, Prentice-Hall, 1983.

[**James93**] P. N. James, Interview – Cyrus F. Gibson on IS Reengineering, *Information Systems Management*, Winter 1993, pp. 83-87.

[**Jarke92a**] M. Jarke, J. Mylopoulos, J. W. Schmidt, Y. Vassiliou, DAIDA: An Environment for Evolving Information Systems, *ACM Trans. Information Systems*, vol. 10, no. 1, Jan 1992, pp. 1-50.

[**Jarke92b**] M. Jarke, ed., *ConceptBase V3.1 User Manual*, University of Passau, 1992.

[Jarke94] M. Jarke and K. Pohl, Requirements Engineering in the Year 2001: On (Virtually) Managing a Changing Reality, *Workshop on System Requirements: Analysis, Management, and Exploitation*, Schloß Dagstuhl, Saarland, Germany, October 4–7, 1994.

[Kaiser90] G. E. Kaiser, N.S. Barghouti, and M. H. Sokolsky, Preliminary Experience with Process Modeling in the Marvel Software Development Environment Kernel, *Proc. 23rd Annual Hawaii International Conference on System Sciences, Vol. II – Software Track*, IEEE Computer Society, Washington DC, 1990, pp 131-140.

[Karlgaard94] R. Karlgaard, ASAP Interview – Mike Hammer, *Forbes ASAP Magazine*, 1994, pp. 69-75.

[Keen81] P. Keen, Information Systems and Organizational Change, *Communications of the ACM*, vol. 24, no. 1, January 1981, pp. 24-33.

[Keen91] P. Keen, *Shaping the Future: Business Design Through Information Technology*, Harvard Business School Press, Boston, Mass., 1991.

[Kellner91] M. Kellner, P. Feiler, A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo, and H. Rombach, Software Process Modeling Example Problem, from *Seventh Intentional Software Process Workshop*, Yountville, California, Oct. 1991.

[Kling82] R. Kling, W. Scacchi, The Web of Computing: Computer Technology as Social Organization *Advances in Computing,* vol. 21, 1982, pp. 1-90.

[Kling87] R. Kling, Defining the Boundaries of Computing Across Complex Organizations, *Critical Issues in Information Systems Research*, R. J. Boland Jr., and R. A. Hirschheim, eds., Wiley, 1987.

[Kling91] R. Kling, Cooperation, Coordination and Control in Computer-Supported Work *Communications of the ACM,* vol. 34, no. 12, December 1991, pp. 83-88.

[Kling93] R. Kling, Organizational Analysis in Computer Science, *The Information Society,* vol. 9, no. 2, 1993.

[Lee90] J. Lee, SIBYL: A Qualitative Decision Management System, *Artificial Intelligence at MIT: Expanding Frontiers*, P. Winston, ed., with S. Shellard, Cambridge, MA: MIT Press, June 1990.

[Lee92] J. Lee, *A Decision Rationale Management System: Capturing, Reusing, and Managing the Reasons for Decisions*, Ph.D. thesis, MIT, 1992.

[Lee93] J. Lee, Goal-Based Process Analysis: A Method for Systematic Process Redesign, *Conference on Organizational Computing Systems*, Milpitas, California, Nov. 1-4, 1993, pp. 196-201.

[Lesperance91] Y. Lesperance, *A Formal Theory of Indexical Knowledge and Action,* Ph.D. Thesis, Dept. of Comp. Sci., Univ. of Toronto, also, Tech. Rept. CSRI-248, Comp. Sys. Res. Inst., Univ. of Toronto, Feb. 1991.

[Lochovsky83] F. H. Lochovsky, A Knowledge-Based Approach to Supporting Office Work, *Database Engineering*, IEEE Computer Society, 16(3), pp. 43-51, Sept. 1983.

[**Loomis94**] C. Loomis, The Real Action in Health Care, *Fortune Magazine*, July 11, 1994, pp. 149-157.

[**Lucas81**] H. C. Lucas, *Implementation: the Key to Successful Information Systems*, New York: Columbia University Press, 1981.

[**Lyytinen87**] K. Lyytinen, Different Perspectives on Information Systems: Problems and Solutions, *ACM Computing Surveys*, 19(1), March 1987, pp. 5-46.

[**MacLean91**] A. MacLean, R. Young, V. Bellotti, T. Moran, Questions, Options, and Criteria: Elements of Design Space Analysis, *Human-Computer Interaction*, vol. 6, 1991, pp. 201-250.

[**Madhavji91**] N. Madhavji, The Process Cycle, *IEE Software Engineering Journal*, Special Issue on Software Process and Its Support, N. Madhavji and W. Schaefer, eds., 6(5) Sept. 1991, pp. 234-242.

[**Malone87**] T. W. Malone, Modeling Coordination in Organizations and Markets, *Management Science*, vol. 33, 1987, pp. 1317-1332.

[**Malone93**] T. W. Malone, K. Crowston, J. Lee, B. Pentland, Tools for Inventing Organizations: Toward a Handbook of Organizational Processes, *Proc. 2nd Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, IEEE Computer Society Press, 1993, pp. 72-82.

[**Malone90**] T. W. Malone, K. Crowston, What is Coordination Theory and How Can It Help Design Cooperative Work Systems? *CSCW 90 Proc.* , 1990, pp. 357-370.

[**March58**] J. G. March and H. A. Simon, *Organizations*, Wiley, 1958.

[**Markus83**] L. Markus, Power, Politics, and MIS Implementation, *Communications of the ACM*, vol. 26, no. 6, June 1983, pp. 430-444.

[**Medina-Mora92**] R. Medina-Mora, T. Winograd, R. Flores, and F. Flores, The Action Workflow Approach to Workflow Management Technology, *Conference on Computer Supported Cooperative Work*, Nov. 1992, pp.281-288

[**Mi93**] P. Mi and W. Scacchi, Articulation: An Integrated Approach to the Diagnosis, Replanning, and Rescheduling of Software Process Failures, *Conference on Knowledge-Based Software Engineering*, 1993.

[**Morgan86**] G. Morgan, *Images of Organization*, Sage Publications, 1986.

[**Mylopoulos90**] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis, Telos: Representing Knowledge about Information Systems, *ACM Trans. Info. Sys.*, 8 (4), pp. 325-362, October 1990.

[**Mylopoulos91**] J. Mylopoulos, Representing Knowledge About Information Systems, *International Workshop on Development of Intelligent Information Systems*, Niagara-on-the-Lake, Ontario, Canada, April 21-23, 1991, pp. 94-96.

[**Mylopoulos92**] J. Mylopoulos, L. Chung, B. Nixon, Representing and Using Non-Functional Requirements: A Process-Oriented Approach, *IEEE Trans. Soft. Eng.*, 18 (6), June 1992.

[Neumann94] P. G. Neumann, Computer Risks Digests, *ACM SIGSOFT Software Engineering Notes*, also, regular feature column in *Communications of the ACM.*

[Nilsson80] N. Nilsson, *Principles of Artificial Intelligence*, Tioga Press, 1980.

[Nixon93] B. Nixon, Dealing with Performance Requirements During the Development of Information Systems, *Proceedings of First IEEE Symposium on Requirements Engineering*, San Diego, Calif., 1993, pp. 42-49.

[Nixon94] Brian A. Nixon, Representing and Using Performance Requirements During the Development of Information Systems. *Advances in Database Technology — EDBT '94,* Matthias Jarke, Janis Bubenko, Keith Jeffery (Eds.), Proc. 4th Int. Conf. Extending Database Technology, Cambridge, United Kingdom, March 1994. Springer-Verlag, 1994, pp. 187–200.

[Nuseibeh93] B. Nuseibeh, J. Kramer, A. Finkelstein, Expressing the Relationships Between Multiple Views in Requirements Specification, *15th Int. Conf. Soft. Eng.*, Baltimore, 1993, pp.187-196.

[Orlikowski92] W. J. Orlikowski, Learning from Notes: Organizational Issues in Groupware Implementation, *Proceedings of ACM Conference on Computer-Supported Cooperative Work*, Toronto, Canada, October 31 – November 4, 1992, pp. 362–369.

[Pfeffer78] J. Pfeffer, G. Salancik, *The External Control of Organizations: A Resource Dependence Perspective*, Harper and Row, 1978.

[Potts88] C. Potts, G. Bruns, Recording the Reasons for Design Decisions, *Proc. 10th International Conference on Software Engineering* , 1988, pp. 418-427.

[Rockart91] J. F. Rockart and J. E. Short, The Networked Organization and the Management of Interdependence, *The Corporation of the 1990's – Information Technology and Organizational Transformation*, M. Scott Morton, ed., 1991.

[Ross77] D. T. Ross and K. E. Shoman, Structured Analysis for Requirements Definition, *IEEE Trans. Soft. Eng.*, Vol. SE-3, No. 1, Jan. 1977.

[Sathi85] A. Sathi, M. S. Fox, and M. Greenberg, Representation of Activity Knowledge for Project Management, *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-7(5), Sept. 1985.

[Scott87] W. R. Scott, *Organizations: Rational, Natural, and Open Systems*, 2nd ed., Prentice Hall, 1987.

[Simon81] H. A. Simon, *The Sciences of the Artificial*, 2nd ed., MIT Press, 1981.

[Scheer89] A.-W. Scheer, *Enterprise-Wide Data Modelling: Information Systems in Industry*, Springer-Verlag, 1989.

[Stewart92] T. A. Stewart, The Search for the Organization of Tomorrow, *Fortune Magazine*, May 18, 1992, pp. 93-98.

[Suchman83] L. Suchman, Office Procedures as Practical Action: Models of Work and System Design, *ACM Trans. Office Information Systems*, vol. 1, no. 4, October 1983, pp. 320-328.

[Suchman84] L. Suchman and E. Wynn, Procedures and Problems in the Office, *Office: Technology and People*, 2, Elsevier Science Publishers B.V., Amsterdam, 1984, pp. 133-154.

[Suchman87] L. Suchman, *Plans and Situated Actions*, Cambridge University Press, 1987.

[Theodoulidis92] C. Theodoulidis, B, Wangler, P. Loucopoulos, The Entity-Relationship-Time Model, *Conceptual Modelling, Databases, and CASE*, P. Loucopoulos, R. Zicari, eds., Wiley, 1992, pp. 81-116.

[Thomas91] B. Thomas, Y. Shoham, A. Schwartz, and S. Kraus, Preliminary Thoughts on an Agent Description Language, *Int. J. Intell. Sys.*, Vol. 6, 1991, pp. 498-508.

[Thompson67] J. D. Thompson, *Organizations in Action,* McGraw-Hill, 1967.

[Tueni88] M. Tueni, J. Li, and P. Fares, AMS: A Knowledge-Based Approach to Task Representation, Organization and Coordination, *Conference on Office Information Systems*, 1988.

[VanLamsweerde93] A. Van Lamsweerde, R. Darimont, Ph. Massonet, The Meeting Scheduler Problem: Preliminary Definition. Copies may be obtained from Prof. Van Lamsweerde, Universite Catholique de Louvain, Unite d'Informatique, Place Sainte-Barbe, 2, B-1348 Louvain-la-Neuve, Belgium. (avl@info.ucl.ac.be)

[Venkatraman91] N. Venkatraman, IT-Induced Business Reconfiguration, *The Corporation of the 1990's – Information Technology and Organizational Transformation*, M. Scott Morton, ed., 1991, pp. 122-158.

[Winograd87] T. Winograd, F. Flores, *Understanding Computers and Cognition*, Reading, MA: Addison-Wesley, 1987.

[Woo88] C. Woo, *An Object-Oriented Model for Supporting Office Work*, Ph.D. Thesis, Dept. of Comp. Sci., Univ. of Toronto, 1988.

[Yu93a] E. Yu, Modelling Organizations for Information Systems Requirements Engineering, *Proceedings of First IEEE Symposium on Requirements Engineering*, San Diego, Calif., January 1993, pp. 34-41.

[Yu93b] E. Yu, An Organization Modelling Framework for Multi-Perspective Information System Design, *Requirements Engineering 1993 – Selected Papers*, J. Mylopoulos et al., eds., Tech. Rpt. DKBS-TR-93-2, Dept. Comp. Sci., Univ. of Toronto, July 1993, pp. 66-86.

[Yu93c] E. Yu, J. Mylopoulos, An Actor Dependency Model of Organizational Work – With Application to Business Process Reengineering, *Proc. Conf. Organizational Computing Systems (COOCS 93)*, Milpitas, Calif., Nov. 1-4, 1993, pp. 258-268.

[Yu93d] E. Yu, An Organization Modelling Framework for Information Systems Requirements Engineering, *Proc. 3rd Workshop on Info. Tech. and Systems, (WITS'93)*, Orlando, Florida, USA, December 4-5 1993, pp. 172-179.

[Yu94a] E. Yu, J. Mylopoulos, Using Goals, Rules, and Methods To Support Reasoning in Business Process Reengineering, *Proc. 27th Hawaii Int. Conf. System Sciences*, Maui, Hawaii, Jan. 4-7, 1994, vol. IV, pp. 234-243.

[Yu94b] E. Yu, J. Mylopoulos, Understanding "Why" in Software Process Modelling, Analysis, and Design, *Proc. 16th Int. Conf. Software Engineering*, May 16-21 1994, Sorrento, Italy, pp. 159-168.

[Yu94c] E. Yu, J. Mylopoulos, *Understanding "Why" in Software Process Modelling, Analysis, and Design*, Technical Report DKBS-TR-94-3, Department of Computer Science, University of Toronto, February 1994.

[Yu94d] E. Yu, J. Mylopoulos, From E-R to "A-R" – Modelling Strategic Actor Relationships for Business Process Reengineering, *Proc. 13th Int. Conf. Entity-Relationship Approach*, December 13-16 1994, Manchester, U.K., to appear.

[Yu94e] E. Yu, J. Mylopoulos, Organization Modelling for Business Process Reengineering, *AAAI-94 Workshop on Artificial Intelligence in Business Process Reengineering*, Seattle, Washington, U.S.A., July 31–August 4, 1994, pp. 9–14.

[Yu94f] E. Yu, J. Mylopoulos, Towards Modelling Strategic Actor Relationships for Information Systems Development – with Examples from Business Process Reengineering, *Proc. 4th Workshop on Information Technologies and Systems (WITS'94)*, Vancouver, B.C., Canada, December 17-18, 1994. A version of this paper was presented at the *Workshop on System Requirements: Analysis, Management, and Exploitation*, Schloß Dagstuhl, Saarland, Germany, October 4–7, 1994.

[Zave81] P. Zave, R. T. Yeh, Executable Requirements for Embedded Systems, *Proc. 5th Intl. Conf. Soft. Eng.*, 1981, pp. 295-304.

[Zisman78] M. D. Zisman, Use of Production Systems for Modeling Asynchronous, Concurrent Processes, *Pattern-directed Inference Systems*, Waterman and Hayes-Roth, eds., Academic Press, New York, 1978.