# An Architecture for Agents with Obligations

Mihai Barbuceanu
Enterprise Integration Laboratory
University of Toronto,
4 Taddle Creek Road, Rosebrugh Building,
Toronto, Ontario, Canada, M5S 3G9
mihai@ie.utoronto.ca

**Abstract.** We explore the view that coordinated behavior is driven by the social constraints (e.g. obligations and interdictions) that agents in organizations are subject to. In this framework, agents adopt those goals that are requested by their obligations, knowing that not fulfilling obligations induces a price to pay or a loss of utility. Based on this idea we build a coordination system where we represent the organization, the roles played by agents, the obligations imposed among roles, the goals and the plans that agents may adopt. To consistently update an agent's obligations and interdictions we present an incremental propagation method that also helps with detecting and deciding between conflicting obligations. Once a goal adopted, a special brand of plans, called conversation plans, are available to the agents for effectively carrying out coordinated action. Conversation plans explicitly represent interactions by message exchange and their actions are dynamically reordered using the theory of Markov Decision Processes to ensure the optimization of various criteria. The framework is applied to model supply chains of distributed enterprises.

## 1 Introduction and Motivation

To build autonomous agents that work coordinately in a dynamically changing world we have to understand two basic things. The first is how an agent chooses a particular course of action and how its choices change in face of events happening in the world. The second is how agents execute coordinated actions. In this paper we present a framework that answers these questions by construing agents as rational decision makers that exist within organizations. Organizations are systems that constrain the actions of member agents by imposing mutual obligations and interdictions. The association of obligations and interdictions is mediated by the *roles* agents play in the organization. For example, when an agent joins a software production organization in the `system administrator` role, he becomes part of a specific constraining web of mutual obligations, interdictions and permissions - *social constraints or laws* - that link him as a `system administrator` to `developers`, `managers` and every other role and member of the organization. Not fulfilling an obligation or interdiction is sanctioned by paying a cost or by a loss of utility, which allows an agent to apply rational decision making when choosing what to do.

Social laws are objective forces that provide the ultimate motivation for coordinated action at the organization level and to a large extent determine the *mental states* at the individual agent level. Agents "desire" and "intend" the things that are requested by their current obligations, knowing that otherwise there will be a cost to pay. However, current models of collective behavior largely ignore this aspect, trying to explain coordination solely from the perspective of socially unconstrained individuals. For this reason they often impose restrictive conditions that limit the generality of the models. For example, the Cohen-Levesque account of teamwork [Cohen & Levesque 91] requires team members to have the same mutual goal. But this is not true in organizations, for example it is normal for supervisors to decompose and schedule work and assign team members different goals which they can carry out coordinately often without being even aware of each other's goals. Similarly, dropping off a team, according to the Cohen-Levesque model, requires an agent to make his goal to convince every member of the team that, e.g., the common motivation for the joint action has disappeared. Since common goals or motivations do not necessarily exist, this is also not true in general. Imagine an agent having to convince everybody else that he's got a better job elsewhere before he can leave the organization!

To initiate a more realistic investigation of such social constraints and of their role in achieving coordinated behavior, we have built an agent coordination framework whose building blocks include the entities social constraints are made of: agents, organizations, roles, obligations, interdictions, permissions, goals, constraints, plans, etc. In this framework agents determine what obligations and interdictions currently apply and on this basis decide on their goals, even when these obligations are in contradiction. Once an agent has chosen a goal, it selects a plan to carry it out. Plans are described in a planning formalism that explicitly represents interactions with other agents by means of structured conversations involving message exchanges (hence the name of conversation plans). Plan actions are dynamically ordered using a Markov Decision Process method to maximize certain kinds of rewards. Besides this empirical line of work that enables us to put ideas to the test immediately and evaluate the significance of our solutions, we are also developing semantic accounts that precisely establish the meaning of our concepts. Details follow.

## 2 The Vocabulary for Social Constraints

We start by briefly introducing our application domain, the integration of supply chains of manufacturing enterprises. A supply chain is a globally extended network of suppliers, factories, warehouses, distribution centers and retailers through which raw materials are acquired, transformed into products, delivered to customers, serviced and enhanced. The key to the efficient operation of such a system is the tight coordination among components. But the dynamics of the enterprise and of the world market make this difficult: customers change or cancel orders, materials do not arrive on time, production facilities fail, workers are ill, etc. causing deviations from plan. Our goal is thus to achieve coordi-

nated behavior in dynamic systems of this kind by applying agent coordination technologies.

At the highest level, the above defines an *organization* where agents play various roles. An organization consists of a set of roles filled by a number of agents. In the example below, `customer`, `coordinator` etc. are roles filled respectively by agents `Customer`, `Logistics`, etc.

```
(def-organization SC1
  :roles ((customer Customer)
          (coordinator Logistics)
          (assembly-plant Plant1)
          (painting-plant Plant2)
          (transportation Transp1)))
```

An agent can be a member of one or more organizations and in each of them it can play one or more roles. An agent is aware of the existence of some of the other agents, but not necessarily of all of them. Each agent has its local store of beliefs (taken as a data base rather than mental states).

A *role* describes a major function together with the obligations, interdictions and permissions attached to it. Roles can be organized hierarchically (for example `assembly-plant` and `painting-plant` would be both `manufacturing` roles) and subsets of them may be declared as disjoint in that the same agent can not perform them (like `manufacturing` and `transport`). For each role there may be a minimum and a maximum number of agents that can perform it (e.g. minimum and maximum 1 `president`).

*Obligation, Interdiction, Permission.* An agent `a1` in role `r1` has an obligation towards an agent `a2` in role `r2` for achieving a goal `G` according to some constraint `C` iff the non-performance by `a1` of the required actions allows `a2` to apply a sanction to `a1`. Agent `a2` (who has authority) is not necessarily the beneficiary of executing `G` by the obliged agent (you may be obliged to your manager for helping a colleague), and one may be obliged to oneself (e.g. for the education of one's children).

In our language we provide a construct for defining obligations generically. The generic obligation exists between two agents in specified roles, whenever a given condition applies. The obligation requires the obliged agent to achieve a goal under given constraints. For example:

```
(def-obligation Reply-to-RFQ
 :obliged coordinator
 :authority customer
 :condition (received-RFQ
             :from (agent-playing customer)
             :by (agent-playing coordinator))
 :goal gReply-to-RFQ
 :enforced (max-reply-time 5))
```

The above requires the `coordinator` agent (the `:obliged` party) to reply to a request for quotation (RFQ) from the `customer` (the `:authority` party) in at most five units of time (a constraint on the goal `gReply-to-RFQ`). This generic obligation becomes active when its condition is satisfied, in this case when the `coordinator` receives a RFQ from the `customer` which requires a reply time not shorter than 5, because that is the best the obligation requires the agent to do. When this happens (checked by evaluating the `:condition predicate`), an actual obligation is created linking the `coordinator` to the `customer` and applying to the actual RFQ received (if many RFQ-s are received, as many actual obligations are created).

In exactly the same manner, our language defines generic and actual *interdictions* (the performance of the goal is sanctioned) and *permissions* (neither the performance nor non-performance are sanctioned). We represent permissions explicitly because we do not assume everything not explicitly obliged or forbidden to be permitted. Agents may choose their goals from their explicit permissions, or requests that can not be proven as obligatory may be served as permissions. Finally, the obligations, interdictions and permissions (short OPI-s) of a role are inherited by sub-roles.

Semantically, OPI-s are modeled using the reduction of deontic logic to dynamic logic due to [Meyer 88] in a multi-agent framework. As this is ongoing work, we only review a few elements that will be used later on to explain the propagation method we use to infer the obligations, permisions and interdictions of agents. We define obligation, interdiction and permission as follows, where $V_\alpha^{ij}$ denotes a violation by $i$ of a constraint imposed by $j$ wrt $\alpha$ (associated with a cost to be paid):

- $F^{ij}\ \alpha \equiv [\alpha]^i V_\alpha^{ij}$: $i$ is forbidden by $j$ to execute $\alpha$.
- $P^{ij}\ \alpha \equiv \neg F^{ij}\ \alpha$: $i$ is permitted by $j$ to execute $\alpha$.
- $O^{ij}\ \alpha \equiv F^{ij}(\text{-}\alpha)$: $i$ is obliged by $j$ to execute $\alpha$.

This reduction leads to a number of validities which, as will be shown immediately, allow us to apply a constraint propagation method to infer new OPI-s from given ones. In the following (indices dropped for clarity), ; denotes sequential composition, $\cup$ nondeterministic choice and & parallel composition of actions.

$\models \mathrm{F}(\alpha;\beta) \equiv [\alpha]\mathrm{F}\beta$
$\models \mathrm{F}(\alpha \cup \beta) \equiv \mathrm{F}\alpha \wedge \mathrm{F}\beta$
$\models (\mathrm{F}\alpha \vee \mathrm{F}\beta) \supset \mathrm{F}(\alpha \& \beta)$
$\models \mathrm{O}(\alpha;\beta) \equiv (\mathrm{O}\alpha \wedge [\alpha]\mathrm{O}\beta)$
$\models (\mathrm{O}\alpha \vee \mathrm{O}\beta) \supset \mathrm{O}(\alpha \cup \beta)$
$\models \mathrm{O}(\alpha \ \& \ \beta) \equiv (\mathrm{O}\alpha \wedge \mathrm{O}\beta)$
$\models \mathrm{P}(\alpha;\beta) \equiv < \alpha > \mathrm{P}\beta$
$\models \mathrm{P}(\alpha \cup \beta) \equiv (\mathrm{P}\alpha \vee \mathrm{P}\beta)$
$\models \mathrm{P}(\alpha \ \& \ \beta) \supset (\mathrm{P}\alpha \wedge \mathrm{P}\beta)$.

*Goal.* According to the dynamic logic framework, goals are either `atomic` (non-decomposable), or compositions of type `seqence`, `parallel` or `choice`. When the type indicates a composition, the goal will specify its components

as subgoals. The description of goals also includes the constraints that can be applied to a goal and the optimizations that can be requested. For example:

```
(def-goal gReply-to-RFQ
   :agent Logistics
   :type choice
   :subgoals (gReply-RFQ-heuristically
              gReply-RFQ-by-scheduling)
   :constraints
     (max-reply-time req-max-reply-time)
   :optimizations (time accuracy))
```

This specifies that goal `gReply-to-RFQ`, belonging to agent `Logistics`, is of choice type and will be satisfied by executing one of its two subgoals. One subgoal answers the RFQ by using heuristics estimating the time required and the probable costs, while the other runs scheduling software to determine exactly if the order can be delivered on time and cost. It can only be constrained by two constraints, `max-reply-time` and `req-max-reply-time`. Whatever plan is used for this goal, its execution may be optimized for both time and accuracy.

*Conversation Plan.* Finally, agents have plans for achieving their goals. At the outermost level plans specify the goal they can be used for, the constraints they guarantee (a plan for `gReply-to-RFQ` may guarantee `(max-reply-time 4)` and thus be applicable for the above obligation) and the optimizations that their execution can provide. A plan is usable for a goal if the requested constraints are satisfied by the plan and preferably (but not necessary) if the plan execution can provide the requested optimizations. More details to follow.

## 3 Reasoning About What To Do (or Not)

Suppose now the `coordinator` receives a message from the `customer` in which the latter requests a quotation for delivering 1000 widgets before some date with a reply in less that 7 units of time and preferring that the `coordinator`'s response be as accurate as possible:

```
(ask :from (customer Customer)
   :to (coordinator Logistics)
   :content ((RFQ :product widget
                  :amount 1000
                  :due-date 17-sept-97)
             :satisfy (req-max-reply-time 7)
             :optimize (accuracy)))
```

This matches the `Reply-to-RFQ` obligation of `Logistics` and hence `Logistics` instantiates an actual obligation for replying to the RFQ. From this a goal `gReply-to-RFQ` to respond in less than 7 units of time and with (preferred) maximal accuracy is generated.

Because of the constraints amongst goals, asserting a goal as obliged or forbidden may have consequences over the other goals of the agent by changing their deontic status. As a result, new goals may become obliged or forbidden or the same goal may become contradictory (both obliged and forbidden). For example, if goal `gReply-to-RFQ` becomes obliged and its subgoal `gReply-RFQ-by-scheduling` is forbidden because of some reason, it follows that subgoal `gReply-RFQ-heuristically` becomes obliged. If both subgoals are forbidden, then the goal `gReply-to-RFQ` is also forbidden and attempting to make it obliged later results in a contradiction.

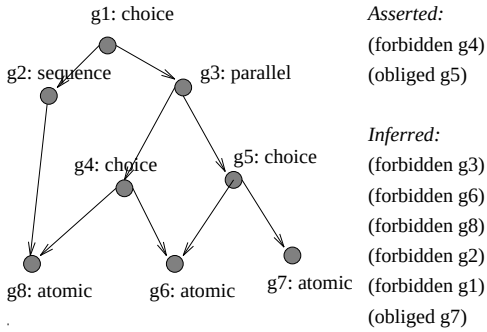


**Fig. 1.** Deontic propagation in a goal network.

To address the general problem of updating the deontic status of goals we apply an incremental method for deontic constraint propagation. Figure 1 illustrates the process using an arbitrary goal network where we have asserted `(forbidden g4)` and `(obliged g5)`. For each of these assertions the propagation process traverses the network along supergoal and subgoal links and applies the deontic validities listed previously. For example, since `g4` is a choice, making it forbidden implies that all its alternatives are also forbidden. This makes both `g8` and `g6` forbidden. Since `g8` is forbidden, `g2` is also forbidden as a sequence with one action forbidden. Propagating along supergoals, `g3` becomes forbidden because of `g4`. Now both `g2` and `g3` are forbidden, and since they are all the alternatives of `g1`, `g1` becomes forbidden as well. When `g5` is made obliged, since `g6` is forbidden, it follows that `g7` must be obliged.

Sequential composition of subgoals creates a new problem for deontic propagation in that if a (whole) sequence is forbidden (or obliged), the status of subgoals (sequence members) can not be determined in advance. As stated by the previous deontic valitities, only after one subgoal has been executed we can post an interdiction (or obligation) for the remaining part of the sequence. Thus, goal execution has to be monitored and for goals that belong to sequences new propagations need to be performed when these goals are finished.

When goals are asserted as obliged or forbidden, we also allow specifying a cost of violating the obligation or interdiction. This helps us handle conflicting

situations. In figure 2 we assert every subgoal of a choice as forbidden with a given cost. Then the choice goal is asserted as forbidden with a cost equal to the cost of the smallest cost alternative (this is just one possibility). If later the choice goal is asserted as obliged with a greater cost, then we propagate this upon the smallest cost subgoal. Now we have contrdictory assertions on `g1` and `g2`, but since we also have the violation costs, the agent is justified to do `g1` and `g2` (accepting their obligatory status) because thus it will incur a smaller penalty of 2 as opposed to a penalty of 5 otherwise. We consider this not as the definitive solution to the problem, but rather as our first shot at it.

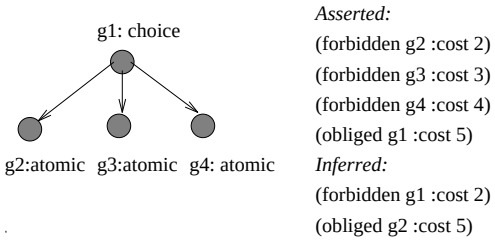


**Fig. 2.** Deontic propagation with costs and conflicts.

Once the deontic status of goals updated, the agent knows what it can do and what it can't. The next step is to order goals (perhaps by violation costs) and select plans for each goal. A plan is selected for a goal if its applicability condition allows it. Once selected, the requested constraints and optimizations are passed to the plan as arguments (plans have a local data store where these are held). Plans can be associated with any goal, not necessarily to atomic goals. A plan for a goal can trigger the execution of another plan for a subgoal of that goal, provided the goal is permitted. Thus, the plan for `gReply-to-RFQ` can trigger the plan for `gReply-RFQ-heuristically`, wait for its completion and then continue. The mechanisms for this are described later on.

Finally, the operational architecture keeps track of how obligations are derived from events like received messages, how goals are derived from obligations and how plans have been selected for goals, using a logical truth maintenance system. This allows agents to retract obligations and interdictions when agents change decisions, like retracting an order or dropping an interdiction because of a more important incompatible obligation. In figure 1 for example, if `(forbidden g4)` is retracted, all inferred assertions will also be retracted as they all rely on this premise.

## 4 Plan Specification and Execution

We now briefly review the plan specification and execution mechanisms in order to connect them with the OPI architecture.

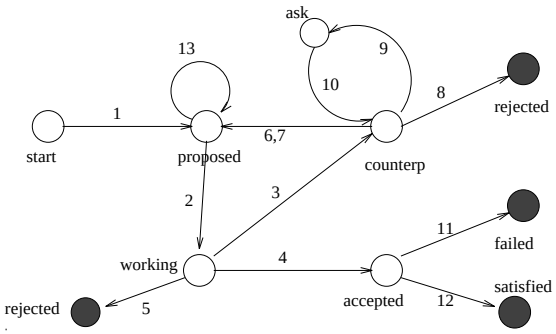


**Fig. 3.** Graph representation of Customer-conversation.

*Conversation plans* [Barbuceanu & Fox 96] are descriptions of how an agent *acts* and *interacts* in certain situations. A conversation plan consists of states (with distinguished initial and final states) and rule governed transitions together with a control mechanism and a local data-base that maintains the state of the conversation. The execution state of a plan is maintained in *actual conversations*. For example, the conversation plan in figure 1 shows how the `Customer` interacts with `Logistics` when the former proposes an order to the latter. After proposing the order, the `Customer-conversation` goes to state `working` where it waits for `Logistics` to either accept, reject or counter propose. If `Logistics` accepts, then the `Customer` waits for the finished order (which can end in success or failure). If `Logistics` counter proposes, a new iteration starts, or the counter proposal is rejected, or clarifications are asked. In each non-final states rules specify how the agent interprets incoming messages, how it updates its status and how it responds with outgoing messages. A conversation plan describes an interaction from the viewpoint of an individual agent (in figure 1 the `Customer`). For two or several agents to "talk", we assume that the conversation plans of each agent generate sequences of messages that the others' conversation plans can process.

*Conversation rules* describe the actions that can be performed when the conversation is in a given state. The rule in figure 2 for example, states that when `Logistics`, in state `start`, receives a proposal for an order from the `Customer`, it should inform the sender that it has started working on the proposal and go to state `order-received`. Note the liberal use KQML-like [Finin et al 92] communicative actions for describing the exchanged messages (the approach assuming that specific types of communicative actions can be freely introduced).

*Error recovery rules* (not illustrated) specify how incompatibilities (caused by planning or execution flaws) among the state of a conversation and the incoming messages are handled: for example by changing the state, discarding inputs, changing the plan, starting new conversations, etc.

*Control.* The framework also defines mechanisms by which agents can carry out many conversations in parallel and a more complex typology of rules includ-

```
(def-conversation-rule 'lep-1
  :current-state 'start
  :received '(propose
              :from (customer Customer)
              :content(customer-order
                        :has-line-item ?li))
  :next-state 'order-received
  :transmit '(tell :from ?agent :to customer
                   :content '(working on it)
                   :conversation ?convn)
  :do '(update-var ?conv '?order ?message))
```

**Fig. 4.** Conversation rule.

ing various forms of event/condition triggered rules. But of particular interest here is the mechanism allowing a conversation associated with a goal to be suspended (with preserved state), and conversations for the subgoals of this goal to be initiated and completed (provided the deontic state of the subgoals allows it). When the child conversations reach a state satisfying certain conditions (e.g. termination in some state), the parent conversation can be resumed and will be given access to the state of the children (for example to retrieve variables or check the state). In this way plans for supergoals can be writen to make use of plans for subgoals while in the same time enforcing that deontic propagation selects those goals that can be executed. All these provide flexible control handles allowing the use of conversations as generalized plans and processes that capture both interaction and local processing.

## 5 Decision Theoretic Planning

Conversations can be mapped to fully-observable, discrete-state Markov decision processes (MDP) [Bellman 57]. In this mapping, conversation states become MDP states and conversation rules become MDP actions. Let $S$ be the set of states and $A$ the set of actions of a conversation plan. For each action (rule) $a \in A$ we define the probability $P(s, a, t)$ that action $a$ causes a transition to state $t$ when applied in state $s$. In our framework, this probability quantifies the likelihood of the rule being applicable in state $s$ and that of its execution being successful. For each action (rule), its reward (a real number) denotes the immediate utility of going from state $s$ to state $t$ by executing action $a$, and is written as $R(s, a, t)$. Since conversation plans operate for indefinite periods of time, we use the theory of infinite horizon MDP-s. A (stationary) policy $\pi : s \rightarrow A$ describes the actions to be taken by the agent in each state. We assume that an agent accumulates the rewards associated with each transition it executes. To compare policies, we use the *expected total discounted reward* as the criterion to

optimize. This criterion discounts future rewards by rate $0 \leq \beta < 1$. For any state $s$, the value of a policy $\pi$ is defined as:

$V_\pi(s) = R(s, \pi(s), t) + \beta \sum_{t \in S} P(s, \pi(s), t) V_\pi(t)$

The value of $\pi$ at any state $s$ can be computed by solving this system of linear equations. A policy $\pi$ is optimal if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in S$ and all policies $\pi'$. A simple algorithm for constructing the optimal policy is value iteration [Bellman 57], guaranteed to converge under the assumptions of infinite horizon discounted reward MDP-s.

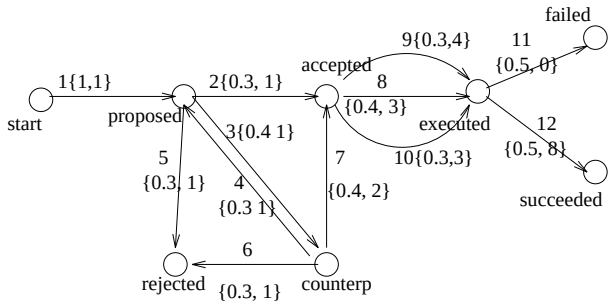


**Fig. 5.** Using value iteration to reorder rules.

The application of this theory to conversation plans is illustrated in figure 3. With each rule number we show the probability and the reward associated to the rule. We use value iteration to actually order the rules in a state rather than just computing the best one. The result is the reordering of rules in each state according to how close they are to the optimal policy. Since the rules are tried in the order they are encountered, the optimal reordering guarantees that the system will always try the better behavior first. Of course, there are several reward structures corresponding to different criteria, like *solution accuracy* or *execution time*. To account for these, we produce a separate ordering for each criterion. Then a weighted combination of criteria is used to produce the final ordering. For example, if we have spent too much time in the current plan, when entering a new state we modify the global criterion giving *execution time* a greater weight. This dynamically reorders the rules in the current state, giving priority to a rule that saves time and thus achieving adaptive behavior of the agent.

## 6 Back to the Supply Chain

One typical round of interactions starts with the `Customer` sending an RFQ about some order to `Logistics`. To answer it, `Logistics` sets up an appropri-

ate run of its scheduling software that decomposes the order into parts doable by the production units in the network and also provides an estimation of whether the order can be executed given the current workload. If the result is positive, `Logistics` tries to obtain tentative agreements from the other production units for executing their part. In this interaction, units are obliged to respond. If the tentative team can be formed, the `Customer` is informed that it can place an order. If this happens (e.g. by using the conversation plan in figure 1), `Logistics` starts another round of interactions in which it asks units to commit to their part of the order. When a unit agrees, it acquires an obligation to execute its part. If everybody agrees, `Logistics` becomes obliged to the `Customer` for execution and the `Customer` to `Logistics` for paying. Then, `Logistics` starts coordinating the actual work by kicking off execution and monitoring its state. Units become obliged to `Logistics` for informing about breakdowns or other events so that `Logistics` can try to replace a unit that can not finish successfully. If breakdowns occur and replacements can not satisfy the initial conditions of the order, `Logistics` tries to negotiate an alternative contract with the `Customer`, e.g. by relaxing some conditions. We usually run the system with 5-8 agents and about 40-60 actual obligations and conversations each. The specification has about 10-20 generic obligations and conversation plans each with about 200 rules and utility functions. The scheduling software is an external process used by agents through an API. All this takes less than 3500 lines of code to describe in our language. We remark the conciseness of the representation given the complexity of the interactions and the fact that the size of this code does not depend on the number of agents and of actual obligations and conversations, showing the flexibility and adaptability of the representation.

## 7 Conclusions and Future Work

We believe the major contribution of this work is a unitary coordination framework and language that goes from the fundamental social constraints like obligations and interdictions to the actual structured conversations by which agents directly interact. At the organization level, social constraints are objective forces determining behavior for both human and artificial agents. As such, social constraints are *necessary* components of any account of organizational behavior. At the individual agent level, obligations and interdictions can be viewed as mental states much like beliefs, desires and intentions. Our framework addresses both levels. By describing obligations and interdictions as relations among organization roles, we use them to shape social behavior. By endowing each agent with an internal representation and an engine for reasoning about their obligations and interdictions we effectively integrate obligations and interdictions with the agent's beliefs, goals and plans, extending the BDI model in a new direction.

Social constraints have been addressed to some extent previously. [Werner 89] describes a theory of coordination within social structures built from roles among which permissions and responsibilities are defined. [Shoham and Tennenholtz 95] study the general utility of social laws. [Castelfranchi 95] stresses the importance

of obligations in organizations but does not advance operational architectures. AOP [Shoham 93] defines obligations locally, but does not really exploit them socially. [Krogh 96] argues for the necessity of artificial agents with normative positions in today's Internet world.

Up to now, our focus has been on prototyping our ideas into systems that can be quickly evaluated and "falsified" in applications. Most important evaluations include a suite of several supply chain coordination projects. One of these deals with dynamic team formation and monitoring (as previously illustrated). Others attempt to capture the dynamic behavior of more realistic supply chains in a combined simulation/control fashion and to study the coordination mechanisms that help atenuate the perturbations and distraction caused by unexpected events and uncertainities. In all situations, the coordination language enabled us to quickly prototype the system and build running versions demonstrating the required behavior. Often, an initial (incomplete) version of the system has been built in a few days, enabling us to immediately demonstrate its functionality. Moreover, we have found the approach explainable to and usable by industrial engineers interested in modeling manufacturing processes. The most interesting experience in this sense is our latest supply chain system consisting of about 40 agents modeling a realistic enterprise that has several plants, distribution centers and transportation facilities. This system is being developed by an industrial engineer without prior programming experience. In spite of that, a prototype able to simulate the supply chain on a 100-150 weeks horizon during which thousands of plan executions take place has been built in about 3 months.

As for future work, it is clear that a system like this also needs clear semantics. Very briefly, one thing we are looking at is using the adopted reduction of deontic logic to dynamic logic to extend the Cohen-Levesque definitions of commitments etc. For example, a local commitment imposed by an obligation is defined as:

(O-goal x y p) $\equiv$
1. (O x y p) $\wedge$
2. (BMB x(intend y (later(done x p)))) $\wedge$
3. (bel x $\neg$p) $\wedge$
4. (goal x (later p)) $\wedge$
5. (know x (prior
(a) ((MB x y $\neg$(intend y (later(done x p)))) $\vee$
(b) ((bel x p)$\wedge$(goal x $\diamond$(bel y p))) $\vee$
(c) ((bel x (always $\neg$p))(goal x $\diamond$(bel y (always $\neg$p)))))
$\neg$(goal x (later p)))))

The new conditions for the O-goal to occur include the existence of an obligation and the obliged agent believing that there's a mutual belief that the agent in authority wants him to achieve the goal. To drop an O-goal, a new possibility is added, the agent in authority can relieve the obliged agent from the obligation.

Other future work includes workflow modeling [Medina-Mora et al 92] as well as using the representation of OPI-s to build "explicable" agents that provide services in a telecommunications organization. In the latter case, besides actively reasoning about their obligations and interdictions when requested to provide

services to internal or external customers, agents also use their representations to explain their decisions to users and to their own human developers.

## 8 Acknowledgments

Tom Gray and Serge Mankovski of Mitel Corp. contributed ideas regarding the practical applications of this work. This research is supported, in part, by the Manufacturing Research Corporation of Ontario, Natural Science and Engineering Research Council, Digital Equipment Corp., Mitel Corp., Micro Electronics and Computer Research Corp., Spar Aerospace, Carnegie Group and Quintus Corp.

## References

[Barbuceanu & Fox 96] Barbuceanu, M. and Fox, M. S. 1996. Capturing and Modeling Coordination Knowledge for Multiagent Systems. *International Journal of Cooperative Information Systems*, Vol.5, Nos. 2 & 3 275-314.

[Bellman 57] Bellman, R. E. 1957. *Dynamic Programming.* Princeton University Press, Princeton.

[Castelfranchi 95] Castelfranchi, C. 1995. Commitments: From Individual Intentions to Groups and Organizations. In Proceedings of ICMAS-95, AAAI Press, 41-48.

[Cohen & Levesque 90] Cohen, P. R. and Levesque, H. 1990. Intention is Choice with Commitment. *Artificial Intelligence* 42, 213-261.

[Cohen & Levesque 91] Cohen, P. R. and Levesque, H. 1991. Teamwork. *Nous* 15, 487-512.

[Finin et al 92] Finin, T. et al. 1992. Specification of the KQML Agent Communication Language. The DARPA Knowledge Sharing Initiative, External Interfaces Working Group.

[Krogh 96] Krogh, K. 1996. The Rights of Agents. In M. Wooldridge,J.P. Muller and M. Tambe (eds) Intelligent Agents II, Agent Theories, Architectures and Languages. *Lecture Notes in AI 1037*, 1-16, Springer Verlag.

[Meyer 88] Meyer, J. J. Ch. 1988. A Different Approach to Deontic Logic: Deontic Logic Viewed as a Variant of Dynamic Logic. *Notre Dame J. of Formal Logic* 29(1) 109-136.

[Medina-Mora et al 92] Medina-Mora, R., Winograd, T., Flores, R. and Flores, F. 1992. The Action Workflow Approach to Workflow Management Technology. In CSCW 92 Proceedings, 281-288.

[Shoham 93] Shoham, Y. 1993. Agent-Oriented Programming. *Artificial Intelligence* 60, 51-92.

[Shoham and Tennenholtz 95] Shoham, Y. and Tennenholtz, M. 1995. On Social Laws for Artificial Agent Societies: Off-line Design. *Artificial Intelligence* 73 231-252.

[Werner 89] Werner, E. 1989. Cooperating Agents: A Unified Theory of Communication and Social Structure. In L. Gasser and M.N. Huhns (eds), *Distributed Artificial Intelligence Vol II* 3-36, Pitman.