From Organization Models to System Requirements
A "Cooperating Agents" Approach

Eric Yu[†] , Philippe Du Bois[‡] , Eric Dubois[‡] and John Mylopoulos[†]

[†] Dept. of Computer Science, University of Toronto
Toronto, Ontario, M5S 1A4 (Canada)
{eric,jm}@cs.toronto.edu

[‡] Computer Science Dept., University of Namur
21 rue Grandgagnage, B-5000 Namur (Belgium)
{pdu,edu}@info.fundp.ac.be

Abstract

Increasingly, information systems development occurs in the context of existing systems and established organizational processes. Viewing organizational and system components as cooperating agents *offers a way of understanding their inter-relationships and how these relationships would or should be altered as new systems are introduced. In this paper, we show how two agent-oriented frameworks can be used in combination during requirements engineering for cooperative information systems. The* ALBERT *language is used to specify requirements, in terms of states and actions, and information and perception. The* i^* *framework is used to understand and redesign organizational processes, in terms of strategic relationships and rationales. A small banking example is used to illustrate how the requirements process may iterate between the two levels of modelling and analysis towards a requirements specification.*

1 Introduction

Increasingly, information systems development occurs in the context of existing systems and established organizational processes. For example, the development of systems to support a banking-by-phone service would have to take into account existing systems that store and process customer information and account transactions. These systems engage in processes that involve customers and bank employees (managers, tellers, accountants, etc.), which together constitute an organizational configuration that makes a banking service possible, and at some desired levels of availability, security, and quality of service. *Cooperation* among the many "agents" — whether they be human or computer-based — is crucial in order to attain these organizational goals.

Information systems can be viewed as being *cooperative* to the extent that they contribute to the achievement of organizational goals. In determining requirements for cooperative information systems, it is necessary to have an understanding of the organizational environment and goals, so that the resulting systems (which may be already existing, or to be developed) will work together with human agents to achieve overall goals (such as a phone-banking service that is viable from the customer's and the bank viewpoints). To support the development of cooperative information systems, we need models and frameworks that recognize that agents in distributed, dynamic organizations typically have limited knowledge about each other, and may have conflicting or complementary goals.

In the past decade, requirements modelling frameworks have been developed to assist in the understanding and specification of systems and their environments (e.g., [Bubenko, 1980; Borgida et al., 1985; Dubois et al., 1988]). More recently, goal-oriented frameworks for requirements engineering involving multiple agents have been developed (e.g., [Feather, 1987; Dubois, 1989; Fickas et al., 1992; Dardenne et al., 1993; Bubenko, 1983; Yu, 1995a]). In such frameworks, *goals* are either (i) associated with a set of constraints which describe restrictions on the behavior desired of the system (and that should result from the interleaving of the behaviors of the different system's components) or (ii) associated with organizational and business rules allowing to analyze the rationale underlying a system's architecture. We draw on this line of research to further develop requirements engineering techniques for cooperative systems.

Our contention is that a single conception of *agent* as embedded in a particular modelling framework may not be adequate to deal with the different types of analyses and reasoning that are needed during requirements engineering. Traditionally, requirements are usually taken to be specificational — as prescribing *what* systems should do. However, to understand and characterize the cooperative aspects of multi-agent systems, we need models that can express and help reason about *why* agents do what they do. Languages designed for prescribing agent behaviour are not well-suited for describing competing or complementary interests among agents, or for reasoning about strategic implications, such as those resulting from failure to adhere to prescriptive specifications.

In this paper, we view the requirements engineering effort as consisting of two levels — a specificational level which prescribes *what* agents should do or know, and an "understanding" level which describes *why* agents relate to each other in a certain way, and why they might prefer some other configuration of relationships. Adopting a two-levelled approach allows each level to offer agent concepts that are appropriate for that level of modelling and reasoning. We show how the two levels can work together to achieve better understanding of systems and their organizational en-

vironments, and to come up with system requirements.

The ALBERT language[1] [Dubois et al., 1993a; Dubois et al., 1994a; Dubois et al., 1994b] has been designed for specifying the (primarily functional) requirements of distributed real-time systems. Agents have states and actions. They are constrained in terms of obligations, information and perception. From an ALBERT specification, one can determine whether certain desired properties are satisfied. Agents cooperate by giving each other information about their own state of knowledge. ALBERT offers a higher-level view than earlier requirements languages (such as ERAE [Dubois et al., 1986] or RML [Greenspan, 1984]) through the use of agent-oriented concepts such as knowledge and obligation.

The i^* framework[2] [Yu, 1995a] is used to obtain an understanding about organizational relationships and the rationales behind them. Agents have wants and abilities. They depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. A model of *strategic dependencies* among agents can be analyzed for opportunities and vulnerabilities. A model of *strategic rationales* can assist in the search for alternative configurations of organizational relationships that can better address the strategic interests of agents, for example, by introducing information systems. i^* offers a more open and strategic conception of agents than other goal-oriented requirements frameworks (e.g., [Dardenne et al., 1993]).

We use the i^* models to support the generation and evaluation of organizational alternatives, and the ALBERT language to produce a requirements specification document for system development. As organizational requirements change (in the i^* models), they need to be reflected in the functional requirements (the ALBERT specification). Elaboration of the functional requirements may reveal further organizational issues that need to be addressed, resulting in an iterative process of refinement of the organizational and functional requirements. We anticipate that this iterative process would lead to a more systematic and thorough examination of organizational issues and system specifications than if either framework were used on its own.

As a running example in this paper, we consider a bank whose existing banking systems have been designed with the assumption that customers visit bank branches to conduct their business (i.e., a *customer* gives a transfer request to a *teller*, who validates it by verifying the identity of the customer; if it is OK, the teller transmits the transfer request to the *account handler* subsystem which processes it providing that balance of the customer's account permits it). Within that context, we envisage that the bank wishes to offer "banking-by-phone" as a new service. In order to decide what changes to the banking systems are required, we use i^* to obtain an understanding of the organizational and business environment — for example, that customers would like to be able to transfer funds more quickly and more conveniently, but are also concerned about security. Using means-ends reasoning in i^*, a PIN code (personal identification number) is proposed as a way to address the security goal. At the ALBERT level, one discovers that a PIN code does not offer security if the code is also known to other agents. Returning to the i^* models, we add the organizational requirement that customers be committed to keeping the PIN code confidential. These new requirements lead to changes to system requirements at the ALBERT level, which may in turn have further implications at the organization model level. The example is pedagogical and is meant to be suggestive of the much more complex sets of issues typically found in actual situations.

In Section 2, we briefly review the main features of ALBERT and i^*, using the traditional banking-by-teller arrangement as the example. Section 3 shows how the two levels of modelling work together to obtain system requirements for introducing a banking-by-phone service. In Section 4 we discuss our approach and compare with related work. We conclude in Section 5 with some observations about the implications of our approach, and outline some avenues for future research.

2 Features of ALBERT and *i**

In this section, we review the main concepts of ALBERT and i^* and we illustrate their use through the bank case study. The example is necessarily greatly simplified and does not reflect the complexity of real banking. It is only used here to illustrate the basic concepts of our approach.

2.1 Specifying requirements using the ALBERT language

The ALBERT language supports the modelling of functional requirements in terms of a collection (or *society*) of *agents* interacting together in order to provide services necessary for the organization. Each agent is characterized by *actions* that change or maintain its own *state* of knowledge about the external world and/or the states of other agents. Such actions are performed by agents in order to discharge contractual obligations expressed in terms of *internal* and *cooperation* constraints.

In ALBERT, functional requirements are expressed in terms of a set of formal statements in typed temporal first order formulas. The language has a formal semantics expressed in terms of RT-OSL [Böhm et al., 1993] (a specific real-time temporal logic enriched with the concept of object), and supports the encoding of requirements in both *declarative* and *operational* styles [Dubois et al., 1994b].

In order to enhance readability, a specification is organized into units called *agents*. Logical statements are grouped around agents in order to define the set of admissible behaviours (or *lives*) they may experience.

[1] ALBERT stands for *Agent-oriented Language for Building and Eliciting Real-Time requirements.*

[2] The name i^* (pronounced *i-star*) refers to the notion of distributed intentionality among cooperative agents.

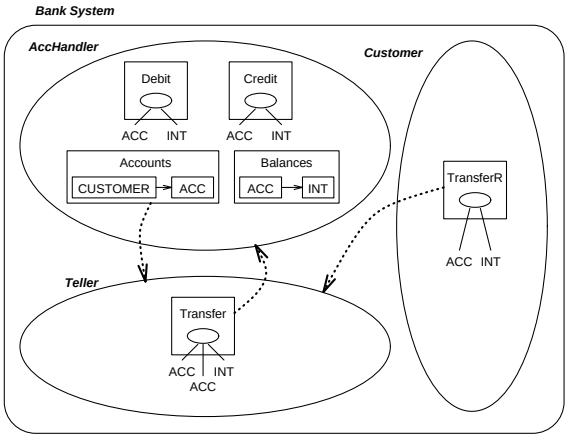

Figure 1: Structure of the Bank System

Logical statements describing an agent are classified into categories, each corresponding to a pattern of property. Such pattern provides guidance in the elicitation and structuring of requirements [3].

The language[4] is made up of (i) a graphical component in terms of which is *declared* the vocabulary of the application to be considered and (ii) a textual component in terms of which the specification of the admissible behaviours of agents is *constrained* through logical formulas. The Declarations and Constraints components of the banking-by-teller example are shown in Fig. 1 and 2 respectively.

Declarations

The Declarations component consists of a description of the general structure of the composite system in terms of agents as well as of the structure of each individual agent.

A specification consists of a collection of agents. Our small example (see Fig. 1) consists of three agents: *AccHandler* (declared as an individual agent), *Customer* and *Teller* (each declared as a population).

The declaration part of an agent consists of the description of its *state* structure (i.e. the memory of the agent) and the list of *actions* which may happen during the life of the agent and which may change the state of the agent. State components (graphically depicted with rectangles) are typed and actions (graphically depicted with ovals) can have typed arguments. Types may vary from simple data types to complex data types (recursively built using predefined type constructors).

In the example (see Fig. 1), the state of the *AccHandler* agent is structured into two tables (resp. *Accounts* and *Balances*). The index of the *Balances* table is of type *ACC*(ount) and the elements of type *INT*(eger). The *AccHandler* may perform two kinds of actions: *Credit* and *Debit*. Both have two arguments: the first of type *ACC* and the second of type *INT*.

In addition, the graphical notation also expresses visibility relationships linking agents to the outside. Dotted lines on Fig. 1 show how agents make information visible to other agents, e.g., the *Account* table of the *AccHandler* agent is exported to the *Teller* agent; on the contrary, the *Balances* table is shown to no other agents. Dotted lines also show how agents influence each others' behaviour through exportation of actions, e.g., the *AccHandler* agent is influenced by the *Transfer* actions of the *Teller* agent.

Constraints

Constraints are used for pruning the (usually) infinite set of possible lives associated with the agents of a composite system. The life of an agent is (usually) an infinite alternating sequence of changes (occurrences of actions) and states values. An admissible life will respect:

1. *local* constraints related to the internal behaviour of the agent;

2. *cooperation* constraints defining how the agent interacts with other agents.

Local constraints are classified under four headings. The use of two of them is illustrated in the example.

Effects of Actions The effect of an action is expressed through its functional characterization in terms of a mathematical relationship between successive information states (see, on Fig. 2, the effects of the *Credit* and *Debit* actions)

Causalities among Actions. Action triggering is usually ensured through ECA (Event-Condition--Action) rules, i.e., at any moment, when an event occurs if a condition on the current state is met, then the action happens. In ALBERT, this rather operational style of specification is supported (see below) but a more declarative style also permits to keep track of action occurrences and of specific causalities among them (see, on Fig. 2, the illustration of this concept of *process*).

Finally, there are two other available headings not used in the example. Under the **Capability** heading, we describe ECA rules. Besides the circumstances under which an action should or should not occur, the ALBERT language also introduces a more non-deterministic characterization where an action is said to be *permitted* under some circumstances (i.e. may or may not happen). This permits to express easily a statement like "the AccHandler may decide to close the account of a customer when it is in the red". This non-determinism is very important to be captured at the requirements engineering level where we are concerned with modelling real-world aspects. Under the **State Behaviour** heading, it is permitted to express properties related to the historical sequence of information states. For example, a statement like "a customer's account cannot be in the red for more than

[3]The usefulness of such patterns was also previously identified in the RML language [Greenspan et al., 1986] built on top of first order logic

[4]For a detailed presentation of ALBERT, see [Dubois et al., 1994b].

AccHandler

LOCAL CONSTRAINTS
EFFECTS OF ACTIONS
Credit(a,n): Balances[a] = Balances[a]+n
The account a is credited with amount n
Debit(a,n): Balances[a] = Balances[a]–n
The account a is debited by amount n
CAUSALITY
$tlr.Transfer(a1,a2,n) \xrightarrow{\Diamond_{\leq 1d}} Debit(a1,n); Credit(a2,n)$
A transfer order should be followed within at most 1 day, by the corresponding credit and debit operations.

COOPERATION CONSTRAINTS
ACTION PERCEPTION
$\mathcal{XK}$ (*tlr.Transfer(a,_,n)* / *Balances[a]>n–$2000*)
A transfer order is processed by the AccHandler if and only if the resulting balance of the customer's account does not reach more than a $2000 overdraft.
STATE INFORMATION
$\mathcal{XK}$ (*Accounts.TELLER* / TRUE)
The Accounts table is always shown to all Tellers.

Customer

COOPERATION CONSTRAINTS
ACTION INFORMATION
$\mathcal{XK}$ (*Transfer(_,_).TELLER* / TRUE)
In any situation, the customer send his/her transfer orders to a Teller.

Teller

LOCAL CONSTRAINTS
CAUSALITY
$c.TransferR(a,n) \xrightarrow{\Diamond} Transfer(AccHandler.Accounts[c],a,n)$
A valid transfer request from the Customer is echoed by a transfer order sent to the AccHandler.

COOPERATION CONSTRAINTS
STATE PERCEPTION
$\mathcal{XK}$ (*AccHandler.Accounts* / TRUE)
The Pbs can always consult the Accounts table maintained by the AccHandler.
ACTION PERCEPTION
$\mathcal{XK}$ (*c.TransferR(a,n)* / $c \in DOM(AccHandler.Accounts)$)
A transfer request is valid if and only if the customer has an account at the bank. The Teller knows the identity of the customer responsible for the transfer.
ACTION INFORMATION
$\mathcal{XK}$ (*Transfer(_,_,_).AccHandler* / TRUE)
In any situation, the Pbs send his transfer orders to the AccHandler.

Figure 2: Constraints on the *AccHandler*, the *Customer* and the *Teller* agents

1 month" could be straight-forwardly mapped in an equivalent formal ALBERT statement.

Cooperation constraints are classified under four headings describing how an agent perceives action performed by other agents (**Action Perception**), how it can see parts of the state of other agents (**State Perception**), how it lets other agents know of the actions that it does (**Action Information**) and how it show parts of its state to other agents (**State Information**). Perception and information provide the analyst a way to add a dynamic dimension to the importation and exportation relationships between agents expressed in the declaration part of the specification. The headings are illustrated on Fig. 2, e.g., the **Action perception** constraint of the *AccHandler* specification defines the conditions under which the *AccHandler* agent is influenced by *Transfer* actions of the *Teller* (in this case, if and only if the transfer will not cause an overdraft of more than $2000).

2.2 Understanding Organizational Relationships Using *i**

When redesigning systems to meet new requirements, we usually need to have a broad understanding of the organizational environment and goals, leading to decisions about what changes to make, and which components can remain.

The i^* framework provides understanding of the "why" by modelling organizational relationships that underlie system requirements. Agents are taken to have goals, and use knowhow and resources in their attempts to achieve goals. The framework includes two models. In the Strategic Dependency model, agents are modelled as depending on each other for goals to be achieved, tasks to be performed, and resources to be furnished. In the Strategic Rationale model, the reasoning that each agent has about its relationships with other agents are described. It supports reasoning about alternative ways for meeting goals, and for evaluating them. Agents are strategic in that they are concerned about opportunities and vulnerabilities.

The framework is intended to assist in gaining a deeper understanding about the organizational environment, help explore alternative patterns of relationships (among software, hardware and human components), to discover the implications of these alternatives for each agent, and to help make tradeoff among the alternatives.

The framework has been presented earlier in the context of information systems requirements engineering [Yu, 1993], business process reengineering [Yu et al., 1994a], and software process modelling [Yu et al., 1994b].

2.2.1 The Strategic Dependency Model

Figure 3 shows a Strategic Dependency model of the banking-by-teller example. The basic relationship is that a Customer depends on the Bank to have funds transferred from account a1 to a2. The Customer also depends on the bank for the transfer operation to be secure, namely, that only he himself (the owner of the account) can initiate a transfer. The unit in the bank which does the transfer — the account handler — depends on the (human) teller to verify the identity

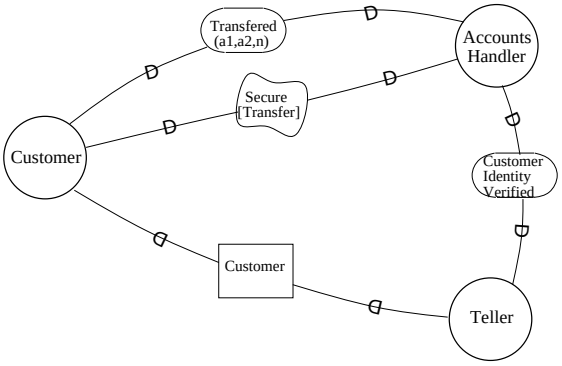

Figure 3: A Strategic Dependency model of banking-by-teller

of the customer. To accomplish this, the teller depends on the physical presence of the customer in the bank.

The SD model provides four types of dependency links. In a *goal dependency*, one agent depends on another to bring about a condition in the world — for example, that funds be transferred from one account to another). The depender does not care in what way the dependee accomplishes the condition.

In a *task dependency*, the depender tells the dependee what to do by specifying how. For example, if the account handler specifies the steps that the teller should go through in verifying customer identity, it would be a task-dependency.

In a *resource dependency*, the depender depends on the availability of an entity as a resource, e.g., (the physical presence of) the customer.

A *softgoal dependency* is similar to a goal dependency except that the condition is not sharply defined a priori. What is "secure" is a matter of interpretation. While the bank may provide measures for security, it is the customer who decides whether they are secure enough for his purposes.

The SD model provides for different degrees of strength of dependency: *open*, *committed*, and *critical* [Yu, 1995a]. The model can also distinguish *agents* from the *roles* that they play and the *positions* that they occupy. In this paper, we will limit our examples to the basic features.

The SD model can be analyzed in terms of opportunities and vulnerabilities. The funds transfer facility offered by the bank enables a customer to, say, cover a cheque using funds from another account. However, in depending on the bank to carry out the transfer, if the bank fails to transfer the funds properly, the customer is vulnerable to the failure, potentially resulting in an overdraft in his account. Agents who are depended on often in turn depend on other agents. The customer's dependency for transfer of funds (and its security) further involves a dependency on the teller to identify the customer, which in turn depends on the physical presence of the customer (see Fig. 3).

Modelling organizational processes in terms of intentional dependencies provides a level of description that acknowledges that organizational actors are often able to cope with open-ended situations (such as exceptions) without fully pre-planned activity steps[Yu, 1995b]. The i^* models are formally represented in the conceptual modelling language Telos [Mylopoulos et al., 1990] and their semantics are characterized by adapting formulations of intentional concepts such as goal, belief, ability, and commitment (e.g., [Cohen et al., 1990]). The underlying conceptual modelling framework allows large amounts of knowledge to be managed along knowledge structuring dimensions such as classification, generalization, aggregation, and time in order to deal with large scale real-life application domains.

2.2.2 The Strategic Rationale Model

Whereas the Strategic Dependency model gives an external view of how agents depend on each other, the Strategic Rationale model gives a more detailed description of the rationales behind the dependencies. One can answer "why" questions more precisely. There are two main types of relationships — means-ends relationships and task decompositions.

Figure 4 shows that the customer has as a goal that funds be transferred. The means for achieving this end is the task "Request Transfer At Bank". A means-ends relationships suggests that there can be other means for achieving the same end. We show this in the next section. The task of requesting transfer at bank can be further detailed by decomposing it into the subtasks of visiting the bank, and then requesting the transfer in person at the teller.

In transferring funds, the customer also has a number of quality goals (or softgoals) that are desired. He wants the transfer to be secure, quick, convenient, and that the service be friendly. Different ways of transferring funds (i.e., different organizational configurations) may be evaluated as contributing positively or negatively to these goals. In Fig. 4, requesting a transfer at the bank is considered to be good for security, and is good in friendly service. However, having to visit the bank is bad from a convenience standpoint. It is also negative for quickness, for example, due to long lineups at the teller. Softgoals may be correlated with each other. For example, quick service is considered to have a positive contribution to convenience. In general the softgoals form a graph, and their mutual influences can be evaluated by using a qualitative reasoning scheme (e.g., [Chung, 1993]).

These relationships provide a representation of the rationales because they explain why the dependencies are arranged in a given configuration. When there are alternatives, the softgoals also serve as evaluation criteria. These rationales can also help in coming up with new alternatives for achieving goals (i.e., addressing the strategic interests of the various stakeholders).

Note that the models (SD and SR) are usually incomplete; only items of strategic concern are included in the model (i.e., items that are considered to make a difference in choosing one configuration over another).

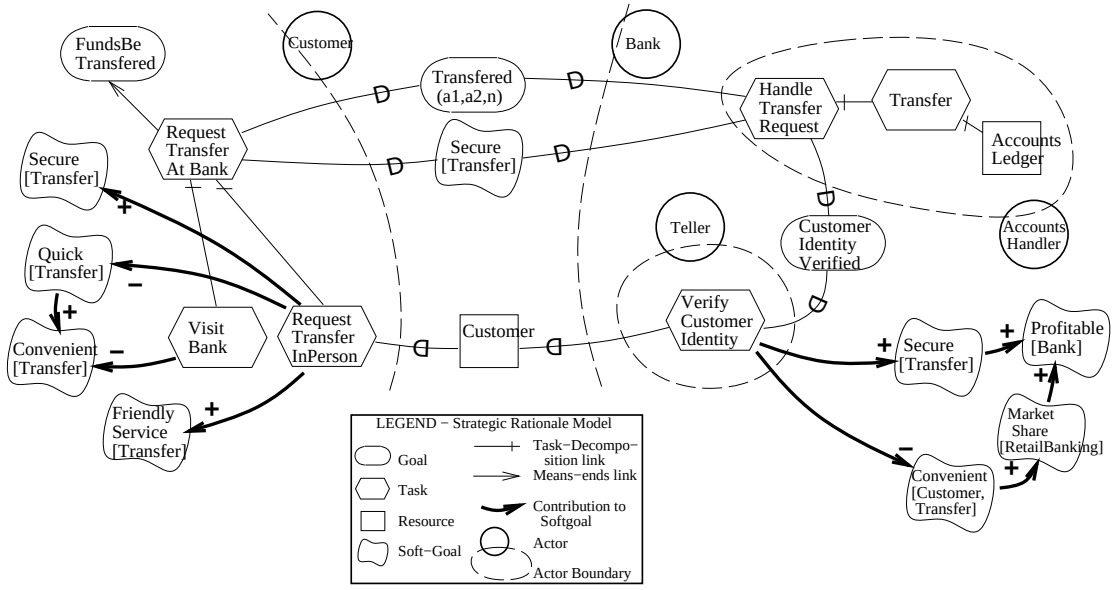

Figure 4: A Strategic Rationale model for banking-by-teller

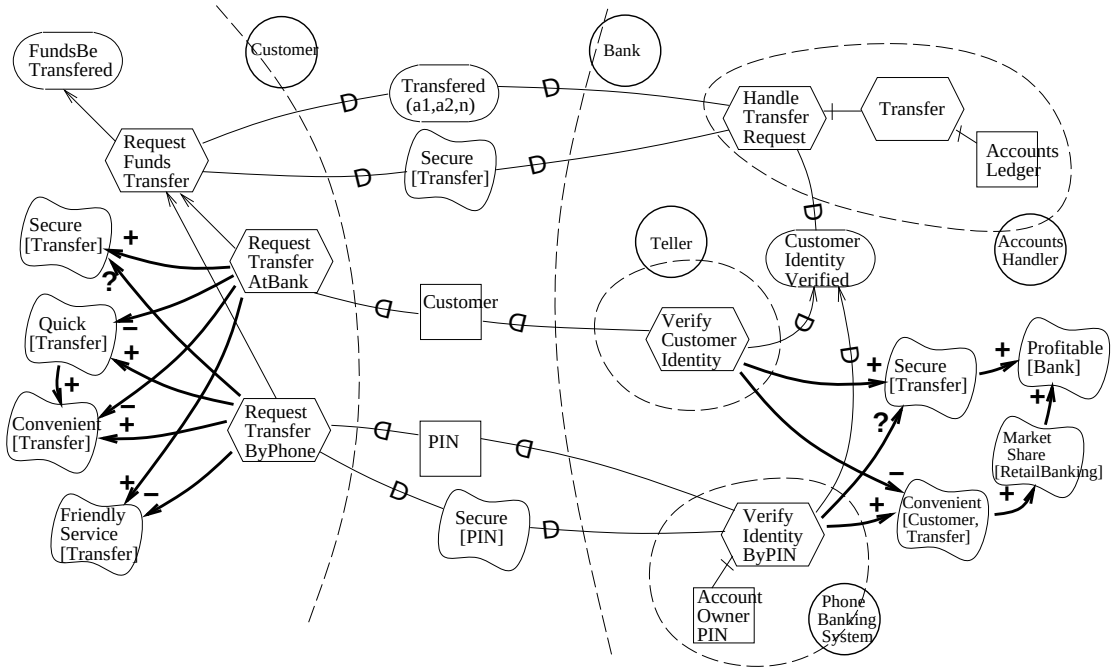

Figure 5: A Strategic Rationale model of banking-by-phone as an alternative to banking-by-teller

3 From Organizational Alternatives to System Requirements

In the above, we have shown how i^* provides an understanding of the organizational relationships in a business domain, while ALBERT is used to specify system requirements. By coupling the two frameworks, one can gain confidence in the ALBERT specifications by linking each fragment of the ALBERT specification to the fulfillment of some organizational goals in the i^* models. Given some organizational goals, the process of obtaining system (functional) requirements is usually far from straight-forward. Typically, one would need to go back and forth between organization modelling and system requirements because issues discovered in one level will need to be looked at from the other level. This section shows how the i^* framework and the ALBERT language can be used in conjunction

to assist in this process.

Figure 5 shows an initial attempt at considering banking-by-phone as an alternative to conventional teller-based banking. A goal of the bank is to be profitable. One way is to increase market share by attracting more business and customers. To do this, banking services should be convenient for the customer. One way to make transfer of funds more convenient is to allow customers to do it over the telephone. From the customer's viewpoint, banking-by-phone is seen to be quicker and more convenient, although it is less user-friendly. A PIN code is proposed as a means for meeting the goal that the customer be identified in phone banking. The Strategic Dependency model for the proposed banking-by-phone configuration is shown in Fig. 6.

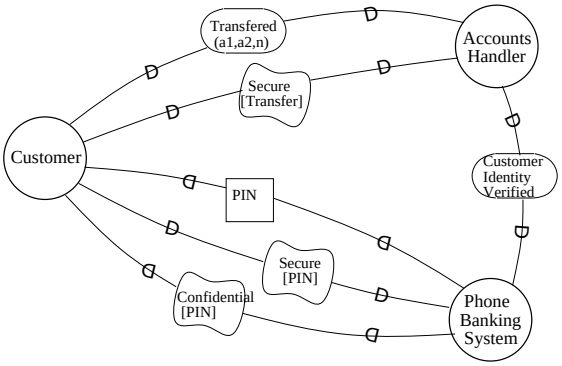

Figure 6: A Strategic Dependency model of banking-by-phone

At the ALBERT level, Fig. 7 shows the new system structure and Fig. 8 shows the system requirements specification produced in response to the organizational goals identified above. In particular, at the level of the 'Customer' and 'Pbs' agents level, one may notice several changes with respect to the original banking-by-teller system (see Fig. 2). These changes pertain to

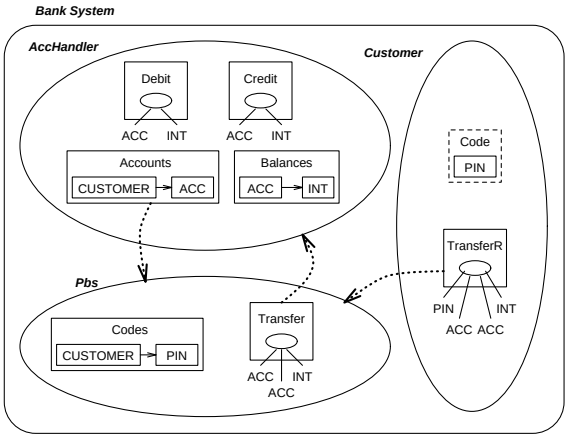

Figure 7: Structure of the Bank System (revised version)

AccHandler

LOCAL CONSTRAINTS

EFFECTS OF ACTIONS

Credit(a,n): Balances[a] = Balances[a]+n

The account a is credited of a n amount

Debit(a,n): Balances[a] = Balances[a]–n

The account a is debited of a n amount

CAUSALITY

$Pbs.Transfer(a1,a2,n) \xrightarrow{\Diamond_{<1d}} Debit(a1,n);\ Credit(a2,n)$

A transfer order should be followed within at most 1 day, by the corresponding credit and debit operations.

COOPERATION CONSTRAINTS

ACTION PERCEPTION

$\mathcal{XK}$ (*Pbs.Transfer(a,_,n)* / *Balances[a]*$>$ *n–$2000*)

A transfer order from the Pbs is processed by the BankIS if and only if the resulting balance of the customer's account does not reach more than a $2000 overdraft.

STATE INFORMATION

$\mathcal{XK}$ (*Accounts.Pbs* / TRUE)

The Accounts table is always shown to the Pbs.

Customer

COOPERATION CONSTRAINTS

ACTION INFORMATION

$\mathcal{XK}$ (*TransferR(_,_,_,_).Pbs* / TRUE)

In any situation, the customer send his/her transfer requests to the Pbs. He/she validates the transfer with his/her PIN code.

Pbs

LOCAL CONSTRAINTS

CAUSALITY

$c.TransferR(\_,a1,a2,n) \xrightarrow{\Diamond} Transfer(a1,a2,n)$

A valid transfer request from the Customer is echoed by a transfer order sent to the AccHandler.

COOPERATION CONSTRAINTS

STATE PERCEPTION

$\mathcal{XK}$ (*BankIS.Accounts* / TRUE)

The Pbs can always consult the Accounts table maintained by the AccHandler.

ACTION PERCEPTION

$\mathcal{XK}$ (*_.TransferR(p,a,_,n)* / $\exists$ *c, Code[c]=p* $\wedge$ *BankIS.Accounts[c]=a*)

A transfer request is valid if and only if the PIN code corresponds to the owner of the account to be debited.

ACTION INFORMATION

$\mathcal{XK}$ (*Transfer(_,_,_).AccHandler* / TRUE)

In any situation, the Pbs send his transfer orders to the AccHandler.

Figure 8: Constraints on the *AccHandler*, the *Customer* and the *Pbs* agents (revised version)

the responsibilities of customers for communicating a PIN code information and of the 'Pbs' for discovering the identity of the customer on the basis of the PIN

code knowledge.

At this stage in the requirements process, the analyst still has to question himself about the adequacy of the PIN code at the level of security issues (see the question mark on Fig. 5). In analyzing the ALBERT specification, it turns out that the use of a PIN code by itself does not guarantee the identity of the customer. The action information constraint in the *Customer* specification says that the PIN code accompanying a transfer request is shown to the *Pbs*. The action perception constraint in the *Pbs* agent specification says that a transfer request is valid if and only if the given PIN code corresponds to that of the owner of the account. This set of constraints does not preclude someone other than the account owner from making a valid transfer using the correct PIN code. The security goal is therefore not met.

Returning to the i^* level, because we need to reduce the possibility, for a customer, to use the PIN of another customer, we have to express an additional goal related to the Bank's dependency on Customer for the confidentiality of PIN. This is a soft goal because it does not appear that there are definitive procedures that can guarantee complete confidentiality.

While at the i^* level, one can identify correlations with other goals arising from the need to keep the PIN confidential. For example, having to keep the PIN confidential contributes negatively to convenience since the customer may forget the PIN.

Additional requirements related to the properties of the PIN code may also be identified through this process. For example, if the number of possibilities for a PIN code is large, then it will be more difficult for a customer to use a wrong one. One idea would be to define the procedure in which PIN codes are assigned to customers so that no two customers will have the same PIN code. This would be reflected as a uniqueness constraint in the ALBERT specification.

Other solutions are also possible like those related to complexity (e.g., the number of digits) of the PIN code. Again, these different solutions all have impacts at the i^* level. For example, imposing a 25-digit PIN code has a positive impact on the confidentiality softgoal, but a negative impact on the convenience softgoal for the customer.

Throughout the requirements process, the analyst needs to iterate back and forth between system requirements and organizational requirements in order to deal with their impacts on each other.

4 Discussion and Related Work

In the above phone-banking example, we have demonstrated that the concept of cooperating agents can offer a good understanding of organizational relationships and goals, and also for stating and analyzing requirement specifications. We have illustrated why cooperation needs to be understood in terms of intentional concepts such as knowledge, commitment, obligation, and goals.

Our two-levelled approach allows us to adopt different concepts of agents at each level that are well-suited to the type of modelling and reasoning for that level. At the level of understanding organizational relationships, we need a notion of agent that recognizes that agents have freedom, and may violate constraints or commitments. One needs to reason about the implications of these violations. The models are used descriptively, to understand the organizational conditions as they are (or might be, in the case of proposed configurations). We need to take a strategic view of agent relationships because new work arrangements alter the configuration of dependencies. The introduction of new systems and/or work processes changes what is possible or not possible, or change the degree of difficulty in achieving goals. Models at this level tend to be very incomplete, but this is appropriate since only issues that are of strategic significance need to be considered.

At the level of requirements specification, a prescriptive view is more appropriate than a descriptive view. Analysts want to be able to confirm that an organizational configuration has certain desired properties, *assuming* that agents abide by the stated restrictions on their behaviour (the obligations) in a declarative way. The specification level typically requires a much higher degree of completeness, in order to be able to guarantee certain properties. Finer-grained modelling concepts such as states, actions, obligations, information, perception and real-time constraints are appropriate.

Our approach may be compared to other frameworks for requirements engineering which take a multi-agent or organizational perspective. The framework for enterprise modelling of [Bubenko, 1993] is similar in spirit in several ways. It emphasizes the need to model organizations and their actors, their motivations and rationales [Nellborn et al., 1994]. It also uses multiple, inter-linked models. The informal (but structured) organizational models are linked to more formal specification models. Our approach using ALBERT and i^* is comparable, but adopts a set of intentional concepts explicitly, with more precise semantics. This will allow more computer-based support.

In the KAOS framework [Dardenne et al., 1993], overall goals are explicitly modelled (following the concept of Composite Systems Design [Feather, 1987; Feather, 1994; Fickas et al., 1992]). Goals are reduced through means-ends reasoning to arrive at responsibilities for agents. The modelling of agents is specificational and prescriptive. Since agents are assumed to conform to prescribed behaviour, one cannot easily analyze strategic relationships and implications.

A number of organization modelling frameworks have been proposed in the organization information systems area, e.g., [Blyth et al., 1993]. Dependency concepts have also been used for modelling coordination in organizations, e.g., [Malone et al., 1994].

The i^* framework differs from these in that it highlights the strategic dimension of agent relationships, and de-emphasizes the operational aspects. Similarly, although multi-agent cooperation has received considerable attention in distributed artificial intelligence

(DAI) (e.g., [Bond et al., 1988]), the emphasis has been on the division of computational work (e.g., the reduction of goals to primitive actions for execution by robots or software programs), and much less on the strategic interests of organizational, social actors [Gasser, 1991]. The DAI community has also developed communication and coordination mechanisms and protocols (such as KQML and KIF) which can serve as alternatives to more conventional implementation techniques for meeting the organizational requirements and specifications at the levels described in this paper.

Finally at the specification level, the ALBERT language is very much in the line of recent formal specification languages designed for the purpose of modelling functional requirements (e.g., MAL [Finkelstein et al., 1987], and ERAE [Dubois et al., 1991], DAL [Ryan et al., 1991], LCM [Feenstra et al., 1993] and TROLL [Saake et al., 1993]. The major difference is the application scope of ALBERT related to the modelling of complex real-time cooperative (distributed) systems.

In this paper, we have concentrated on showing how i^* and ALBERT can work together. Comparisons of i^* and ALBERT to their respective related work and more detailed discussions can be found in [Yu, 1995b; Yu, 1995a; Yu et al., 1994a] and [Dubois et al., 1993b; Dubois et al., 1994a; Dubois et al., 1994b].

5 Conclusions

As information system development techniques and tools advance, we anticipate that the technical design and implementation stages will occupy a less central place in system development. On the other hand, new systems are becoming more interconnected, and increasingly interwoven into complex organizational processes. The challenge in information system development will shift towards the understanding of organizational environments and needs, and how to make decisions involving technical systems to address those needs and concerns [Jarke, 1994].

To this end, we need a clearer understanding of what it means for systems to be "cooperative". Systems that are merely interconnected, but which may have been designed by different groups, at different times, to serve the purposes and interests of different parties, are not necessarily cooperative. In this paper, we have argued that a characterization of cooperation requires the use of intentional concepts. Agents and how they relate to each other need to be characterized in terms of concepts such as knowledge, obligation, commitments, and goals. Information systems (and other kinds of agents) are cooperative to the extent that they contribute to some larger, overall goals in an organizational context.

To make this kind of understanding and analysis concrete and amenable to support by computer-based tools, we have brought together two agent-oriented modelling frameworks, both based on formal knowledge representation techniques. Each offers a set of capabilities for its respective level of modelling and reasoning. We have demonstrated that a requirements analyst needs to iterate over the two levels of modelling and analysis to arrive at system requirements.

This work is preliminary. We have outlined the approach and illustrated it through a realistic but small example. In ongoing work, we are studying larger real-life cases to test the practicality of the approach. In future work, we plan to elaborate on the steps needed to obtain system requirements from the analysis of strategic organizational relationships, and to identify the types of situations where analysis of the specification would suggest changes to the organization model.

Another direction for future work is related to the development of knowledge-based tools for supporting the requirements engineering process. In these tools, the two languages — ALBERT and i^* — can be (weakly-) coupled by using a common underlying conceptual modelling framework which relate representational objects by knowledge structuring relationships such as classification, generalization, aggregation and time (e.g., as provided by the Telos language [Mylopoulos et al., 1990]). This approach may be seen as an extension of the approach adopted in the DAIDA project [Jarke et al., 1992], where three different sets of concepts were used for representing knowledge about the requirements, design, and implementation phases during system development, and which are linked and managed by a common global knowledge base management system.

Acknowledgements: We thank the anonymous referees for suggestions for improving the paper.

This work has been partially supported by the Permanent Joint Commission Canada / French Community of Belgium, and the Natural Sciences and Engineering Research Council of Canada.

References

[Blyth et al., 1993] A.J.C. Blyth, J. Chudge, J.E. Dobson, and M.R. Strens. ORDIT: a new methodology to assist in the process of eliciting and modelling organisational requirements. In Simon Kaplan, editor, *Proc. of the conference on organizational computing systems – COOCS'93*, pages 216–227, Milpitas CA, November 1-4, 1993. ACM Press.

[Bond et al., 1988] A.H. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufman Publishers, 1988.

[Borgida et al., 1985] Alexander Borgida, Sol Greenspan, and John Mylopoulos. Knowledge representation as the basis for requirements specifications. *IEEE Computer*, pages 82–91, April 1985.

[Böhm et al., 1993] Klemens Böhm and Amilcar Sernadas. Real-time object specification logic. Technical Report 7/93, Departamento de Matemática, Instituto Superior Técnico, Lisbon (Portugal), March 1993.

[Bubenko, 1980] J.A. Bubenko. Information modeling in the context of system development. In S.H. Lavington, editor, *Information Processing 80*, pages 395–411. North-Holland, 1980.

[Bubenko, 1983] Janis A. Bubenko. On concepts and strategies for requirements and information analysis. In *Information modeling*, pages 125–169. Chartwell-Bratt, 1983.

[Bubenko, 1993] Janis A. Bubenko. Extending the scope of information modeling. In *Proc. of the 4th International Workshop on the Deductive Approach to Information Systems and Databases*, pages 73–98, Lloret-Costa Brava (Spain), September 20-22, 1993.

[Chung, 1993] Lawrence Chung. *Representing and using Non-functional Requirements: a Process-Oriented Approach*. PhD thesis, Computer Science Department, University of Toronto, Toronto (Canada), June 1993.

[Cohen et al., 1990] P.R. Cohen and H.J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3), 1990.

[Dubois et al., 1986] Eric Dubois, Jacques Hagelstein, Eugene Lahou, Frank Ponsaert, Andre Rifaut, and Fiona Williams. The ERAE model: a case study. Manuscript M136, Philips Research Laboratory, Brussels (Belgium), January 1986.

[Dubois et al., 1988] Eric Dubois, Jacques Hagelstein, and André Rifaut. Formal requirements engineering with ERAE. *Philips Journal of Research*, 43(3/4):393–414, 1988.

[Dubois, 1989] Eric Dubois. A logic of action for supporting goal-oriented elaborations of requirements. In *Proc. of the 5th International Workshop on Software Specification and Design – IWSSD'89*, pages 160–168, Pittsburgh PA, May 19-20, 1989. IEEE, CS Press.

[Dubois et al., 1991] Eric Dubois, Jacques Hagelstein, and André Rifaut. A formal language for the requirements engineering of computer systems. In André Thayse, editor, *From natural language processing to logic for expert systems*, chapter 6. Wiley, 1991.

[Dubois et al., 1993a] Eric Dubois, Philippe Du Bois, and Michaël Petit. O-O requirements analysis: an agent perspective. In O. Nierstrasz, editor, *Proc. of the 7th European Conference on Object-Oriented Programming – ECOOP'93*, pages 458–481, Kaiserslautern (Germany), July 26-30, 1993. LNCS 707, Springer-Verlag.

[Dubois et al., 1993b] E. Dubois, Ph. Du Bois, and J.-M. Zeippen. Object-oriented formal development of cooperative information systems. In *Proc. of the ECOOP'93 Workshop on the Application of Object-Oriented Formal Methods*, Kaiserslautern (Germany), July 26-30, 1993.

[Dubois et al., 1994a] Eric Dubois, Philippe Du Bois, and Frédéric Dubru. Animating formal requirements specifications of cooperative information systems. In *Proc. of the Second International Conference on Cooperative Information Systems – CoopIS-94*, pages 101–112, Toronto (Canada), May 17-20, 1994. University of Toronto Press inc.

[Dubois et al., 1994b] Eric Dubois, Philippe Du Bois, Frédéric Dubru, and Michaël Petit. Agent-oriented requirements engineering: A case study using the albert language. In A. Verbraeck, H.G. Sol, and P.W.G. Bots, editors, *Proc. of the Fourth International Working Conference on Dynamic Modelling and Information System – DYNMOD-IV*, Noordwijkerhoud (The Netherlands), September 28-30, 1994. Delft University Press.

[Dardenne et al., 1993] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.

[Feather, 1987] Martin S. Feather. Language support for the specification and development of composite systems. *ACM Transactions on Programming Languages and Systems*, 9(2):198–234, April 1987.

[Feather, 1994] Martin S. Feather. Composite system design. In *Proc. of the ICSE-16 Workshop on Research Issues in the Intersection Between Software Engineering and Artificial Intelligence*, Sorrento (Italy), May 16-20, 1994.

[Fickas et al., 1992] Stephen Fickas and Rob Helm. Knowledge representation and reasoning in the design of composite systems. *IEEE Transactions on Software Engineering*, SE-18(6):470–482, June 1992.

[Finkelstein et al., 1987] Anthony Finkelstein and Colin Potts. Building formal specifications using "structured common sense". In *Proc. of the 4th International Workshop on Software Specification and Design – IWSSD'87*, pages 108–113, Monterey CA, April 3-4, 1987. IEEE, CS Press.

[Feenstra et al., 1993] Remco B. Feenstra and Roel J. Wieringa. LCM 3.0: A language for describing conceptual models – syntax definition. Technical Report IR-344, Faculteit der Wiskunde en Informatica, Vrije Universiteit Amsterdam, Amsterdam (The Netherlands), December 1993.

[Gasser, 1991] L. Gasser. Social conceptions of knowledge and action: DAI foundations and open systems semantics. *Artificial Intelligence*, 47:107–138, 1991.

[Greenspan et al., 1986] Sol J. Greenspan, Alexander Borgida, and John Mylopoulos. A requirements modeling language. *Information Systems*, 11(1):9–23, 1986.

[Greenspan, 1984] Sol J. Greenspan. Requirements modeling: a knowledge representation approach to software requirements definition. Technical Report CSRG 155, Computer Science Department, University of Toronto, Toronto (Canada), March 1984.

[Jarke et al., 1992] M. Jarke, J. Mylopoulos, J.W. Schmidt, and Y. Vassiliou. DAIDA: An environment for evolving information systems. *ACM Transactions on Information Systems*, 10(1):1–50, January 1992.

[Jarke, 1994] M. Jarke and K. Pohl. Requirements engineering in the year 2001: On (virtually) managing a changing reality. In *Proc. of the Workshop on System Requirements: Analysis, Management, and Exploitation*, Shloß Dagstuhl, Saarland (Germany), October 4-7, 1994.

[Mylopoulos et al., 1990] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: A language for representing knowledge about information systems. *ACM Transansaction on Information Systems*, 8(4):325–362, October 1990.

[Malone et al., 1994] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *Computing Surveys*, 26:87–119, March 1994.

[Nellborn et al., 1994] C. Nellborn and P. Holm. Capturing information systems requirements through enterprise and speech act modeling. In Gerard Wijers, Sjaak Brinkkemper, and Tony Wasserman, editors, *Proc. of the 6th conference on advanced information systems engineering – CAiSE'94*, pages 172–185, Utrecht (The Netherlands), June 6-10, 1994. LNCS 811, Springer-Verlag.

[Ryan et al., 1991] Mark D. Ryan, Jose Fiadeiro, and Tom Maibaum. Sharing actions and attributes in modal action logic. In T. Ito and A. Meyer, editors, *Theoretical Aspects of Computer Software*. Springer-Verlag, 1991.

[Saake et al., 1993] Gunter Saake, Ralf Jungclaus, and Thorsten Hartmann. Application modelling in heterogenous environments using an object specification language. In *Proc. of the International Conference on Intelligent and Cooperative Systems – ICICIS'93*. IEEE CS Press, 1993.

[Yu et al., 1994a] Eric S. K. Yu and John Mylopoulos. From E-R to "A-R" – modelling strategic actor relationships for business process reengineering. In *Proc. of the 13th International Conference on the Entity-Relationship Approach – ER'94*, Manchester (UK), December 13-16, 1994.

[Yu et al., 1994b] Eric S. K. Yu and John Mylopoulos. Understanding "why" in software process modelling, analysis, and design. In *Proc. of the 16th International Conference on Software Engineering – ICSE'94*, Sorrento (Italy), May 16-21, 1994. IEEE & ACM.

[Yu, 1993] Eric S. K. Yu. An organization modelling framework for information systems requirements engineering. In *Proc. of the 3rd Workshop on Information Technologies and Systems – WITS'93*, Orlando FL, December 4-5, 1993.

[Yu, 1995a] Eric S. K. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, Computer Science Department, University of Toronto, Toronto (Canada), 1995. Also appears as Technical Report DKBS-TR-94-6, December 1994.

[Yu, 1995b] Eric S. K. Yu. Models for supporting the redesign of organizational work. Submitted for publication, 1995.