An Organization Ontology for Enterprise Modelling

Mark S. Fox, Mihai Barbuceanu, Michael Gruninger, and Jinxin Lin

Enterprise Integration Laboratory
Department of Mechanical and Industrial Engineering, University of Toronto
4 Taddle Creek Road, Toronto, Ontario, CANADA M5S 3G9
tel:+1-416-978-6823 *fax:*+1-416-971-2479 *http:*//www.eil.utoronto.ca/EIL/

Appeared in: *Simulating Organizations: Computational Models of Institutions and Groups*, M. Prietula, K. Carley & L. Gasser (Eds), Menlo Park CA: AAAI/MIT Press, pp. 131-152.

Abstract

The paper presents our exploration into an organization ontology for the TOVE enterprise model. Its primary focus has been in linking structure and behavior through the concept of empowerment. Empowerment is the right of an organization agent to perform status changing actions. This linkage is critical to the unification of enterprise models and their executability.

1.0 Introduction

What is an organization and how do we model it in an information system? Many disciplines have explored the former and every information system built has created a version of the latter. The purpose of this paper is to explore the latter from the perspective of Artificial Intelligence.

As information systems play a more active role in the management and operations of an enterprise, the demands on these systems have also increased. Departing from their traditional role as simple repositories of data, information systems must now provide more sophisticated support to manual and automated decision making; they must not only answer queries with what is explicitly represented in their Enterprise Model, but must be able to answer queries with what is *implied* by the model. The goal of the TOVE Enterprise Modelling project is to create the next generation Enterprise Model, a *Common Sense Enterprise Model*. By common sense we mean that an Enterprise

Model has the ability to deduce answers to queries that require relatively shallow knowledge of the domain.

We are taking what can be viewed as a “second generation knowledge engineering” approach to constructing our Common Sense Enterprise Model. Rather than extracting rules from experts, we are “engineering ontologies.” An ontology is a formal description of entities and their properties, relationships, constraints, behaviors. Our approach to engineering ontologies begins with defining an ontology’s requirements; this is in the form of questions that an ontology must be able to answer. We call this the *competency* of the ontology. The second step is to define the terminology of the ontology - its objects, attributes, and relations. The third step is to specify the definitions and constraints on the terminology, where possible. The specifications are represented in First Order Logic and implemented in Prolog. Lastly, we test the competency of the ontology by “proving” the competency questions with the Prolog axioms.

Our initial efforts have focused on ontologies to support reasoning in industrial environments. The tasks that we have targeted to support are in “supply chain management” which extends MRP (Manufacturing Requirements Planning) to include logistics/distribution and “Concurrent Engineering” which looks at issues of coordination of engineering design. Much of our effort has been in creating representations of organization behavior: activity, state, causality and time, and the objects they manipulate: resources [Fadel 94, Fadel et al. 94], inventory, orders and products. We also have efforts underway in formalizing knowledge of ISO 9000 quality [Kim & Fox 93], activity-based costing [Tham et al. 94] and organization agility.

This paper describes the *organization* ontology being developed as part of the TOVE Project. In particular it focuses on organization structure, roles, authority and empowerment.

2.0 What is an Organization?

We consider an organization to be a set of constraints on the activities performed by agents. This view follows that of Weber [87] who views the process of bureaucratization as a shift from management based on self-interest and personalities to one based on rules and procedures.

Mintzberg [1983] provides an early (and informal) analysis of organization structure distinguishing among five basic parts of an organization and five distinct organization configurations that are encountered in practice. This “ontology” includes several mechanisms that together achieve coordination, like goals, work processes, authority, positions and communication. The various parts of an organization are distinguished by the specific roles they play in achieving coordination with the above means.

The “language/action perspective” [Winograd 1987] on cooperative work in organizations provides an ontology that emphasizes the social activity by which “agents” generate the space of cooperative actions in which they work, rather than the mental state of individuals. The basic idea is that social

activity is carried out by language and communication. The pragmatic nature of communication as the way of creating commitments among participants is exploited in the Coordinator system [Flores et al. 1988].

In the same vein, [Auramaki et al. 1988] present a method for modeling offices as systems of communicative action through which people engage in actions by creating, modifying and deleting commitments that bind their current and future behaviors.

The work of Lee [1988] looks at language acts in the bureaucratic office, viewing language not as a mechanism for information transfer but as a mechanism for social interaction and control. He presents a logic-based representation of deontic notions - authorization, permission, prohibition and the like - and shows how this can be used to model cooperative work in the office.

More recently, Yu and Mylopoulos [1994] have proposed a framework for modeling organizations as being made of social actors that are intentional, having motivations, wants and beliefs and strategic, evaluating their opportunities and vulnerabilities with respect to each other. This formal model is used to explore alternative process designs in business reengineering.

3.0 Ontology Competence

A problem in the engineering of ontologies is their evaluation. A number of criteria have been proposed including [Fox et al. 93] [Gruber 93]:

- **Functional Completeness:** Can the ontology represent the information necessary to support some task?
- **Generality:** To what degree is the ontology shared between diverse activities such as engineering design and production, or design and marketing? Is the ontology specific to a sector, such as manufacturing, or applicable to other sectors, such as retailing, finance, etc.?
- **Efficiency:** Does the ontology support efficient reasoning, i.e., space and time, or does it require some type of transformation?
- **Perspicuity:** Is the ontology easily understood by the users so that it can be consistently applied and interpreted across the enterprise? Does the representation “document itself?”
- **Precision/Granularity:** Is there a core set of ontological primitives that are partitionable or do they overlap in meaning? Does the representation support reasoning at various levels of abstraction and detail?
- **Minimality:** Does the ontology contain the minimum number of objects (i.e., terms or vocabulary) necessary [Gruber 93]?

The criterion we have found most useful is *competence*. For any task in which the ontology is to be employed, the task imposes a set of requirements on the ontology. These requirements can best be

specified as a set of queries that the ontology should be able to answer, if it contains the relevant information. These requirements, which we call competency questions, are the basis for a rigorous characterization of the information that the ontology is able to provide to the task. Competency questions are benchmarks in the sense that the ontology is necessary and sufficient to represent the tasks specified by the competency questions and their solution. They are also those tasks for which the ontology finds all and only the correct solutions. Tasks such as these can serve to drive the development of new ontologies and also to justify and characterize the capabilities of existing ontologies.

This characterization of competency raises an important issue: where does the representation end and inference begin? If no inference capability is to be assumed, then query processing is reducible to "looking up" an answer that is represented explicitly. In contrast, object/semantic network representations assume at least inheritance as a deduction mechanism. In defining an ontology a key question then becomes: should we be restricted to just a terminology? Should the terminology assume an inheritance mechanism, or some type of theorem proving capability as provided, say, in a logic programming language with axioms restricted to Horne clauses (i.e., Prolog)? What is the *deductive capability* that is to be assumed by an ontology? In the TOVE project we assume a theorem prover of the power of Prolog.

The basic entities in the TOVE ontology are represented as objects with specific properties and relations. Objects are structured into taxonomies and the definitions of objects, attributes and relations are specified in first-order logic. An ontology is defined in the following way. We first identify the objects in our domain of discourse; these will be represented by constants and variables in our language. We then identify the properties of these objects and the relations that exist over these objects; these will be represented by predicates in our language.

We next define a set of axioms in first-order logic to represent the constraints over the objects and predicates in the ontology. This set of axioms provides a declarative specification for the various definitions and constraints on the terminology. Further, we need to prove the competency of the ontology. The ontology must contain a necessary and sufficient set of axioms to represent and solve these questions, thus providing a declarative semantics for the system. It is in this sense that we can claim to have a competent ontology, and it is this rigor that is lacking in previous approaches to ontology engineering.

The competency questions are generated by requiring that the ontologies be necessary and sufficient to support the various tasks in which it is employed. Within our applications of activity/time ontology, these include:

- Planning and scheduling -- what sequence of activities must be completed to achieve some goal? At what times must these activities be initiated and terminated?
- Temporal projection -- Given a set of actions that occur at different points in the future, what are the properties of resources and activities at arbitrary points in time? This includes the management of resources and activity-based costing (where we are assigning costs to resources and activities).

- Execution monitoring and external events -- What are the effects on the enterprise model of the occurrence of external and unexpected events (such as machine breakdown or the unavailability of resources)?
- Hypothetical reasoning -- what will happen if we move one task ahead of schedule and another task behind schedule? What are the effects on orders if we buy another machine?
- Time-based competition -- we want to design an enterprise that minimizes the cycle time for a product [Blackburn 91]. This is essentially the task of finding a minimum duration plan that minimizes action occurrence and maximizes concurrency of activities.

We propose the following set of competency questions for the organization ontology[1].

Structure Competency

- What role does an agent play?
- Which division does the agent belong to?
- Who must the agent communicate with?
- What kinds of information does the agent communicate?
- Who does the agent report to?
- Is a role a generalization of another role?

In linking the structure of an organization with the behavior of agents within the organization, we must define how the organization ontology is integrated with the activity ontology. If we consider an organization to be a set of constraints on the activities performed by agents, then the competency questions for the organization ontology are extensions of the temporal projection and plan existence problems to incorporate the abilities and obligations of agents. The temporal projection problem is used to characterize the constraints that agents must satisfy to be able to perform activities, and plan existence characterizes the set of achievable goals.

Behavior Competency

- What are the goals of the organization?
- What are the goals of a role?
- What are the goals of person X?
- What activities are available for a role to achieve its goal?
- What resources are available to achieve a goal?

Authority, Empowerment and Commitment Competency

1. We do not view these competency questions as being complete but indicative of what is needed.

- What resources does the person have authority to assign?
- In order to perform a particular activity, whose permission is needed?
- What activities may a person execute under their own authority?

4.0 Activity/Time Ontology

In this section we define the ontology of time and action that is used to represent the behavior of the organization. An important component of representing behavior is the ability to temporally project, that is, to determine the possible set of future states given a current state. Temporal projection induces the following set of requirements on the ontologies:

- Temporal projection requires the evaluation of the truth value of a proposition at some point in time in the future. We therefore need to define axioms that express how the truth of a proposition changes over time. In particular, we need to address the frame problem and express the properties and relations that change or do not change as the result of an activity.
- We must define the notion of a state of the world, that is, define what is true of the world before and after performing different activities. This is necessary to express the causal relationship between the preconditions and effects of an activity.
- The time interval over which the state has a certain status is bounded by the times at which the appropriate actions that change status occur. This interval defines the duration of a state if the status is enabled. This is essential for the construction of schedules.
- We want a uniform hierarchical representation for activities (aggregation). Plans and processes are constructed by combining activities. We must precisely define how activities are combined to form new ones. The representation of these combined activities should be the same as the representation of the subactivities. Thus aggregate activities (sets of activities or processes) should themselves be represented as activities.
- The causal and temporal structure of states and subactivities of an activity should be explicit in the representation of the activity.

4.1 Situation Calculus Specification

We represent time as a continuous line; on this line we define time points and time periods (intervals) as the domain of discourse. We define a relation < over time points with the intended interpretation that $t < t'$ iff t is earlier than t'.

One important property that must be represented to define what holds in the world after performing some action in order to capture the notion of causality. How do we express these notions if we have a continuous time line? The extended situation calculus of [Pinto & Reiter 93] allows us to incorporate the notions of situations and a time line by assigning durations to situations.

The intuition behind the situation calculus is that there is an initial situation, and that the world changes from one situation to another when actions are performed. There is a predicate Poss(a,σ) that is true whenever an action a can be performed in a situation σ.

The structure of situations is that of a tree; two different sequences of actions lead to different situations. Thus, each branch that starts in the initial situation can be understood as a hypothetical future. The tree structure of the situation calculus shows all possible ways in which the events in the world can unfold. Therefore, any arbitrary sequence of actions identifies a branch in the tree of situations.

Further, we impose a structure over situations that is isomorphic to the natural numbers by introducing the notion of successor situation [Reiter 91]. The function *do(a,σ)* is the name of situation that results from performing action *a* in situation σ. We also define an initial situation denoted by the constant *σ0*.

Situations are assigned different durations by defining the predicate *start(s,t)* [Pinto & Reiter 93]. Each situation has a unique start time; these times begin at 0 in *σ0* and increase monotonically away from the initial situation.

To define the evaluation of the truth value of a sentence at some point in time, we will use the predicate *holds(f,σ)* to represent the fact that some ground literal *f* is true in situation σ. Using the assignment of time to situations, we define the predicate *holdsT(f, t)* to represent the fact that some ground literal *f* is true at time *t*. A fluent is a predicate or function whose value may change with time.

Another important notion is that actions occur at points in time. The work of [Pinto & Reiter 93] extends the situation calculus by selecting one branch of the situation tree to describe the evolution of the world as it actually unfolds. This is done using the predicate actual. To represent occurrences, we then introduce two predicates, *occurs(a,σ)* and *occursT(a,t)*, defined as follows:

$$occurs(a,\sigma) \equiv actual(do(a,\sigma))$$

$$occursT(a,t) \equiv occurs(a,\sigma) \quad start(do(a, \sigma), t)$$

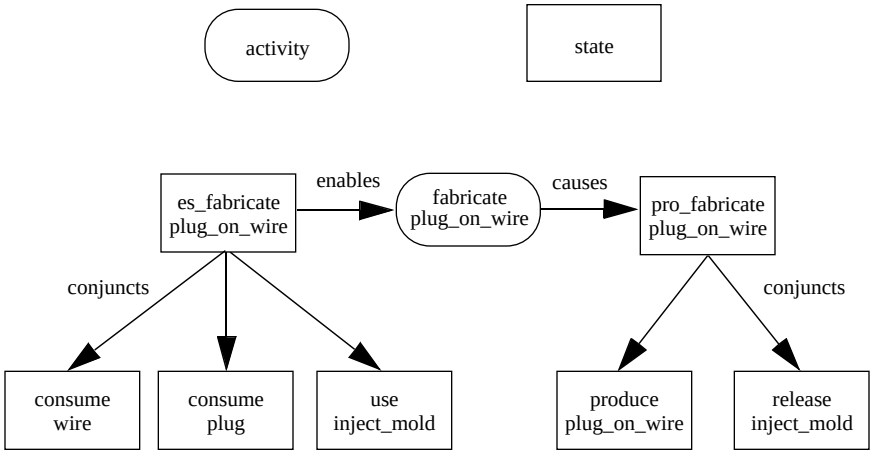
We will now apply this formalism to the representation of activities in an enterprise.

4.2 Activity/Time Terminology

At the heart of the TOVE Enterprise Model lies the representation of an *activity* and its corresponding enabling and caused *states* [Sathi et al. 85], [Fox et al 93]. In this section we examine the notion of states and define how properties of activities are defined in terms of these states. An activity is the basic transformational action primitive with which processes and operations can be represented; it specifies how the world is changed. An enabling state defines what has to be true of the world in order for the activity to be performed. A caused state defines what is true of the world once the activity has been completed.

An activity, along with its enabling and caused states, is called an *activity cluster.* The state tree linked by an *enables* relation to an activity specifies what has to be true in order for the activity to be performed. The state tree linked to an activity by a *causes* relation defines what is true of the world once the activity has been completed. Intermediate states of an activity can be defined by elaborating the aggregate activity into an activity network (see Figure 1).

FIGURE 1. Activity-State Cluster



There are two types of states: *terminal* and *non-terminal*. In Figure 1, *es_fabricate_plug_on_wire* is the nonterminal enabling state for the activity *fabricate_plug_on_wire* and *pro_fabricate_plug_on_wire* is the caused state for the activity. The terminal conjunct substates of *es_fabricate_plug_on_wire* are *consume_wire, consume_plug,* and *use_inject_mold* since all three resources must be present for the activity to occur; the terminal states of *pro_fabricate_plug_on_wire* are *produce_plug_on_wire* and *release_inject_mold.*

In TOVE there are four terminal states represented by the following predicates:*use(s,a), consume(s,a), release(s,a), produce(s,a).* These predicates relate the state with the resource required by the activity. Intuitively, a resource is used and released by an activity if none of the properties of a resource are changed when the activity is successfully terminated and the resource is released. A resource is consumed or produced if some property of the resource is changed after termination of the activity; this includes the existence and quantity of the resource, or some arbitrary property such as color. Thus *consume(s,a)* signifies that a resource is to be used up by the activity and will not exist once the activity is completed, and *produce(s,a)* signifies that a resource, that did not exist prior to the performance of the activity, has been created by the activity. We define use and consume states to be enabling states since the preconditions for activities refer to the properties of these states, while we

define release and produce states to be caused states, since their properties are the result of the activity.

Terminal states are also used to represent the amount of a resource that is required for a state to be enabled. For this purpose, the predicate *quantity(s,r,q)* is introduced, where *s* is a state, *r* is the associated resource, and *q* is the amount of resource r that is required. Thus if *s* is a consume state, then *q* is the amount of resource consumed by the activity, if *s* is a use state, then *q* is the amount of resource used by the activity, and if *s* is a produce state, then *q* is the amount of resource produced.

A state may have a status whose value is one of the following constants:{*possible, committed, enabled, completed, disenabled, reenabled*}. The status of a state is changed by one of the following actions:*commit(s,a), enable(s,a), complete(s,a), disenable(s,a), reenable(s,a).* Note that these actions are parametrized by the state and the associated activity.

Similarly, activities have a status whose value is one of the following constants: {dormant, executing, suspended, completed}. The status of an activity is changed by one of the following actions: *execute(a), suspend(a), complete(a).*

As part of our logical specification of the activity ontology, we define the successor axioms that specify how the above actions change the status of a state. These axioms provide a complete characterization of the value of a fluent after performing any action, so that we can use the solution to the frame problem in [Reiter 91]. Thus if we are given a set of action occurrences, we can solve the temporal projection problem (determining the value of a fluent at any point in time) by first finding the situation containing that time point, and then using the successor axioms to evaluate the status of the state in that situation. We present one of the successor axioms in the ontology:

The status of a state is committed in a situation iff either a commit action occurred in the preceding situation, or the state was already committed and an enable action did not occur.

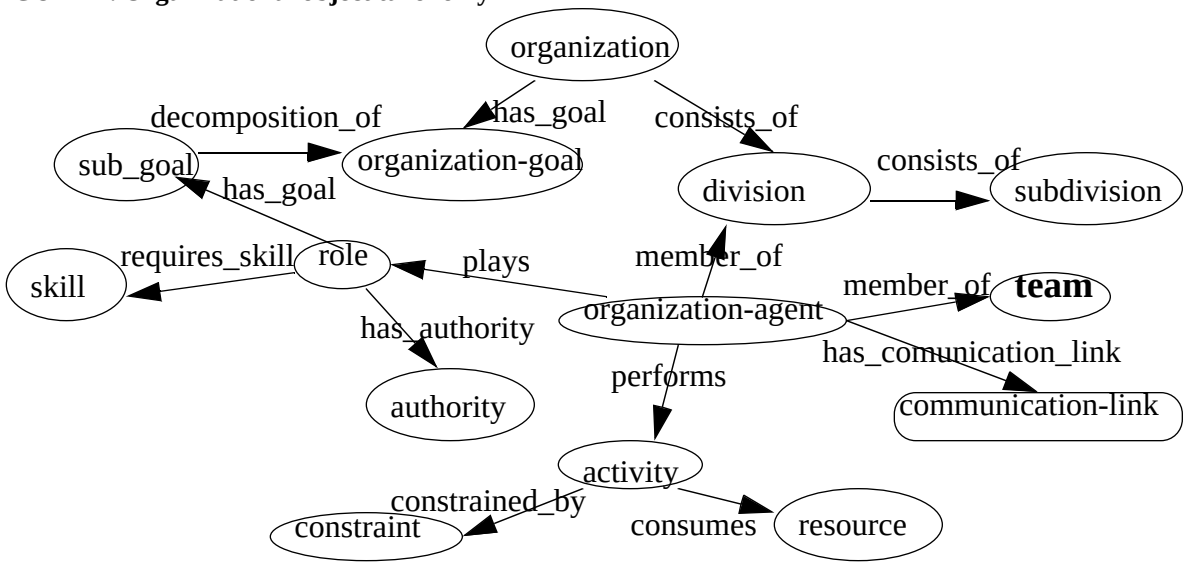
$$(\ s,a,e,\sigma)\ holds(status(s,a,committed), do(e,\sigma)) \equiv (e = commit(s,a) \quad holds(status(s,a,possible),\sigma)) \quad \neg(e=enable(s,a)) \quad holds(status(s,a,committed),\sigma)$$

A more complete specification can be found in [Gruninger & Fox 94].

5.0 Organization Terminology & Axioms

In this section we introduce the basic terminology, i.e., ground terms, and axioms of our organization ontology. Figure 1 shows the basic elements of our organization ontology.

FIGURE 2. Organizational object taxonomy



We consider an organization to be a set of constraints on the activities performed by agents. In particular, an *organization* consists of a set of *divisions* and *subdivisions* (recursive definition), a set of *organization-agents* (said to be members of a division of the organization), a set of *roles* that the members play in the organization, and an *organization-goal* tree that specifies the goals (and their decomposition into subgoals) the members try to achieve. For example, the Department of Industrial Engineering (IE) can be modeled as an organization having: a number of goals related e.g. to education and research, sub-divisions such as the Enterprise Integration Laboratory (EIL), the Human-Computer Interfaces Laboratory, etc., a number of organization agents consisting of individual faculty, research staff, students, etc., and roles such as professors, students, teaching assistants, general-secretaries, etc.

An organization-agent (or in short *agent*) *plays* one or more roles. Each role is defined with a set of *goals* that the role is created to fulfill and is allocated with proper *authority* at the level that the role can achieve its goals. Agents *perform* activities in the organization, each of which may *consume resource* (e.g. materials, labors, tools, etc.) and there is a set of constraints that constrain the activities. An agent can also be a member of a *team* set up in response to a special task, has *skill* requirements, and has a set of *communication-link* defining the protocol that it communicates with other agents in the organization.

In the rest of this paper, we use *o* to denote organization, *d* denote division, *oa* or *agent* denote organization-agent, *r* denote role, *cl* denote communication link, *g* denote goal, *ath* denote authority, *a* denote activity, *s* denote state, *sk* denote skill, *tm* denote team, *t* denote time, *con* denote constraints.

An organization consists of divisions and has goals that the organization is actively pursuing overall:

consist_of(o, d)

has_goal(o, g)

5.1 Role

A **Role** defines one or more prototypical job functions in an organization. Each role is associated with:

- *Goals*: one or several goals that the role is intended to achieve.

has_goal(r, g)

- *Processes*: activity networks that have been defined to achieve the goals.

has_process(r, a)

- *Authority*: adequate authority needed for the role to achieve its goals. Authorities include the right of using resource, the right to perform *activities*, and the right to execute *status changing actions* (more on *activity* and *status changing action* later).

has_authority(r, ath)

- *Skills*: one or more skills required for the realization of the job functions.

requires_skill(r, sk)

- *Policies*: constraints on the performance of the role's processes. These constraints are unique to the organization role.

has_policy(r, con)

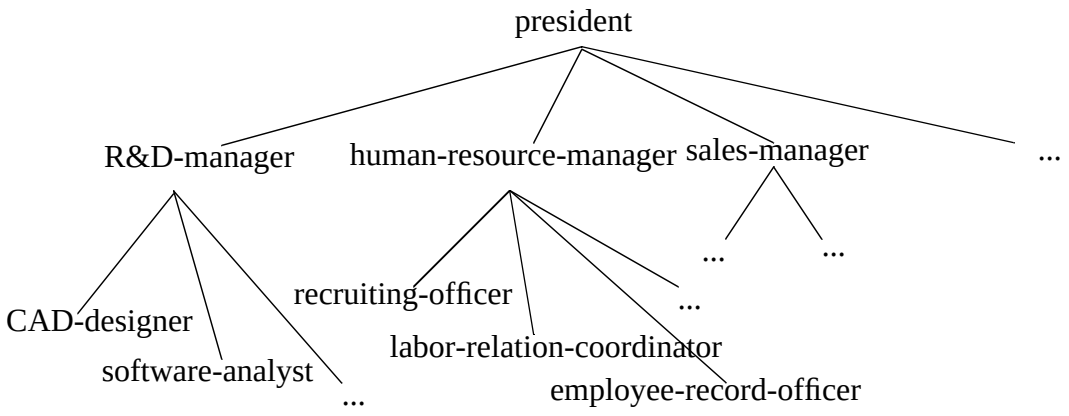
- *Resource:* One or more resources may be allocated to a role for disposition under its authority.

has_resource(r, rs)

5.1.1 Role hierarchy

Within the organization, there is usually a hierarchy of roles. Figure 3 shows an example where there are one president role, several dept-manager roles and many general employee roles, with a latter role being a *subordinate* of a former.

FIGURE 3. Hierarchy of roles in organization



For example, recruiting-officer is a subordinate of human-resource-manager, which in turn is a subordinate of president. We denote *subordinate_of(r, r')* to mean role *r* is a subordinate of role *r'*. The *subordinate_of* relation establishes some kind of ranking of roles. A role higher in the hierarchy is of higher rank. It is obvious that *subordinate* relation is transitive:

$$subordinate\_of(r_1, r_2) \quad subordinate\_of(r_2, r_3) \quad subordinate\_of(r_1, r_3).$$

It is also non-reflexive and anti-symmetric, i.e. no role is a subordinate of itself, and there do not exist two roles that are subordinates of each other.

$$\neg\ subordinate\_of(r, r).$$

$$subordinate\_of(r, r') \quad \neg\ subordinate\_of(r', r).$$

If a role is a subordinate of another role, then the latter is called the *superior* of the former:

$$subordinate\_of(r, r') \equiv superior\_of(r, r').$$

In an organization with *centralized* authority, *authority* increases along the ladder up in the role hierarchy based upon the *subordinate* relation, and therefore the top executive (usually the president) has ultimate authority for everything in the organization.

For this type of organization, we have:

$$subordinate\_of(r, r') \equiv [(\ ath)\ has\_authority(r, ath) \quad has\_authority(r', ath)] \quad (\ ath')\ has\_authority(r', ath') \quad \neg\ has\_authority(r, ath').$$

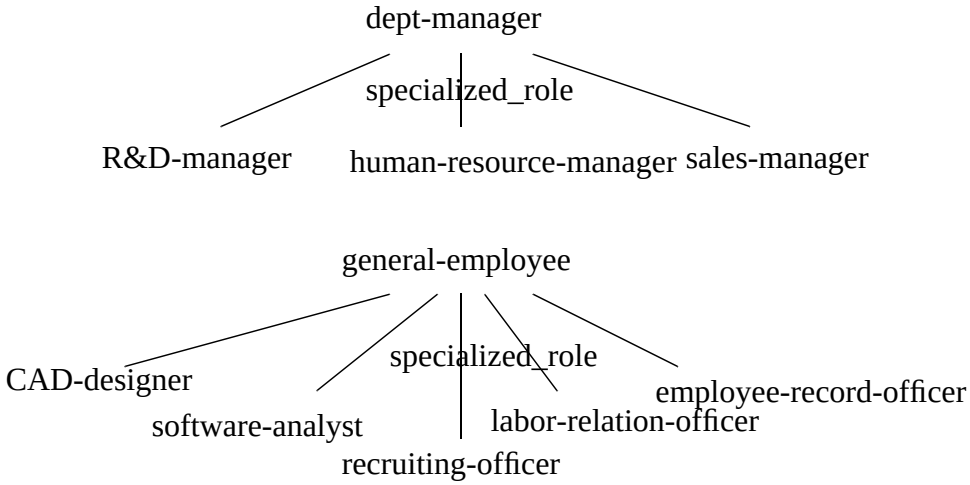
That is, *r* is a subordinate of *r'* if *r* has strictly less authority than *r'*.

In real life, organizations with *decentralized* authority are more common.

5.1.2 Role Generalization and Specialization

Besides the subordinate relation, different roles may also relate to each other through the concept of generalization and specialization [Sandhu et al][Sandhu et al 94]. A role may be a *generalized* or *specialized role* of another. For example, R&D-manager, human-resource-manager, and sales-manager are all *specialized roles* of dept-manager role, and CAD-designer, software-analyst, etc. are specialized roles of general-employee role (Figure 4). In this case, dept-manager and general-employee are *generalized-roles.*

FIGURE 4. Role generalization and specialization



Clearly, specialization and generalization can be captured by class, subclass and instance in object-oriented systems. However, because of the special characteristics of roles, we use a new relation, *specialized_role(r,r')* to mean *r* is a specialized role of *r'*. And the reverse relation *generalized_role(r, r')* is defined as:

generalized_role(r, r') ≡ *specialized_role(r', r).*

Specialized roles “inherit” all the authority from their generalized roles:

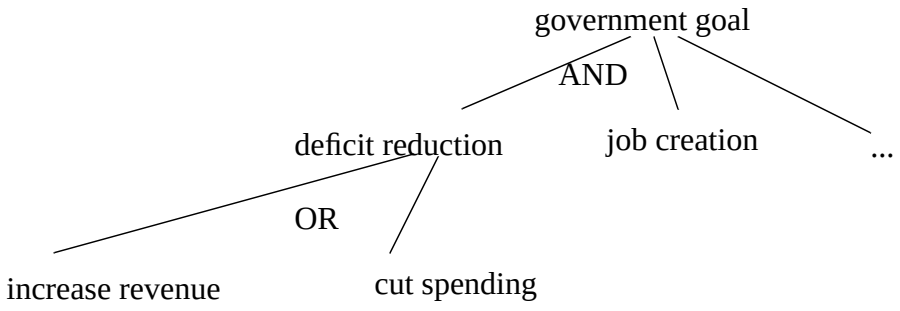
specialized_role(r, r') has_authority(r', ath) has_authority(r, ath). *(1)*

Thus, R&D-manager, human-resource-manager, and sales-manager inherit all the rights that belong to the role dept-manager. And software-analyst, recruiting-officer, etc. inherit the rights of general-employee. Axiom (1), the inheritance property, allow us to grant rights to a whole class of organization-agents without the need of doing so one-by one. For example, if the quarterly earning information of the company is made available to the role dept-manager, it is automatically available to the roles R&D-manager, human-resource-manager, and sales-manager. Daily news or report of the company is often directed to the general-employee role and so every employee in the company can access to it.

5.2 Goal

Our ontology models organization goals that can be decomposed into an AND/OR subgoal trees, and can be achieved by executing activity clusters. Figure 5 shows the goal of a government which is decomposed into subgoals *deficit reduction*, *job creation* and others, where *deficit reduction* goal is decomposed into *increase revenue* or *cut spending* goal.

FIGURE 5. Subgoal tree



The nodes in the subgoal tree have dependency relations amongst them. Let *achieved(g, t)* mean the organization goal *g* has been achieved at time *t*. Then it is clear that for a goal *g* which is decomposed into several subgoals *g1,..., gn* with the "AND" relationship, we have:

achieved(g, t) ≡ *achieved(g1, t) ... achieved(gn, t).*

If the relationship is "OR", we have:

achieved(g, t) ≡ *achieved(g1, t) ... achieved(gn, t).*

A goal *g1* is said to *depend on* goal *g2* if *g1* can not be achieved unless *g2* has been achieved previously*:*

depend_on(g1, g2) ≡ (*t) [achieved(g1, t) achieved(g2, t)].*

Let *decomposition_of(g, g')* mean that goal *g* is a node in the subgoal tree of *g'*. Then every goal of a role in a organization is a decomposition (sub-goal) of some goal of the organization.

has_goal(r, g) (*g') has_goal(o, g') decomposition_of(g, g').*

This ensures that everyone in the organization contributes to the overall goal of the organization. In such a case, we also say that the individual goal is *consistent with* the overall goal.

The goal of a sub-division is a goal or a subgoal of its parent division:

has_goal(d, g) subdivision_of(d, d') has_goal(d', g) (g') has_goal(d', g') decomposition_of(g, g').

5.3 Organization Agent, Division, Team

An organization-agent (*oa)* is an individual member, a human being, in the organization. The concept of organization-agent can be extended to include machine agent or software agent if needed. An organization-agent is a member of some division, plays one or more roles in the organization, can perform activities, and communicate with other *oa*s using communication-links.

member_of(oa, d)

plays(oa, r)

has_communication_link(oa, cl)

If an agent is assigned to a role, then a commitment (more on commitments later) is created on the agent's part to act to achieve the goal(s) of the role. The goals of an agent are defined as the goals of the roles that the agent plays:

has_goal(oa, g) ≡ (r) plays(oa, r) has_goal(r, g).

And the authority of the agent is the authority of the roles that the agent plays:

has_authority(oa, ath) ≡ (r) plays(oa, r) has_authority(r, ath).

Each organization-agent is member of or affiliated with some *division* (or sub-division) in the organization. In our model each agent is a member of some division:

(oa) (d) member_of(oa, d).

An agent can be a member of more than one divisions. This is shown by the following axiom where *home_div(oa, d)* means that the home division of agent *oa* is *d*:

home_div(oa, d) home_div(oa, d') d = d'.

An organization-agent may also be a member of some *teams* set up to pursue specific tasks in the organization. Compared to *division* which is usually a long-term setup within the organization, *team* is temporary in nature and is usually set up when needed. Members of a team may be from different divisions and there may be many teams set up in the organization. The relationship between an agent and a team is *member_of(oa, tm)*, which means agent *oa* is a member of team *tm*.

Only two or more members can form a team; hence we have:

(tm) (oa, oa') oa ≠ oa' member_of(oa, tm) member_of(oa', tm).

A team, as a whole, can play a role in the organization. If everyone in a team plays a same role, we also say that the team plays the role:

(r, tm) [(oa) member_of(oa, tm) plays(oa, r)] plays(tm, r).

5.4 Communication-link

Communication-links are established among organizational agents in various roles. **Communication-links** capture the notion of benevolent communication in which agents regard each other as peers volunteer information that they believe relevant to other agents. This exchange does not create obligations for any agent.

The **communication-link** is a unidirectional link used to communicate information from one agent to another. It describes, for an agent in a given organizational role, the information it is interested in receiving and the information it can benevolently distribute to others.

For example, an agent in the "C++ programmer" role may distribute information about the state of the file server to other programmers, alerting them each time the server is down.

The communication-link specifies:

- *Sending-Agent*: the agent sending information along the link.

  has_sending_agent(cl, oa)

- *Receiving-Agent*: the agent receiving information from the link.

  has_receiving_agent(cl, oa)

- *Sending-Role*: the organization role played by the sending agent.

  has_sending_role(cl, r)

- *Receiving-Role*: the organization role played by the receiving agent.

  has_receiving_role(cl, r)

- *Interests*: the information interests of the receiving agent.

  has_interest(cl, inf)

- *Volunteers*: the information the sending agent can supply to other agent.

  will_volounteer(cl, inf)

It is understood that information distribution in the above case is non-committal, in the sense that it does not create obligations for either the sender or the receiver. Our axiomatization of the communication-link focuses on consistency. In particular, that the agent specified by an communication-link also has the role specified by the link.

has_sending_agent(cl, oa) *has_sending_role(cl, r)* *has_role(oa, r)*

has_receiving_agent(cl, oa) *has_receiving_role(cl, r)* *has_role(oa, r)*

5.5 Authority and Commitment

A special kind of authority is the control relationship between two organizational agents. For OA1 to have *authority* over OA2 implies that OA1 is able to extract a commitment from OA2 to achieve a **goal** that is defined as part of OA2's organization-roles. In order to extract that commitment, OA1 has to be related directly or indirectly by a **communication-with-authority** (CWA) link relation.

communication_with_authority(cwa)

has_supervisor(cwa, oa)

has_supervisee(cwa, oa)

The **Communication-with-Authority** link, used when communication is intended to create obligations, specifies the two agents, one in the authority position (called *supervisor*) and the other in the controled position (called *supervisee*), among which communication takes place. Because we model communication as exchange of speech-acts, authority of an agent appears as the set of speech-acts this agent can use in order to create obligations for the other agent. For example, an agent may have authority to request another agent to perform action A1, but not to perform action A2. In this case, the second agent will have to commit to achieving A1 when requested by the first agent, but not A2.

We introduce a set of properties for the communication-with-authority relation that delineate the extent of authority that can be exercised by a supervisor.

- *has_goal:* defines the set of goals that the supervisor can assign to the supervisee.

  has_goal(cwa, g)

- *has_resource:* defines the set of resources that the supervisor can allocate to the supervisee.

  has_resource(cwa, rs)

- *has_empowerment:* defines the set of empowerment the supervisor can assign to the supervisee (More on empowerment later).

  has_empowerment(cwa, em)

- *has_role:* defines the roles that the supervisor can assign to the supervisee.

  has_roles(cwa, r)

The authority relationship is defined among agents in given organization roles. An agent in a "project-manager" role can request an another agent in a "programmer" role to write a program for a given function, but can not request the second agent to, say, deliver a mail package. This is because writing programs is a goal of the "programmer" role, while delivering mail package is not.

We introduce the concept of an OA's commitment to achieving a goal [Jensen 93]. The predicate

committed_to(oa, g)

signifies that agent *oa* is committed to the achievement of goal *g*. Consequently, the totality of activities performed by *oa* must include the achievement of *g*. Prioritisation of goals, etc. are not considered here.

The next axiom states that any agent that fills an organization role is committed to the goals associated with the role.

has_role(agent, role) [has_goal(role, goal) commited_to(agent, goal)]

An agent can only allocate resources that have been assigned to a role it plays.

has_resource(cwa, rs) (r) [has_supervisor(cwa, oa) plays(oa, r) has_resource(r, rs)]

5.6 Empowerment: Linking Structure and Behavior

With the introduction of organizational knowledge, we now have to address the problem of how to specify "who can do what". That is, what is the set of activities that an OA is allowed to perform as a member of the Organization. It would appear that by associating processes with OAs via the **has_process** property, we have solved the problem. That is, an OA can perform any activities specified by the process in the roles that the OA plays. But consider the following situation:

> "Jill, in her role as a CNC machine operator, has a process she must perform in order to achieve the goal of producing an order. The process is composed of three activities: 1) machine-setup, 2) machine-run and 3) machine-teardown. But before the machine-run activity can commence, she must receive permission from her supervisor."

The problem is that Jill has a process that specifies a sequence of activities that she must perform. But she cannot perform the second activity, machine-run, without permission. The implication is that within our Activity ontology, she is not allowed to change the state of the machine-run activity to "execute".

An obvious way to solve the problem is to insert a fourth activity between machine-setup and machine-run where she seeks approval from her supervisor. If approval is obtained, then she can commence the subsequent machine-run activity. Again we have a problem. Who is allowed to change the status of this new approval activity to "completed"? If Jill is allowed to make any status changes she wants to the activities in her process, she can change the status of the approval activity regardless of whether she obtained approval or not.

The problem lies with who is allowed to make status changes to states and activities. When Jill goes to her supervisor for permission, is it Jill who changes the status of the approval activity to completed or her supervisor? It is not clear. Therefore the only solution to the problem of permission to perform

an action lies in precisely stating who is allowed to change the status of the activity, e.g., from "dormant" to "executing".

We introduce the concept of Empowerment as a means of specifying the status changing rights of an *oa*. *Empowerment is the right of an oa to perform status changing actions*, such as "commit", "enable", "suspend", etc. Empowerment naturally falls into two classes: state and activity empowerment.

State empowerment specifies the range of stati through which an *oa* may take a state by performing the appropriate actions, such as commit. State empowerment not only specifies allowable status changes but may be used to restrict the set of resources an *oa* is empowered to commit to a use/consume state. An *oa* may be empowered for any type of resource, including other *oa*s. The implication being the first *oa* may commit the second to a state.

state-empower(oa, s, c)

where *s* is a state, and *c* is one of: "commit", "enable", "complete", "disenable", "reenable".

Activity empowerment specifies the range of stati through which an *oa* may take an activity by performing the appropriate actions, such as execute and suspend. Even though an activity may be enabled, the *oa* whose role contains the process which contains the activity may not be empowered to start its execution.

activity-empower(oa, a, c)

where *a* is an activity, and *c* is one of: "execute", "suspend", "complete".

With the addition of empowerment, a second type of authority arises. That is, the supervising agent may alter what a supervisee is empowered to do.

Following are the definitions of and constraints on the empowerment predicates.

For any activity *a* that requires that the agent be empowered, the status changing actions for the activity require *holds(activity_empower(agent,a,c),σ)* as a precondition.

For any state *s* that requires that the agent be empowered, the status changing actions for the activity require *holds(state_empower (agent,s,c),σ)* as a precondition.

It is possible to for one agent to empower another agent for an activity if the first agent supervises the second, and the supervisor is empowered for that activity.

$$Poss(activity\_empowers(agent, agent', a, c), \sigma) \equiv holds(supervises(agent, agent'), \sigma) \quad holds(activity\_empower(agent, a, c), \sigma)$$

It is possible to for one agent to disempower another agent for an activity if the first agent supervises the second, and the supervisor is empowered for that activity.

$$Poss(activity\_disempowers(agent, agent', a, c), \sigma) \equiv holds(supervises(agent, agent'), \sigma) \quad holds(activity\_empower(agent, a, c), \sigma)$$

An agent is empowered for an activity only as a result of the action *activity_empowers*, and is no longer empowered only as a result of the action *activity_disempowers*.

$$holds(activity\_empower(agent, a, c), do(a', \sigma)) \equiv (\exists agent')\, a' = activity\_empowers(agent', agent, a, c)$$
$$holds(activity\_empower(agent, a, c), \sigma) \quad \neg(\exists agent')\, a' = activity\_disempowers(agent', agent', a, c)$$

It is possible for one agent to empower another agent for changing the status of states if the first agent supervises the second, and the supervisor is empowered for changing the status of that state.

$$Poss(state\_empowers(agent, agent', s, c), \sigma) \equiv holds(supervises(agent, agent'), \sigma) \quad holds(state\_empower(agent, s, c), \sigma)$$

It is possible to for one agent to disempower another agent for changing the status of that state if the first agent supervises the second, and the supervisor is empowered to change the status of that state.

$$Poss(state\_disempowers(agent, agent', s, c), \sigma) \equiv holds(supervises(agent, agent'), \sigma) \quad holds(state\_empower(agent, s, c), \sigma)$$

An agent is empowered for changing the status of a state only as a result of the action *state_empowers*, and is no longer empowered only as a result of the action *state_disempowers*.

$$holds(state\_empower(agent, a, c), do(a', \sigma)) \equiv (\exists agent')\, a' = activity\_empowers(agent', agent, a, c)$$
$$holds(state\_empower(agent, a, c), \sigma) \quad \neg(\exists agent')\, a' = state\_disempowers(agent', agent', a, c)$$

5.7 Agent interaction and speech acts

Agent interaction takes place at several levels. The first level is concerned with the *information content* communicated among agents. A piece of information communicated at this level may be a proposition (fact) like “(produce 200 widgets)”.

The second level specifies the *intentions* of agents. The same information content can be communicated with different intentions. For example:

- (*ask* (produce 200 widgets)) - the sender asks the receiver if the mentioned fact is true,
- (*tell* (produce 200 widgets)) - the sender communicates a belief of his to the receiver,
- (*achieve* (produce 200 widgets)) - the sender requests the receiver to make the fact one of his beliefs
- (*deny* (produce 200 widgets)) - the sender communicates that a fact is no longer believed.

This level supports interaction through explicit linguistic actions, called *speech-acts*. The speech act [Searle 69] framework has been developed by philosophers and linguists to account for human communication.

The third level is concerned with the *conventions* [Jennings 93] that agents share when interacting by exchanging messages. The existence of shared conventions makes it possible for agents to coordinate in complex ways, e.g. by carrying out negotiations about their goals and actions. As an example, consider the supply chain of our TOVE virtual manufacturing enterprise as a multi-agent system. The Order Acquisition Agent interacts with the customer and acquires an order for 200 lamps with a due date for 28 Sept. 94. It sends this as a *proposal* to the Logistics Agent. Knowing that Logistics can only answer with *accepting*, *rejecting* or *counter-proposing*, Order Acquisition is able to check that the actual response is one of these and carry out a corrective dialogue with Logistics if this is not the case or if other events occur (such as delays or lost messages). If Logistics answers with a counter-proposal (e.g. 200 lamps with due date 15 oct 94), Order Acquisition may use knowledge about acceptable trade-off and negotiate with Logistics an amount and a due-date that can be achieved and satisfies the customer. In its turn, upon receiving the order proposal, Logistics will start negotiations with the Scheduling agent to determine the feasibility of scheduling the production of the order and with the Transportation agent to determine feasibility of the delivery date.

In order to be able to model the complexity of the third level, the organization ontology includes a representation of speech-acts as a foundation. This representation distinguishes between the performative (e.g. propose, accept, reject, etc.), the sender (an agent initiating the communication) the receiver (an agent receiving the communication) the content being communicated (we make no commitment at this level), the language the content is expressed in and the ontology defining the context in which the content message must be interpreted.

Our solution for the third level of interaction is reported in [Barbuceanu & Fox 95].

6.0 Competency Revisited

In this section we repeat the competency and follow each with the query that would provide the answer. Note that parameters that are preceded with a "?" are variables. We also leave out the situational calculus predicates since all of the logical statements refer to the current situation.

6.1 Structure

- What roles does agent P play?

  ```
  plays(P,?r)
  ```

- Which division does the agent belong to?

  ```
  member_of(P,?d)
  ```

- Who must agent P communicate with (*?rec* provides the answer)?

```
plays(P,?r)  has_communication_link(?r,?cl) has_receiving_agent(?cl,?rec)
```

- What kinds of information does person P communicate (*?inf* provides the answer)?

```
plays(P,?r)  has_communication_link(?r,?cl) will_volounteer(?cl,?inf)
```

- Who does P report to (*?p* provides the answer)?

```
subordinate_of(P,?p).
```

- Is role R a generalization of another role R'?

```
generalized_role(R, R').
```

6.2 Behavior

- What are the goals of the organization O (*?g* provides the answer)?

```
has_goal(O,?g)
```

- What are the goals of the role R (*?g* provides the answer)?

```
has_goal(R,?g)
```

- What are the goals of person P (*?g* provides the answer)?

```
plays(P,?r)  has_goal(?r,?g)
```

- What activities are available for a role R to achieve its goals (*?a* provides the answer)?

```
has_process(R,?a)
```

- What resources are available to the agent P to achieve a goal G (*?rs* provides the answer)?

```
plays(P,?r)  has_goal(?r, G)  has_resource(?r,?rs)
```

6.3 Authority, Empowerment and Commitment

- What resources does the person P have authority to assign (*?rs* provides the answer)?

```
plays(P,?r)  has_resource(?r,?rs) state_empower(P,?s, commit) ((?s = use)  (?s =
consume))
```

- Whose permission does an agent need to perform activity A (*?oa* provides the answer)?

```
activity_empower(?oa, A, execute)
```

- What activities may a person P execute without explicit permission (*?a* provides the answer)?

```
activity_empower(P,?a, execute)
```

7.0 Conclusions

Ontologies are shared views of domains. They provide conceptualizations that are agreed upon by participants in collaborative action and decision making. The explicit existence of such shared per-

spectives makes it possible for both people and programs to collaborate by ensuring that everybody makes the same distinctions and uses the same terms with the same meaning.

The paper presents our preliminary exploration into an organization ontology for the TOVE enterprise model. The ontology views organizations as composed of agents playing roles in which they are acting to achieve specific goals according to various constraints defining the “rules of the game”. A primary focus has been in linking structure and behavior through the concept of empowerment. Empowerment is the right of an organization agent to perform status changing actions. This linkage is critical to the unification of enterprise models and their executability. Further work can be done on completing the axiomatization of the ontology and extending it to capture other concepts such as skill, intention, access right to the information system of the organization, etc.

8.0 Acknowledgments

This research is supported, in part, by the Natural Science and Engineering Research Council, Manufacturing Research Corporation of Ontario, Digital Equipment Corp., Micro Electronics and Computer Research Corp., Numetrix, Spar Aerospace, Carnegie Group, Quintus Corp, Toyo Engineering, and BHP Research.

This paper revises and extends an earlier version that appeared as: Fox, M.S., Barbuceanu, M., Gruninger, M., (1996), “An Organisation Ontology for Enterprise Modeling: Preliminary Concepts for Linking Structure and Behaviour”, *Computers in Industry*, Vol. 29, pp. 123-134.

9.0 References

[Auramaki et al. 88] Auramaki, E., Lehtinen, E. and Lyytinen, K. A speech-act-based office modelling approach, *ACM Transactions on Office Information Systems*, Vol. 6, No. 2, april 1988, 126-152.

[Blackburn 91] Blackburn J. *Time-based Competition.* Business One Irwin, 1991.

[Fadel et al 94] Fadel, F., Fox, M.S., and Gruninger, M. A resource ontology for enterprise modelling. *Third Workshop on Enabling Technologies-Infrastructures for Collaborative Enterprises*, (West Virginia University 1994).

[Flippo and Munsinger 75] Flippo, E. and G. M. Munsinger. Management. Allyn and Bacon, Inc. 1975.

[Flores et al. 88] Flores, F., Graves, M., Hartfield, B. and Wionograd, T. Computer systems and the design of organizational interaction, *ACM Transactions on Office Information Systems*, Vol. 6, No. 2, april 1988, 153-172.

[Fox et al. 93] Fox, M.S., Chionglo, J., Fadel, F. A Common-Sense Model of the Enterprise, *Proceedings of the Industrial Engineering Research Conference 1993.*

[Gruber 93] Gruber, T. R.,(1993): Toward principles for the design of ontologies used for knowledge sharing, Report KSL 93-04, Stanford University, august 1993.

[Gruninger & Fox 94] Gruninger, M., and Fox, M.S., (1994), "The Role of Competency Questions in Enterprise Engineering", *Proceedings of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice,* Trondheim, Norway. June 94.

[Jennings 93] Jennings, N., R., (1993): Commitments and conventions: The foundation of coordination in multi-agent systems, The Knowledge Engineering Review, vol. 8:3, pp 223-250, 1993.

[Kim & Fox 93] Kim, H. and Fox, M.S. Quality Systems Modelling: A Prospective for Enterprise Integration, *Fourth Annual Meeting of the Production and Operations Management Society.* 1993.

[Lee 88] Lee, R. M. Bureaucracies as deontic systems, *ACM Transactions on Office Information Systems*, Vol. 6, No. 2, april 1988, 87-108.

[McFarland 74] McFarland, D. Management: Principles and Practices. Macmillan Publishing Co., Inc. 1974.

[Mintzberg 83] Mintzberg, H. *Structure in Fives - Designing Effective Organizations*, Prentice Hall Inc., 1983

[Pinto & Reiter 93] Pinto, J. and Reiter, R. Temporal reasoning in logic programming: A case for the situation calculus. In *Proceedings of the Tenth International Conference on Logic Programming* (Budapest, June 1993).

[Reiter 91] Reiter, R. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy.* Academic Press, San Diego, 1991.

[Sandhu et al] Sandhu, R. S., Coyne, E. J., Feinstein, H. L. and Youman, C. E. Role-based access control models. IEEE Computer, to appear.

[Sandhu et al 94] Sandhu, R.S., Coyne, E.J., Feinstein, H.L. and Youman, C.E. "Role-Based Access Control: A Multi-Dimensional View." *Proc. Tenth Annual Computer Security Applications Conference*, Orlando, Florida, December 5-9, 1994, pages 54-62.

[Sathi et al 85] Sathi, A., Fox, M.S., and Greenberg, M. Representation of activity knowledge for project management. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. PAMI-7:531-552, September, 1985.

[Searle 69] Searle, J. *Speech Acts,* Cambridge University Press, Cambridge, UK, 1969.

[Tham et al. 94] Tham, D., Fox, M.S., and Gruninger, M., A cost ontology for enterprise modelling *Third Workshop on Enabling Technologies-Infrastructures for Collaborative Enterprises,* (West Virginia University 1994).

[Weber 87] Weber, M. Economy and Society, *University of California Press,* Berkeley, Calif. 1987.

[Winograd 87] Winograd, T. A language/action perspective on the design of cooperative work, *Human Computer Interaction* 3, 1 (1987-1988), 3-30

[Yu & Mylopoulos 94] Yu, E. S. K. and Mylopoulos, J. From E-R to "A-R" - Modelling strategic actor relationships for business process reengineering, *13-th Int. Conf. on the Entity-relationship Approach*, Dec. 13-16 1994, Manchester, UK.