Modelling Organizations for Information Systems Requirements Engineering

Eric S. K. Yu
Dept. of Computer Science, University of Toronto
Toronto, Ontario, Canada M5S 1A4

Abstract

In attempting to understand information system environments during requirements engineering, it is often helpful to have an understanding of the "whys" as well as the "whats" about the environment. A natural way to answer *why* questions is by tracing them to *goals*. In an *organizational* environment, however, the "whys" do not originate from a single set of given goals. Organizational *agents depend* on each other for *goals* to be achieved, *tasks* to be performed, and *resources* to be furnished. A *requirements model* that captures knowledge about an organizational environment can be enriched by including the *network* of *dependency relationships* among agents. In this paper, we propose a set of *intentional* operators for modelling dependencies among agents, and present a preliminary axiomatic characterization.

1 Introduction

Modelling the *environment* in which software systems operate has been recognized as an important part of the requirements engineering process (e.g., [1, 22, 12]). In recent years, a number of frameworks and languages for *requirements modelling* have been proposed (e.g., [13, 8, 17]). We focus on *organizational* environments — an important class of environments within which many computer-based *information systems* operate.

Consider an engineering project organization in which members perform project tasks such as design, prototyping, and testing under the direction of a manager. The manager has to deliver the completed project to a client on time and within budget. To this end, she must monitor progress and control expenses.

Using a *requirements modelling language* such as RML [13], one could model this organization in terms of activities (carrying out project tasks, submitting progress reports, obtaining trip approvals), entities (project deliverables, expense reports), and assertions (that a project task is complete, that a trip is approved). Such a body of knowledge captures *what* an organizational environment is like — what activities occur, what entities exist, and what conditions hold.

In understanding an organization, it is often not enough to understand *what is the case*, but also *why it is the case* — why do milestones exist? why do expenses have to be approved? The relevance of "why" questions suggests that requirements models might be enriched by including *intentional* aspects of an organizational environment, i.e., elements of motivation, intent, purpose, and rationales.

A natural way to answer *why* questions is by tracing them to *goals*. *Why* milestones exist can be related to the goal that projects be completed on time. *Why* expenses need to be approved can be related to the goal of not exceeding allocated budgets. The concept of *goals* is extensively used in Artificial Intelligence, and has been adopted in a number of design and requirements frameworks (e.g., [7, 18]).

In an *organizational* environment, the "whys" behind the "whats" do not originate from a single set of overall, global goals. An organization consists of numerous processes occuring simultaneously, asynchronously, following diverse paths, intersecting each other at various points. If we were to see organizational processes as being goal-driven, then we would see the effects of goals propagating through the organization not in a single uniform direction, but in complex patterns that criss-cross each other, often not having definite beginnings or ends. Goals, and goal-driven processes in organizations, therefore, need to be described from particular perspectives or vantage points. These are the vantage points of *agents* in the organization.

The notion of *agent* as being the locus of knowledge and action is also a familiar one in AI (e.g., [19]). Rational agents use means-ends knowledge (or *knowhow*) to select *actions* that achieve *goals*. Models for characterizing agents in terms of intentional properties such as knowledge, belief, goals, ability, and commitment have been developed (e.g., [4, 21, 14]). Most treatments of agents, however, do not deal with agents that are embedded in a rich organizational context [11].

When agents *participate* in an organization, they are able to accomplish many things that they are unable to achieve as individuals in isolation. However, they are no longer entirely free to choose their own goals or actions. Each organizational agent *depends on* others for accomplishing some parts of what it wants, and are in turn *depended on* by others. Agents have *wants* that are met by others' *abilities*, *run* tasks that are *performed* by others, and *deploy* resources that are *furnished* by others. These dependencies form a complex and intricate network of *intentional relationships* among agents that might be called the *intentional structure* of the organizational environment.

In this paper, we propose a set of operators for modelling intentional *dependency* relationships among organizational agents. We build on the requirements modelling language RML and its successor Telos [17]. In section 2, we introduce each operator informally using our project organization domain example. In section 3, we give a preliminary axiomatic characterization of the operators by relating them to the *plans* that an agent has for achieving goals. Section 4 outlines a context in which these operators might be used, and discusses related work. Section 5 summarizes this research and points to future work.

2 Agent dependencies

We distinguish among three main types of agent dependencies. In **goal dependency**, one agent, the *depender*, depends on another, the *dependee*, for the fulfilment of a goal. The dependee is free to choose *how* to accomplish the goal. The depender is only interested in the outcome. In **task dependency**, a depender agent depends on some dependee agent for the performance of a task. The task specification constrains the choices that the dependee can make regarding *how* the task is to be carried out. In **resource dependency**, a depender agent presupposes the availability of a resource, which is made available by a dependee agent.

For each type of dependency, we provide operators for three *grades of strength* on the depender side: *general*, *committed,* and *critical.* These indicate the degree to which the agent will be affected if what the agent depends on fails. On the dependee side, the two grades of strength *general* and *committed* indicate the effort that the agent will put in to meet the dependency.

Each dependency operator relates an agent to an *object.* We follow RML [13] in distinguishing among three basic types of objects: *entities*, *activities*, and *assertions.* A goal-dependency operator relates an agent to an assertion. A task-dependency operator relates an agent to an activity (or task). A resource-dependency operator relates an agent to an entity.

A fourth type of dependency, a hybrid of goal and task dependency, is offered for expressing the *satisficing* of a goal in terms of constraints on *how* the goal is to be accomplished.

We treat agent dependency operators as extensions to the assertional sublanguage of the conceptual modelling language Telos [17]. Figure 1 is a graphical summary of the proposed set of agent dependency operators.

2.1 Goal dependency

The **Want** operator $W(a, \phi)$ means that agent a would like to have the condition described by assertion ϕ achieved by some agent. Suppose Chuck is in charge of an engineering development program which contracts out projects to various teams. Chuck would like to have proposals from all teams which are qualified for each project.

$$(\forall \mathsf{p/Project})(\forall \mathsf{team/Team})((\mathsf{qualified(team,p)}) \supset \mathsf{W(chuck,hasProposal(chuck,p,team))})$$

In this general form of goal dependency, the achievement of ϕ is usually not essential to the depender agent. In the example, it is not essential to Chuck that he has proposals from all teams.

The **Able** operator $A(a, \phi)$ means agent a is able to achieve ϕ for some other agent. To say that manager Mary has the ability to provide proposals to potential clients on all projects for which her team qualifies, we write:

$$(\forall \mathsf{c/Client})(\forall \mathsf{p/Project})(\mathsf{qualified(maryTeam,p)} \supset \mathsf{A(mary,hasProposal(c,p,maryTeam))})$$

The *Able* operator is used to convey that the agent has the knowhow to achieve a goal. When that knowhow is exercised, there can be many factors which stand in the way of the achievement of the goal, e.g. the unavailability of resources.

The **Committed-Want** operator $CW(a, b, \phi)$ means that agent a wants ϕ to be true and has a commitment from agent b that b will make ϕ true. The **Committed-Able** operator $CA(a, b, \phi)$ means that a is able to make ϕ true and is committed to b on making ϕ true. Commitment implies that each side believes that there is a high degree of certainty that the condition ϕ will hold. The statement:

$$(\forall \mathsf{p/Project})(\forall \mathsf{ph/ProjectHolder})(\mathsf{contractedOut(p)} \supset \mathsf{CW(chuck,ph,delivered(p))})$$

says that for all projects that have been contracted out, Chuck has a commitment from all project holders that they will deliver their projects. Mary, as a dependee, has a commitment to deliver all the projects that she manages:

$(\forall \mathsf{p/Project})(\forall \mathsf{cl/Client})(\mathsf{manages(mary,p)} \supset$
$\quad \mathsf{CA(mary,cl,delivered(p)))}$

A committed dependency has different implications on each side. On the dependee side, the dependee agent will try its best to achieve ϕ. The depender, assuming that ϕ will hold, may stop exploring or maintaining other channels for achieving ϕ, thus freeing itself for other activities.

The **Critical-Want** operator $XW(a, \phi, \phi_0)$ means that agent a critically depends on ϕ being true in order that the (overall) goal ϕ_0 may succeed. For instance, suppose that all the projects that Chuck contract out are components of larger development programs. In each program, Chuck has defined key projects without which the program cannot be completed.

$(\forall \mathsf{p/Project})(\forall \mathsf{prog/Program})(\mathsf{keyProject(p,prog)} \supset$
$\quad \mathsf{XW(chuck,delivered(p),completed(prog))}$

The analogue of *Critical-Want* on the dependee side would be a guarantee of success. Unfortunately, in a realistic, open environment, outcomes can rarely be guaranteed. We omit the operator for this case.

2.2 Task dependency

The **Run** operator $R(a, \pi)$ means that agent a would like to have the task π performed by some other agent, or in short, to "run" the task π (as in running a project, or running someone's life). A task description specifies a set of subgoals and/or subtasks to be met or carried out, usually accompanied by constraints such as temporal ordering. To say that manager Mary runs all the work packages in all projects that she manages, we write:

$(\forall \mathsf{p/Project})(\forall \mathsf{wp/WorkPackage})$
$\quad (\mathsf{manages(mary,p)} \land \mathsf{wp} \in \mathsf{p} \supset \mathsf{R(mary,wp)})$

In the context of project management, this usually means that each work package is divided into a number of subtasks (parts, steps), and subgoals (intermediate and final deliverables), with checkpoints and milestones.

A task description does not specify the goal for which the task is designed. The depender does not want the dependee to perform the task any other way, even if there are other ways to arrive at the same goal. In our example, Mary's goal is to have each project succeed. By using task dependency, she constrains how each work package will be done, based on her own knowhow about how to make a project succeed. She only depends on her team members for the *performance* of the tasks as she defines them. By using task dependency instead of goal dependency, a depender can exercise control over *how* things are done, but limits the degree to which the performer can use their own knowhow to adapt to unanticipated situations.

The **Perform** operator $P(a, \pi)$ means that agent a can perform task π. To say David can perform all circuit design tasks which require an experience level up to his, we write:

$(\forall \mathsf{wp/WorkPackage})$
$\quad ((\mathsf{experienceLevelReqd(wp)} \leq \mathsf{experienceLevel(david)}$
$\qquad \land\ \mathsf{wp.type{=}circuitDesign}) \supset \mathsf{P(david,wp)})$

Although a task specification constrains how the task is to be carried out, an agent performing the task still needs to have knowhow for addressing the unconstrained parts of the task, such as meeting subgoals that may arise in further task decomposition.

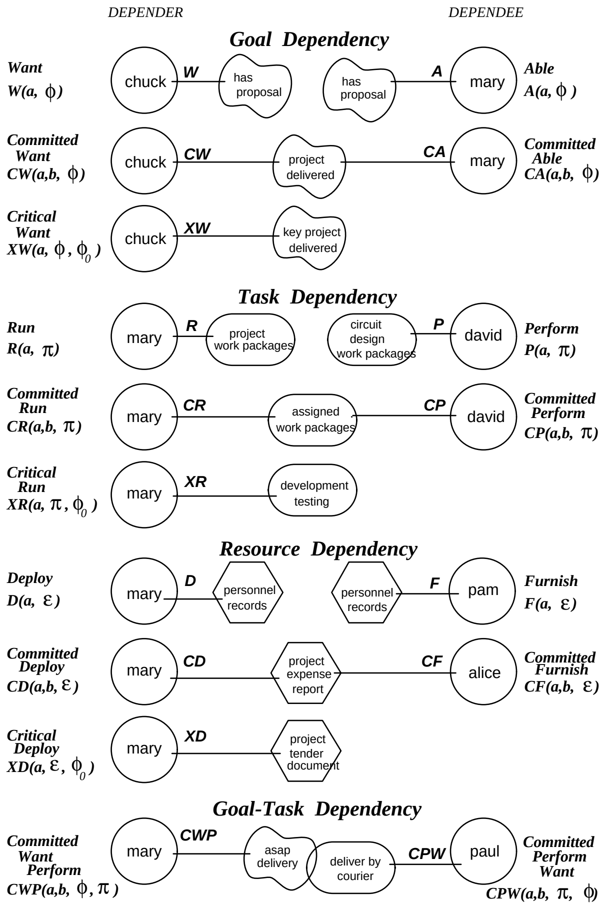

Figure 1. Agent Dependency Operators with Examples

The **Committed-Run** operator $CR(a, b, \pi)$ means that agent a would like to have the task π performed, and that agent b is committed to performing that task (or in short, a is committed to running π on b). The **Committed-Perform** operator $CP(a, b, \pi)$ means agent a is committed to performing π for agent b. For example,

(∀wp/WorkPackage)(∀m/Member)
    (assigned(mary,m,wp) ⊃ CR(mary,m,wp))

says that Mary runs all work packages that has been assigned to members with commitments from them.

Task dependency is often useful for describing procedures and policies, which are intended to curtail the freedom of the performer. For instance, as a matter of policy, Mary wants her team members to report their progress weekly.

(∀p/Project)(∀m/Member)
    ((manages(mary,p) ∧ worksOn(m,p)) ⊃
        CR(mary,m,reportProgressWeekly(m,mary,p)))

The **Critical-Run** operator $XR(a, \pi, \phi_0)$ means that agent a critically depends on the running of ϕ to achieve the goal ϕ_0. Suppose that no development project can be delivered without doing the testing step. We write:

(∀p/Project)(manages(mary,p) ∧ devProject(p) ⊃
        XR(mary,p.testing,delivered(p)))

2.3 Resource Dependency

The **Deploy** operator $D(a, \epsilon)$ means that agent a may presuppose that the entity ϵ is available. For instance, Mary may deploy personnel records for members working on each project.

(∀p/Project)(∀m/Member)(manages(mary,p)∧
        worksOn(m,p) ⊃ D(mary,perRecord(m)))

The **Furnish** operator $F(a, \epsilon)$ means agent a is able to make entity ϵ available. Pam in personnel is able to furnish personnel records of all members.

(∀m/Member)(F(pam,perRecord(m)))

A resource entity can be a physical object, or a piece of information, such as the price of an item to be purchased. It can also be a physical object carrying information, such as an invoice. Since the availability of agents is important for many types of situations, agents themselves constitute an important class of resource.

The **Committed-Deploy** operator $CD(a, b, \epsilon)$ means that agent a presupposes the availability of the resource ϵ, and that it believes agent b is committed to making it available. The **Committed-Furnish** operator $CF(a, b, \epsilon)$ means agent a is committed to making resource entity ϵ available for agent b.

Mary has committed-deploy dependencies on the expense reports on all her projects.

(∀p/Project)(manages(mary,p)⊃CD(mary,expenseReport(p)))

Alice is committed to furnishing expense reports for all projects.

(∀p/Project)(CF(alice,expenseReport(p)))

A common pattern of dependencies is that, if a depender seeks a resource, it will also have a goal or task dependency on an agent who can furnish the resource. For instance, Mary and her client Chuck have the understanding that a commitment to have a project delivered implies a commitment to make the deliverables available at the end of the project.

(∀p/Project)(∀cl/Client)(∀ph/ProjectHolder)
    (CA(ph,cl,delivered(p)) ⊃ CF(ph,cl,p.deliverables))

2.4 Goal-task dependency

There are certain types of goals whose satisfaction may be subject to various interpretation. For example, when a goal involves predicates such as accurate, cheap, as-soon-as-possible, or reasonable-return-on-investment, it is difficult for depender and dependee to agree on what it means for the goal to be satisfied. Very often, this is resolved by identifying a *way* (a procedure, a set of methods, a process) to achieve the goal which the depender believes will *adequately* address the goal [18]. We say that task description π *satisfices* goal ϕ for the depender agent.

To model this, we use a hybrid type of dependency. The **Committed-Want-Perform** operator $CWP(a, b, \phi, \pi)$ means that agent a has a commited-want on ϕ to be true, and believes that agent b is committed to performing π, and agent a is satisfied that π meets the goal ϕ adequately. From the dependee's viewpoint, the **Committed-Perform-Want** operator $CPW(a, b, \pi, \phi)$ means that agent a is commited to performing π, and believes that agent b has a commited-want on ϕ to be true, and agent a believes that π satisfices ϕ for agent b.

Suppose Mary wants to have an item delivered as soon as possible. After exploring various options with Paul the purchasing specialist, Mary decides that delivery by courier would be "soon enough".

CWP(mary,paul,asap(delivered(item)),
        deliverByCourier(item))

2.5 Agents, roles, and positions

In an organization, dependencies are usually not tied to a particular agent, but rather to a *role*. For instance, Mary the manager plays a number of roles, such as controlling expenses, purchasing equipment, and managing projects. Each role has a set of associated dependencies. We call a set of roles that are played by one agent a *position*. When Mary's position is vacated and filled by Molly, Mary's former roles and dependencies become associated with Molly.

Figure 2 shows a graphical representation of a sampling of the dependencies in a project organization.

Agent dependency relationships are shown to hold between roles. A role may deploy an agent as one of its resources.

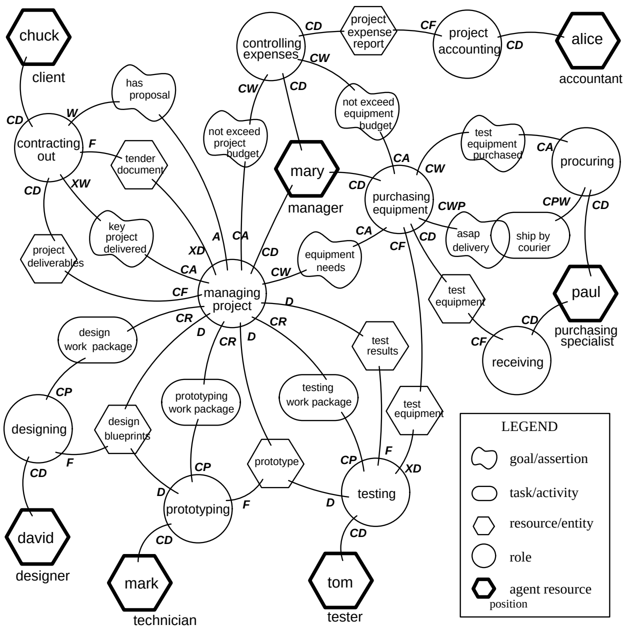

Figure 2. A Sampling of Agent Dependencies in an Engineering Project Organization

The type of dependency that exists between two roles is a feature of the particular organization being modelled. For example, the purchasing equipment role of manager Mary is related to the procuring role of Paul via a goal dependency. This is because when Mary wants to have some item purchased, she does not want to stipulate how the purchasing staff should do their job. For items that she wants delivered ASAP, she specifies "ship-by-courier" through a goal-task dependency.

Mary exercises more control over how her team members perform project tasks by specifying activity steps in their work packages. We model the relationship between Mary's managing project role and the team members work roles as task dependencies. Note that an organization model describes the particular way an organization is organized. It is not the only way to organize.

3 Formalizing agent dependency operators

Having introduced agent dependency operators informally through examples, we now provide a more precise characterization of these operators by relating them to the plans that agents use to achieve goals.

Our notion of a plan is based on an extended notion of an RML activity. An RML activity consumes input entities, produces output entities, and may also use control entities (similar to SADT [20]). Its effects are described by pre- and post-conditions. Additional constraints, such as temporal ordering, start and stop conditions, may also be specified. As in SADT, an important structuring dimension is the decomposition of activities into subparts, forming a hierarchy. We extend the RML notion of activity by allowing an activity to be associated with a goal, and for the decomposition of an activity to contain subgoals as well as activity subparts (called subtasks) [5]. Activities are thus linked by means-ends relationships, as well as by other structuring dimensions. It is this notion of activity that we have used for task specification in a task dependency.

A *plan* is a set of activities for addressing a goal. A plan is modelled in a similar way as in the modelling of RML, SADT-like activities, with the additional feature that activities in a plan may be related by goal-subgoal relationships. We also allow each subgoal, subtask, input and control attribute to be declared *external* to indicate that the plan assumes that some external agent will supply these items.

Note that we are *modelling* the plans of agents, i.e. writing down what an agent thinks it will do under what circumstances, rather than generating plans for agents using an automated planner. Usually agent's plans are open-ended (incomplete and underconstrained), and may contain only skeletal elements and constraints, with the agent believing that it will be able to fill in the details during plan execution. We assume that each agent has a repertoire of plans for achieving goals of interest to it, although not all plans will be represented explicitly in a requirements model.

We use the predicate $occur(p)$ to mean that the plan p is developed into a course of action and executed. Predicate $result(p, \phi)$ means that ϕ is true at the end of executing plan p. This assumes that all the external dependencies of p hold. $fail(a, p, \phi_0)$ means that agent a will not achieve ϕ_0 by executing p, taking all factors that affect a into account.

The belief operator $B(a, \phi)$ expresses that agent a believes that ϕ is true. It provides an opaque, intentional context for the beliefs of agents. What an agent believes need not be true, and an agent need not believe what other agents believe. A common way to axiomatize belief is as follows (corresponding to a "Weak S5" modal logic): [4]

- b1: $B(a, \phi) \wedge B(a, \phi \supset \psi) \supset B(a, \psi)$

- b2: $B(a,\phi) \supset B(a, B(a,\phi))$
- b3: $\neg B(a,\phi) \supset B(a, \neg B(a,\phi))$
- b4: $B(a,\phi) \supset \neg B(a, \neg\phi)$

We use a belief operator instead of a knowledge operator because what is captured in a requirements model is only what the agent believes it will do, which may differ from what it will actually do. Its abilities are only claims.

In the following, we provide a preliminary axiomatic characterization of the dependency operators.

We interpret $W(a,\phi)$ to mean that a believes that it has some plan p for achieving some goal ϕ_0 which contains an activity α which has ϕ as an external subgoal. The interpretation for *Run* and *Deploy* are similar, with π being an external subtask and ϵ an external input (or control) respectively.

- w1: $W(a,\phi) \supset B(a, \exists p \exists \phi_0 \exists \alpha(\alpha \in p \wedge extSubgoal(\phi,\alpha) \wedge result(p,\phi_0)))$
- r1: $R(a,\pi) \supset B(a, \exists p \exists \phi_0 \exists \alpha(\alpha \in p \wedge extSubtask(\pi,\alpha) \wedge result(p,\phi_0)))$
- d1: $D(a,\epsilon) \supset B(a, \exists p \exists \phi_0 \exists \alpha(\alpha \in p \wedge (extInput(\epsilon,\alpha) \vee extControl(\epsilon,\alpha)) \wedge result(p,\phi_0)))$

An agent a is *Able* to achieve ϕ if it believes it has some plan which will result in ϕ being true. An agent can perform plan/activity π if π is in its repertoire. An agent can furnish resource ϵ if it has a plan which results in ϵ being available.

- a1: $A(a,\phi) \supset B(a, \exists p\ result(p,\phi))$
- p1: $P(a,\pi) \supset B(a, \exists p\ (p = \pi))$
- f1: $F(a,\epsilon) \supset B(a, \exists p\ result(p, avail(\epsilon)))$

We define each inter-agent operator in terms of operators which only has local implications for each agent. For example, a has a *Committed-Want* relationship with b on ϕ if a itself has a *Committed-Want* on ϕ, and a believes that b has a *Committed-Able* on ϕ.

- cw1: $CW(a,b,\phi) \equiv CW(a,\phi) \wedge B(a, CA(b,\phi))$
- cr1: $CR(a,b,\pi) \equiv CR(a,\pi) \wedge B(a, CP(b,\pi))$
- cd1: $CD(a,b,\epsilon) \equiv CD(a,\epsilon) \wedge B(a, CF(b,\epsilon))$

The counterparts on the dependee side are:

- ca1: $CA(a,b,\phi) \equiv CA(a,\phi) \wedge B(a, CW(b,\phi))$
- cp1: $CP(a,b,\pi) \equiv CP(a,\pi) \wedge B(a, CR(b,\pi))$
- cf1: $CF(a,b,\epsilon) \equiv CF(a,\epsilon) \wedge B(a, CD(b,\epsilon))$

A *Committed-Want* means that a has some plan p which will fail if ϕ is not true. A *Committed-Run* means that a has some plan p which will fail if π does not occur. A *Committed-Deploy* means that a has some plan p which will fail if ϵ is not available.

- cw2: $CW(a,\phi) \supset B(a, \exists p \exists \phi_0(\neg\phi \supset fail(a,p,\phi_0)))$
- cr2: $CR(a,\pi) \supset B(a, \exists p \exists \phi_0(\neg occur(\pi) \supset fail(a,p,\phi_0)))$
- cd2: $CD(a,\epsilon) \supset B(a, \exists p \exists \phi_0(\neg avail(\epsilon) \supset fail(a,p,\phi_0)))$

Commitment on the dependee side means that the dependee believes that the commitment will be met. We take this to mean that the agent believes that it has a plan which is able to meet the commitment, and that everything that the plan depends on are OK.

- ca2: $CA(a,\phi) \supset B(a, \exists p(result(p,\phi) \wedge allDepOK(a,p)))$
- cp2: $CP(a,\pi) \supset B(a, \exists p(p = \pi) \wedge allDepOK(a,p)))$
- cf2: $CF(a,\epsilon) \supset B(a, \exists p(result(p, avail(\epsilon)) \wedge allDepOK(a,p)))$

We define $allDepOK(a,p)$ to mean that agent a has a plan p for which it has secured commitments on all its dependencies (for which it is depender), and that it believes that all its critical dependencies will hold. For this definition, we need versions of dependency operators which are parameterized on a plan p of a. $W(a,\phi,p)$ means a wants ϕ for plan p, so that $W(a,\phi) \equiv \exists p W(a,\phi,p)$.

$$
\begin{aligned}
allDepOK(a,p) =_{def}\ & \\
& \forall \phi_i(W(a,\phi_i,p) \supset \exists b CW(a,b,\phi_i)) \wedge \\
& \forall \pi_i(R(a,\pi_i,p) \supset \exists b CR(a,b,\pi_i)) \wedge \\
& \forall \epsilon_i(D(a,\epsilon_i,p) \supset \exists b CD(a,b,\epsilon_i)) \wedge \\
& \forall \phi_i(XW(a,\phi_i,\phi_0) \supset B(a,\phi_i)) \wedge \\
& \forall \pi_i(XR(a,\pi_i,\phi_0) \supset B(a, occur(\pi_i)) \wedge \\
& \forall \epsilon_i(XD(a,\epsilon_i,\phi_0) \supset B(a, avail(\epsilon_i))
\end{aligned}
$$

We interpret critical dependency to mean that agent a believes that all the plans that it has will fail to achieve goal ϕ_0 if ϕ is false (XW), if π does not occur (XR), or if ϵ is not available (XD).

- xw1: $XW(a,\phi,\phi_0) \supset B(a, (\neg\phi \supset \forall p fail(a,p,\phi_0)))$
- xr1: $XR(a,\pi,\phi_0) \supset B(a, (\neg occur(\pi) \supset \forall p fail(a,p,\phi_0)))$
- xd1: $XD(a,\epsilon,\phi_0) \supset B(a, (\neg avail(\epsilon) \supset \forall p fail(a,p,\phi_0)))$

Goal-Task dependency is defined by introducing a predicate $satisficed(a,\pi,\phi)$, which says that agent a is sufficiently satisfied that π meets the goal ϕ.

- cwp1: $CWP(a,b,\phi,\pi) \equiv CW(a,\phi) \wedge B(a, CP(b,\pi)) \wedge satisficed(a,\pi,\phi)$
- cpw1: $CPW(a,b,\pi,\phi) \equiv CP(a,\pi) \wedge B(a, CW(b,\phi)) \wedge B(a, satisficed(b,\pi,\phi))$

If an agent's plans are explicitly represented in a requirements model, its dependencies may be deduced from its plans. For example, we can deduce that an agent is able to achieve ϕ if we can examine its repertoire of plans, and determine that it indeed believes of some plan which has the desired result. We only list the axioms for goal dependency. The others are analogous.

- w2: $W(a,\phi) \subset \exists p \exists \phi_0 \exists \alpha B(a, \alpha \in p \wedge extSubgoal(\phi,\alpha) \wedge result(p,\phi_0))$
- cw3: $CW(a,\phi,p) \subset \exists \phi_0 B(a, (\neg\phi \supset fail(a,p,\phi_0)))$
- xw2: $XW(a,\phi,\phi_0) \subset \forall p B(a, (\neg\phi \supset fail(a,p,\phi_0)))$
- a2: $A(a,\phi) \subset \exists p B(a, result(p,\phi))$
- ca3: $CA(a,\phi) \subset \exists p B(a, result(p,\phi) \wedge allDepOK(a,p))$

We also expect a number of relationships among dependency operators to hold in the model. All the committed dependencies should imply the general versions, e.g.: $CA(a,b,\phi) \supset A(a,\phi)$; $CW(a,b,\phi) \supset W(a,\phi)$. All the critical ones should also imply the general ones, e.g. $XW(a,\phi,\phi_0) \supset W(a,\phi)$. The critical should imply the committed. $XW(a,\phi,\phi_0) \supset \exists b\ \ CW(a,b,\phi)$. The inter-agent committed dependencies should be reciprocated: $CA(a,b,\phi) \supset CW(b,a,\phi)$; $CA(a,b,\phi) \subset CW(b,a,\phi)$.

4 Discussion

In this section, we outline a usage context in which the proposed operators might be applied and discuss related work.

During the requirements process, analysts try to obtain an understanding of the environment, while organizational agents try to express their needs and concerns. The dependency relationships identified in the above sections may be seen as one way to capture some of these needs and concerns.

In our example, Chuck's critical dependency on key projects suggests why he might be concerned about the *accuracy* of the information he receives about these projects. Mary's commited dependency on her project members suggests a need for a record of the tasks that she is "running", and whether they have been performed. The particular network of dependencies in an organization shows *where* the agents' concerns lie. In our example, project members are commited to manager Mary, and Mary is commited to client Chuck, but project members are not directly committed to the client. Given this pattern of dependencies, one would not conclude that Chuck would be concerned about the progress of individual project members.

The operators we have adopted aim to accommodate an autonomous, yet fallible, notion of agent. Organizational agents have a propensity to search for new alternatives when sufficiently dissatisfied [15]. Thus we assume that agents would tend to perceive technical systems as enhancing or restricting their repertoire of plans for pursuing their goals, rather than as dictating a unique course of action. By modelling and analyzing the network of dependencies, analyst and agent may come to a better understanding of the impact that different technical options might have on each agent. Being part of a requirements model, agent dependencies would also be used for guiding the design process and for justifying design decisions [3, 18].

The use of a belief operator allows agents to have differing viewpoints as well as false beliefs. The commitment operators embody a weak form of prescription–obligation relationship (in comparison to, for example, deontic logic [9, 10]), acknowledging that agents may fail to live up to their obligations. Even when an agent is committed and responsible, it may still fail to meet the commitment due to uncertainties in the environment. The sources of failure are partially captured in the agent's dependencies.

As with world modelling in general, knowledge about agent dependencies is usually incomplete. A requirements model only presents a partial picture of how agents might behave, reflecting an "open world" view of organizational environments [16]. As the development process proceeds from requirements to design, stronger assumptions such as closed world assumptions would need to be introduced.

In comparison, the Composite System Design approach treats hardware, software, and human agents as a single *closed* system to be designed. Starting with global system goals, subgoals are derived and *assigned* to agents, which are treated as *components* of the overall system [7]. [6] presents a goal-directed strategy for requirements acquisition.

Our agent modelling concepts are adaptations of those in Artificial Intelligence [4, 21, 14] to reflect organization modelling needs. For example, we separately consider the implications of a commitment on each side of a dependency. One work that specifically addresses agent dependencies is [2]. Its treatment includes a notion of resource dependency, patterns of dependencies, and a notion of influencing, but not commitment or criticality. It argues, however, that dependencies are not fundamentally mental, in that they hold independently of agents' awareness of them. In our approach, we capture agents' *beliefs* about their environments, since that is what they base their decisions on, even if they may turn out to be wrong.

Given a formal characterization of the dependency operators, we anticipate that verification techniques can be applied to a model – to verify, for example, that agents' beliefs are consistent, and that the basic relationships among dependencies hold. Further properties may be asserted as constraints on elements in the model, e.g., the viability of an agent's plans, or the fulfilment of a commitment between two agents. These constraints may be viewed as invariants that safeguard the integrity of the model. When changes are made to a model to explore technical or organizational alternatives, the violation of constraints suggests potential issues for discussion and negotiation. Verification may be limited by incompleteness in the model. For instance, one can prove that a plan will fail if a resource that has been identified as critical is unavailable. However, to prove that a plan will succeed, one would need to introduce stronger assumptions.

5 Conclusions

In modelling an organizational environment for the purpose of requirements engineering, capturing the intentional relationships among agents that can help answer "why" questions can be a useful addition to conventional models which have focused mainly on capturing the "whats".

In this paper, we have identified a set of operators which we believe can serve as a basis for describing the intentional structure of an organizational environment. We have illustrated these operators with examples and have provided a preliminary set of axioms. The extent to which this set of operators can capture complex issues in social settings has yet to be tested against more elaborate example settings.

In future work, we plan to integrate these operators into a rich conceptual modelling language, reconciling them with knowledge structuring dimensions such as classification, generalization, aggregation, and time. As well, we need to provide a formal semantics for the representational features, and to identify special forms of reasoning, and produce tractable algorithms.

Acknowledgements

Sincere thanks are due to Professor John Mylopoulos for his encouragement and guidance, and to Lawrence Chung, Yves Lesperance, Brian Nixon, Sol Greenspan, and the anonymous referees for their many useful comments and suggestions.

References

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

[2] 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.

[3] 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.

[4] P. R. Cohen and H. J. Levesque, Intention is Choice with Commitment, *Artif. Intell.,* 42 (3), 1990.

[5] W. B. Croft and L. S. Lefkowitz, A Goal-Based Representation of Office Work, *Office Knowledge: Representation, Management, and Utilization,* W. Lamersdorf (ed.), Elsevier, pp. 99-124, 1988.

[6] A. Dardenne, S. Fickas, and A. van Lamsweerde, Goal-directed Concept Acquisition in Requirements Elicitation, *Proc. 6th Intl. Workshop on Software Spec. and Design*, pp. 14-21, Como, Italy, Oct. 25-26, 1991.

[7] E. Doerry, S. Fickas, R. Helm, and M. Feather, A Model for Composite System Design, *Proc. 6th Intl. Workshop on Software Spec. and Design*, pp. 216-219, Como, Italy, Oct. 25-26, 1991.

[8] 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.

[9] E. Dubois, A Formal Language for the Requirements Engineering of Composite Systems, *Working Notes, AAAI Spring Symp.: Design of Composite Systems*, pp. 11-29, Stanford Univ., Mar. 26-28, 1991.

[10] A. Finkelstein and C. Potts, Building Formal Specifications Using "Structured Common Sense", *Proc. 4th Intl. Workshop on Software Spec. and Design*, pp. 108-113, Monterey, California, Apr. 3-4, 1987.

[11] L. Gasser, Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics, *Artif. Intell.,* 47, pp. 107-138, Jan. 1991.

[12] S. J. Greenspan, J. Mylopoulos, and A. Borgida, Capturing More World Knowledge in the Requirements Specification, *Proc. 6th Intl. Conf. Soft. Eng.*, pp. 225-234, Tokyo, Japan, 1982.

[13] S. J. Greenspan, *Requirements Modelling: A Knowledge Representation Approach to Software Requirements Definition,* Ph. D. Thesis, Dept. of Comp. Sci., Univ. of Toronto, 1984.

[14] Y. Lesperance, *A Formal Theory of Indexical Knowledge and Action,* Tech. Rept. CSRI-248, Comp. Sys. Res. Inst., Univ. of Toronto, Feb. 1991.

[15] J. G. March and H. A. Simon, *Organizations*, 1958.

[16] J. Mylopoulos, Representing Knowledge About Information Systems, *Intl. Workshop on Development of Intelligent Information Systems*, Niagara-on-the-Lake, Ontario, Canada, pp. 94-96, April 21-23, 1991.

[17] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis, Telos: Representing Knowledge about Information Systems, *ACM Trans. Info. Sys.*, 8 (4), 1991.

[18] J. Mylopoulos, L. Chung, B. Nixon, Representing and Using Non-Functional Requirements: A Process-Oriented Approach, *IEEE Trans. Soft. Eng.*, 18 (6), June 1992.

[19] A. Newell, The Knowledge Level, *Artif. Intell.*, vol. 18, pp. 87–127, 1982.

[20] D. T. Ross and K. E. Shoman, Structured Analysis for Requirements Definition, *IEEE Trans. Soft. Eng.*, Vol. SE-3, No. 1, Jan. 1977.

[21] B. Thomas, Y. Shoham, A. Schwartz, and S. Kraus, Preliminary Thoughts on an Agent Description Language, *Intl. J. Intell. Sys.*, Vol. 6, pp. 498-508, 1991.

[22] P. Zave and R. T. Yeh, Executable Requirements for Embedded Systems, *Proc. 5th Intl. Conf. Soft. Eng.*, pp. 295-304, 1981.