Coordinating 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

Tom Gray and Serge Mankovski
Mitel Corporation
350 Legget Drive, P.O. Box 13089
Kanata, Ontario, Canada K2K 1X3
{tom_gray,serge_mankowski}@mitel.com

Abstract

Organizations constrain the behavior of agents by imposing multiple, often contradictory, obligations and interdictions amongst them. To work in harmony, agents must find ways to satisfy these constraints, or to break less important ones when necessary. In this paper, we present a solution to this problem based on a representation of obligations and interdictions in an organizational framework, together with an inference method that also decides which obligations to break in contradictory situations. These are integrated in an operational, practically useful agent development language that covers the spectrum from defining organizations, roles, agents, obligations, goals, conversations to inferring and executing coordinated agent behaviors in multi-agent applications. One strength of the approach is the way it supports negotiation by exchanging deontic constraints amongst agents. We illustrate this and the entire system with a negotiated solution to the feature interaction problem in the telecommunications industry and a work process coordination example for a manufacturing supply chain.

1 Introduction and Motivation

Working together in harmony requires that everybody fulfils their obligations and respects everybody else's rights. In other words, it requires that everybody respects the social laws of their community. To build agents that can be trusted to work with and on behalf of humans in organizations requires the same thing, that agents know and fulfil their obligations while respecting the rights and authority of humans and of other agents in the organzation. Multiple symultaneous obligations and interdictions require agents to find the right behavior that achieves the goals induced by obligations without violating the interdictions. Often, there is no way to find the right behavior without violating less important obligations or interdictions in order to ensure the more important ones are fulfilled. Current models of collective behavior often oversimplify this situation. The the Cohen-Levesque account of teamwork [8] for example and the implemented systems based on it [10, 16] assume that all members of a team have essentially a single obligation towards a common mutual goal.

In this paper we present a solution to building agents that can find the right behavior in face of multiple, possibly conflicting obligations and interdictions. The solution relies on (1) a representation - semantically founded on dynamic deontic logic - of social laws as obligations, permissions and interdictions among the roles that agents play in an organization and (2) a constraint propagation reasoning method allowing agents to infer the applicable obligations and to decide among conflicting ones. This is presented in Section 2. The approach is fully implemented and operational, being integrated in a coordination language that supports agent development along the entire spectrum from organization and role specification, definition of social obligations and interdictions, agent construction, proactive and interactionist agent behavior according to the applicable social laws and to the agent's own conversation plans. This is presented in Section 3. From the application perspective, one consequence of the approach is the way it supports negotiation as exchange of obligations and interdictions among agents. In Section 4 we illustrate this with an agent negociated solution to the feature interaction problem in the telecommunications industry, one of the industries we work with directly in applying our system. A second application, in work process coordination for distributed supply chains, is reviewed in Section 4. We end with conclusions, a review of related work and future work hints.

2 Representing and Reasoning about Obligation

Intuitively, an agent a1 has an obligation towards an agent a2 for achieving a goal G iff the non-performance by a1 of the required actions allows a2 to apply a sanction to a1. The sanction is expressed as a cost or loss of utility. 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).

Semantics. We model obligations, permissions and interdictions (OPI-s) using the reduction of deontic logic to dynamic logic due to [13] in a multi-agent framework. Briefly, we define obligation, interdiction and permission as follows, where V_α^{ij} denotes a violation by i of a constraint imposed by j wrt action or goal α (associated with a cost to be paid):

- $F^{ij}\ \alpha \equiv [\alpha]^i V_\alpha^{ij}$: i is forbidden by j to execute α. An agent is forbidden to do α iff in any state resulting after executing α the violation predicate holds.

Autonomous Agents 98 Minneapolis MN USA
Copyright 1998 0-89791-983-1/98/ 5...$5.00

• $P^{ij}\ \alpha \equiv \neg F^{ij}\ \alpha$: i is permitted by j to execute α. Permission is the same as non-interdiction.

• $O^{ij}\ \alpha \equiv F^{ij}(\text{-}\alpha)$: i is obliged by j to execute α. Obligation is an interdiction for the negation of the action (forbidden not to do α).

As shown by [13], this reduction eliminates the paradoxes that have plagued deontic logic for years and moreover, leads to a number of theorems which, as will be shown immediately, allow us to apply an efficient constraint propagation method to reason about OPI-s in action networks. Both of these are essential for applying this model to real applications.

The main theorems that we use are as follows (indices dropped for clarity), where ; denotes sequential composition, $\cup$ nondeterministic choice and & parallel composition of actions.

$\models$ F($\alpha;\beta$) $\equiv$ [α]Fβ (1)
$\models$ F($\alpha \cup \beta$) $\equiv$F$\alpha \wedge$ Fβ (2)
$\models$ (F$\alpha \vee$ Fβ) $\supset$ F(α&β) (3)
$\models$ O($\alpha;\beta$) $\equiv$ (O$\alpha \wedge$ [α]Oβ) (4)
$\models$ (O$\alpha \vee$ Oβ) $\supset$ O($\alpha \cup \beta$) (5)
$\models$ O(α & β) $\equiv$ (O$\alpha \wedge$ Oβ) (6)
$\models$ P($\alpha;\beta$) $\equiv$ $<\alpha>$ Pβ (7)
$\models$ P($\alpha \cup \beta$) $\equiv$ (P$\alpha \vee$ Pβ) (8)
$\models$ P(α & β) $\supset$ (P$\alpha \wedge$ Pβ) (9)
$\models$ O($\alpha \cup \beta$) $\wedge$ Fα $\wedge$ Pβ $\supset$ Oβ (10).

In words, these theorems tell us that: (1) A sequence is forbidden iff after executing the first action the remaining subsequence is forbidden. (2) A choice is forbidden iff all components are also forbidden. (3) If at least one component of a parallel composition is forbidden, the parallel composition is forbidden as well. (4) A sequence is obliged iff the first action is obliged and after executing it the remaining subsequence is obliged as well. (5) If at least one component of a choice is obliged, the choice is also obliged. (6) A parallel composition is obliged iff all components are obliged. (7) A sequence is permitted iff there is a way to execute the first action after which the remaining subsequence is permitted. (8) A choice is permitted iff at least one component of it is permitted. (9) If a parallel composition is permitted, then all components must be permitted. (10) If a choice is obliged and one component is forbidden while the other is permitted, then the permitted component is obliged.

Deontic Constraint Propagation. To infer the consequences of given obligations and interdictions and to solve conflicts amongst them we use constraint propagation over acyclic networks in which goals are connected to their subgoals. Figure 1 shows a somewhat arbitrary such network in which g1 is a choice between g2 and g3, g2 is a sequence containing g8 and g9, g3 has g4 and g5 executing in parallel, etc. If the arc connecting a goal to one of its subgoals is labelled with a '-', it means that the subgoal is negated in the goal. In figure 1, g4 is negated in g3, meaning that g3 consists of *not doing* g4 in parallel with doing g5.

Figure 1 illustrates this process using a goal network where we have initially asserted that `(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 theorems listed previously. For example, since g4 is a choice, forbidding it also forbidds all its alternatives, cf. theorem (2). This makes both g8 and g6 forbidden. Since g8 is forbidden, g2 is also forbidden as a sequence with one subaction forbidden, cf. (1). Propagating along supergoals, since g4 is negated in g3, it follows that g3 becomes obliged, having both subgoals obliged, cf. (6) (g5 asserted as such initially, and ~g4 obliged because g4 is

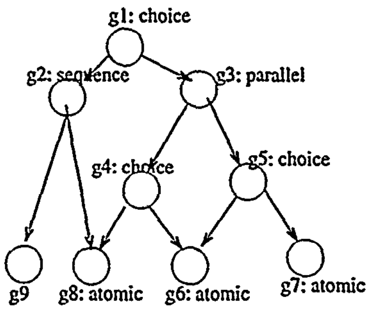


Figure 1: Deontic propagation in a goal network.

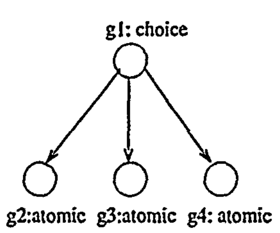


Figure 2: Deontic propagation with costs and conflicts.

forbidden). Next, since g3 is obliged, g1 becomes obliged as well, cf. (5). As g5 has been asserted as obliged and since g6 is forbidden, it follows that g7 must be obliged, cf. (10). We note that g9 has no label, as nothing could be inferred about it.

When goals are asserted as obliged or forbidden, we can also propagate the 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 given qualitative costs. 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 contradictory labelings for g1 and g2, but by comparing the violation costs the agent is justified to accept g1 and g2 as obligatory because thus it will incur a smaller penalty. This scheme works with both *quantitative* and *qualitative* violation costs by means of a cost abstract data type allowing each agent to define the nature of violation costs it uses.

Deontic Propagation Algorithm. The propagation algorithm puts the theorems in section 2 in rule form inside a recursive invocation mechanism. In figure 3 we show one example of such a rule. The rule is activated when (1) a subgoal gi of a choice type goal g has been propagated as forbidden, (2) g is obliged, (3) all its subgoals are forbidden, (4) s0 is the sum of all obligation costs on g (derived from all independent obligations placed on g) and (5)`g-min` and `c-min` are the subgoal with smallest interdiction cost and that cost respectively. In this case, the rule checks whether `c-min` is less than s0. If so, `g-min` becomes obliged (or forbidden if it occurs negated in g). Otherwise, the choice g becomes forbidden with `c-min` as violation cost.

Labelings consist of multiple, independently justified propositions of type `(obliged <goal> <cost>)` or `(forbidden <goal> <cost>)`. These propositions are stored in a LTMS [12] and are justified by the other propositions that make the rules applicable. This allows us to implement non-

```
Propagation-rule: choice-conflict-3
 :when-asserted F(gi)
 :such-that O(g) and
          sum-O-costs(g, sO) and
          all-subgoals-forbidden(g) and
          min-subgoal-F-cost(g, g-min, c-min)
 :propagate if c-min <= sO
          then if negated(g-min, g)
                then F(g-min, sO)
                else O(g-min, sO)
          else F(g, c-min)
```

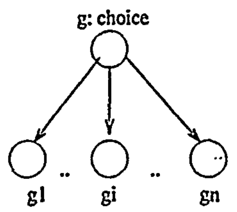
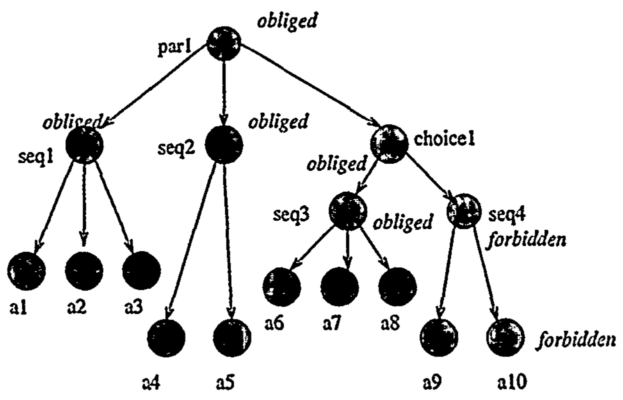


Figure 3: Deontic propagation rule.



Resources and usage: r1(a1, a4, a7), r2(a2, a5, a8, a10), r3(a3, a6)
Action duration: a1(3), a2(4), a3(5), a4(3), a5(5), a6(4), a7(3), a8(3)
a9(4), a10(6)

Figure 4: Action network to be scheduled

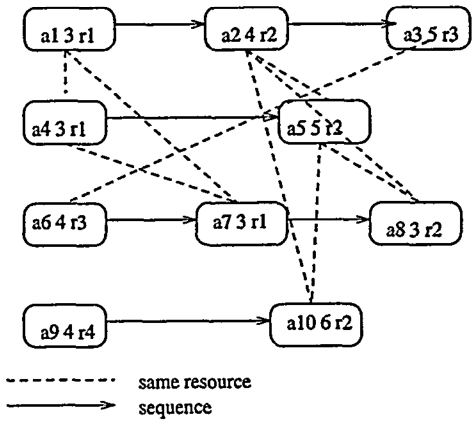


Figure 5: Same problem in a usual scheduling depiction

monotonic reasoning and to provide explanations of every labeling in the system.

The propagation process propagates one input deontic assertion at a time. For each assertion, all goals reachable from the goal of the input assertion along both supergoal and subgoal links are visited at most once. For each visited goal, all rules are checked and those applicable are executed.

Action Scheduling. Knowing which actions are obliged and which are forbidden is not sufficient for execution. The agent also needs to know the relative order of actions and the allowed time windows for execution. These however, depend on different constraints about resource usage and capacity, action duration, time horizons and the execution semantics of composed actions. To deal with these, we first include in the representation of each action (1) an execution time window specified as an interval `[earliest-start, latest-end]`, and (2) a specification of the duration of atomic actions. The time window contains the time limits (in a discrete model of time) within which the action can be executed consistently with all other constraints, and is computed by scheduling as explained immediately. Second, for composed actions, we define the following execution semantics. For parallel actions, all subactions are constrained to be executed within the time-window of the parallel (super)action, with temporal overlapping allowed. For choices, only the chosen (obliged) subactions must be executed within the time-window specified by the choice (super)action, also with overlapping allowed. For sequences, all subactions must be executed within the time-window of the sequence (super)action, without temporal overlapping amongst subactions. Note that these temporal constraints operate outside the deontic propagation framework which assumes that all obligations and interdictions hold for the entite time horizon (including all action time windows). Third, we assume finite capacity of resources, in that a resource can be used by a given finite number of actions at any moment.

Consider now the action network shown in figure 4 with the given durations and associated resource usage (all resources can support only one action at a time). Suppose that the time window for `par1` is given, $time\text{-}window(par1, [2, 17])$. Assume also that `(obliged par1)` and `(forbidden a10)`. Deontic propagation determines `(obliged seq1)`, `(obliged seq2)`, `(obliged seq3)` and `(forbidden seq4)`. But in what order should the agent execute the obliged actions? This can only be determined by scheduling the obliged actions in a way that considers all the resource, duration and order constraints. In our system, we solve the problem by endowing the agent with a constraint based scheduler of the type described e.g. in [3]. Its output is a complete ordering of actions that is consistent with all input constraints, plus the allowed execution time windows for each action, as implied by the ordering. Figure 5 shows the same problem in a normal scheduling depiction, where sequences correspond to jobs, the topmost parallel action sets the time window for every job and each action uses some finite capacity resource and has its own specified duration.

3 The Coordination Language

Having presented the representation and reasoning mechanisms for OPI-s, we now show how these are integrated and used within an implemented, practical coordination language for multi-agent system development.

Organizations, Agents and Roles. 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 speci fic constraining web of mutual obligations, interdictions and permissions - *social constrai nts or laws* - that link him as a

```
(def-organization O1
  :roles ((customer Customer)
          (developer Bob)
          (help-desk-attendant Bob)
          (development-manager Alice)
          (help-desk-manager John)))

(def-agent 'Bob
  :database 'bob-db
  :acquaintances '((Alice development-manager)
                   (John help-desk-manager)))
```

Figure 6: Organizations and agents

```
(def-role 'help-desk-member
  :super-roles '(division-member)
  :min-agents 1)

(def-role 'help-desk-manager
  :super-roles '(help-desk-member)
  :max-agents 1)

(def-role 'help-desk-attendant
  :super-roles '(help-desk-member))
```

Figure 7: Roles

system administrator to developers, managers and every other role and member of the organization. Not fulfilling an obligation or interdiction is sanc tioned by paying a cost or by a loss of utility, which allows an agent to apply rational d ecision making when choosing what to do.

Our coordination language allows organizations to be described as consisting of a set of roles filled by a number of agents. In the example in figure 6 customer, developer, help-desk-attendant etc. are roles filled respectively by agents Customer, Bob, etc.

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 in specific roles, but not necessarily of all of them. Each agent has its local store of beliefs (its database).

A *role* describes a major function together with the obligations, interdictions and permissions attached to it. Roles can be organized hierarchically (for example developer and development-manager would be both development-member roles) and subsets of them may be declared as disjoint in that the same agent can not perform them (like help-desk-member and customer). 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). Some examples are shown in figure 7.

Situations. A situation is a specific combination of occurring events and agent's local state in which the agent acquires obligations and/or interdictions and starts acting in accordance with these. Situations are generically defined in terms of roles, rather than in terms of specific agents. That means that any set of agents that play the specified roles can be involved in the situation, if all conditions are met.

Consider the situation in figure 8. This is a description of a situation in which an agent in the help-desk-attendant

```
(def-situation 'accept-help-desk-work-s
 "attendant must accept work when idle"
 :acting 'help-desk-attendant
 :authority 'help-desk-manager
 :beneficiary 'customer
 :received
  '(request :from (help-desk-manager ?manager)
     :receiver-role help-desk-attendant
     :content do-help-desk-attending)
 :such-that
  '(and(believes ?agent '(now-doing idle)
                 :pspace 'at-work)
      (known-to-me-as ?agent ?manager
                 'help-desk-manager))
 :beliefs-in
  '(list (proposition 'not 'now-doing 'idle)
         (proposition 'requested-hdw ?manager)
         :pspace 'at-work)
 :add-clause
  '(clause
    (conse 'obliged 'accept-hd-work :cost 8)
    (ante 'not 'now-doing 'idle)
    (ante 'requested-hdw ?manager)))
```

Figure 8: A Situation

role (the acting party) acquires an obligation to accept work from the agent in the help-desk-manager role (the authority party). The beneficiary is someone in the client role. According to the definition, this happens when the help desk attendant receives a request in this sense from the help desk manager such that the help desk attendant knows the sender of the request as a help desk manager and the help desk attendant is currently idle. Situations are always described from the viewpoint of the acting party (here the help-desk-attendant). If the above conditions are met, the help desk attendant will add two new beliefs to its context, namely it is not idle anymore and that the help desk manager has requested work. These beliefs justify the agent to believe that it has an obligation to accept the requested work. This is described by the agent creating a new LTMS clause, as shown in the :add-clause slot. To deal with this request from its manager, the agent stores the beliefs relevant to this request in a special propositional space (or :pspace) of its database, named at-work. This enables the agent to differentiate the beliefs related to this request from other beliefs and to reason separately in chosen spaces (context switching). To help with this, spaces can also inherit beliefs from other spaces. Finally, any situation becomes an entry in the agent's agenda, guaranteeing that it will be dealt with by the agent.

Conversation Plans. An agent's possible behaviors in a given situation are described by one or more *conversation plans.* To choose one of them, the agent evaluates specific conditions in the context of the given situation. We borrow the idea of conversation plans from [1], as descriptions of both how an agent *acts locally* and *interacts* with other agents by means of communicative actions. 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 conversation plan is maintained in *actual conversations.*

For example, the conversation plan in figure 9 shows how

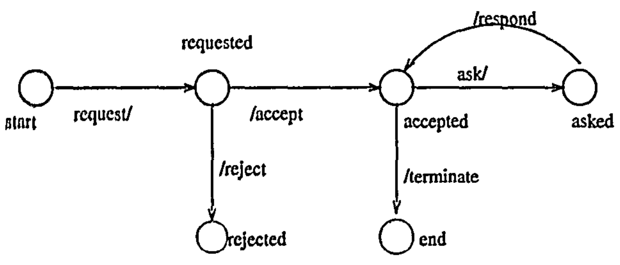


Figure 9: The Customer-conversation

```
(def-conversation-rule 'hda-1
 :current-state 'start
 :received
  '(request :from (customer ?c)
      :content ((assistance PBX setup) ??))
 :next-state 'request-received
 :such-that '(help-desk-attendant-available)
 :transmit '(accept :to ?c
                    :conversation ?convn)
 :do '(update-var ?conv '?customer ?c))
```

Figure 10: Conversation rule

the Customer interacts with the help-desk-attendant when requesting assistance. After making the request for assistance, the customer-conversation goes to state requested where it waits for the help-desk-attendant to either accept or reject. If the help-desk-attendant accepts to provide assistance, the interaction enters an iterative phase in which the Customer asks questions and the help-desk-attendant responds. This cycle can end only when the Customer decides to terminate it. In each non-final state *conversation rules* specify how the agent interprets incoming messages, how it updates its status and how it responds with outgoing messages. The language in which messages are expressed is a liberal form of KQML [9], but any communicative action language is usable. Figure 10 shows an example of conversation rule that a help-desk-attendant would use to respond to a request for assistance.

To see how conversation plans are used to specify agents' behavior in given situations, the conversation plan in figure 11 shows a generic behavior involving both interaction and local reasoning that an agent would use in a situation when it is requested to satisfy a set of obligations and interdictions from another agent. This plan shows how an agent receives a set of obligations and interdictions that another agent requestes it to satisfy, performs deontic propagation (going to state deontic-propagation-done) and, if no requested constraints are violated, plans and schedules the required actions (going to state act-net-done). Then it executes the planned/scheduled actions in rule execute-actions. If no execution failure occurs, the plan ends in state execution-ok, otherwise it ends in execution-failed. If during deontic propagation the agent determines that it can not satisfy some of the requested constraints, or if actions can not be scheduled or planned, the violated constraints may be sent back to the sender for revision.

The Action Executive executes scheduled actions according to the specified time windows. The time windows produced by scheduling satisfy the ordering conditions imposed

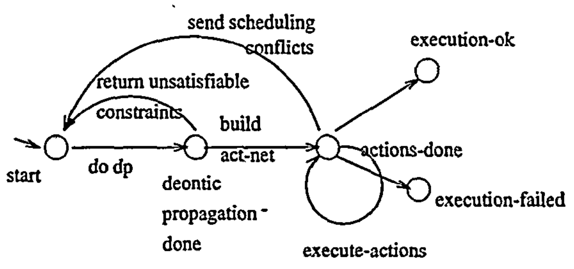


Figure 11: Conversation plan doing deontic propagation, scheduling and action execution

by sequences and parallel compositions (e.g. two consecutive elements of a sequence have time windows that do not allow overlapping, while two elements of a parallel compositions have time windows that may overlap). For this reason, the Executive only needs to pick up atomic actions and choices for execution, making sure time windows are obeyed. When a component action of an obliged sequence is about to be executed, the Executive propagates the component action as obliged and, after executing it, propagates the remaining subsequence as obliged (cf. theorem 4, section 2). This may have as effect new obliged or forbidden actions, in which case we have to reschedule the remaining actions. Similarly, after a component action of a forbidden sequence has been executed, the remaining subsequence is propagated as forbidden (cf. theorem 1, section 2), with the same possible consequences. This shows how much intertwined deontic propagation, scheduling and execution actually are.

To execute an action, either a one-shot method is invocked, or a full conversation plan is initiated. In particular, conversation plans for choices may initiate exchanges with other agents and more complex decision making to determine which alternative to execute (if several are permitted). The architecture allows conversation plans to be suspended in any state, waiting for conditions or events to happen and be resumed when the waited for events or conditions have happened. Conversation plans are always executed incrementally, in a multi-threaded fashion in that each time only at most one state transition is executed, after which the next action is tackled. The Executive is first invocked inside conversation plans tackling situations, as shown in figure 11.

Control Architecture. Each agent operates in a loop where: (1) Events are sensed, like the arrival of messages expressing requests from other agents. (2) Applicable situations are activated updating the agent's beliefs, possibly creating new propositional spaces. (3) Agent selects an entry from the agenda. This is either a new situation, for which a plan is retrieved and initiated, or one that is under processing, in which case its execution continues incrementally, as shown above.

4 Coordination by Exchanging Deontic Constraints

A ubiquitous way in which an agent A can work together with an agent B to achieve some goals in an organization they are both part of is to have A and B request various things from each other and execute each other's requests. A straightforward way for A (say) to request something from B is by formulating the request as a set of things A either wants B to do or wants B to refrain from doing. This list of *do-s* and *don't-s* can be described as a set of obligations and interdictions A wants B to satisfy. The roles of the agents must be such that they are entitled by the laws of the organization to make these requests. Suppose B receives a message from A containing such a set of obligations and interdictions. By propagating these locally, together with its own obligations and interdictions and with other obligations and interdictions it is committed to, B will determine which of them it can satisfy and which it can't. Both sets are then revealed to A, perhaps with some explanations for the reasons for failure attached. A may now revise its request in various ways. It may drop constraints, it may add constraints, or it may raise or lower costs (e.g. to make B violate other constraints). The revised set of constrains is sent back to B, which will repeat the same cycle. The process may end with both agents agreeing on a set of constraints that A still wants and B can satisfy, or may terminate before any agreement is reached. (Many refinements of this mechanism exist, as shown next).

To illustrate the use of this approach, we consider the feature interaction problem, a general service creation problem in the telecommunications industry [5], on which we are working with industrial partners. We assume A and B are agents responsible for establishing voice connections amongst their users. The creation and administration of connections can use various levels of functionality, or *features*, that provide different services to subscribers or the telephone administration.

Here are a few examples for the features that are usually available (modern telecommunication services may have many hundreds of such features): (1) *Incoming Call Screening*: the calee will refuse all calls from callers in an incoming call screening list. (2) *Call Forward*: the calee will forward the incoming call to another number. (3) *Recall*: if the calee is busy, the caller will be called back later when the calee becomes available. (4) *Outgoing Call Screening*: the caller does not allow to be connected to some specified directory numbers.

The feature interaction problem is that often combinations of features interact in undesired ways, affecting the intended functionality of the provided services. For example, several combinations of the above features may interact in an undesired fashion. *Incoming Call Screening* and *Recall* may conflict if *Recall* is done without checking that the number belongs to the incoming call screening list - we shouldn't call back numbers that are not accepted in the first place. Similarly, *Call Forward* and *Outgoing Call Screening* may conflict if a caller is forwarded to a number that it does not wish to be connected to.

The deontic propagation framework can be used to solve such interactions in a principled manner. When agent A wishes to connect to agent B, it sends to B a set of constraints that specify A's relevant features that B must consider. For example, if A has *Outgoing Call Screening*, it will send to B a list of interdictions about the numbers that it doesn't want to be connected to. If B has *Call Forward*, A's interdictions will be used to forbid forwarding to A's undesired numbers.

For illustration, figure 12 shows the inferences performed by a calee B when receiving a call from A. A has *Outgoing Call Screening* for number #1, and B has *Incoming Call Screening* with A in its incoming call screening list (meaning B does not want to talk to A directly). The set of constraints that A sends to B is {(obliged accept-call :from A :cost 5) (forbidden forward :from A :to #1 :cost 9)}.

In response to this message, a situation becomes applicable within B that posts an obligation for B to execute Process Incoming Call. A generic plan that can be used in this situation (shown in figure 11) performs deontic propagation, schedules and then executes the action network. Incoming

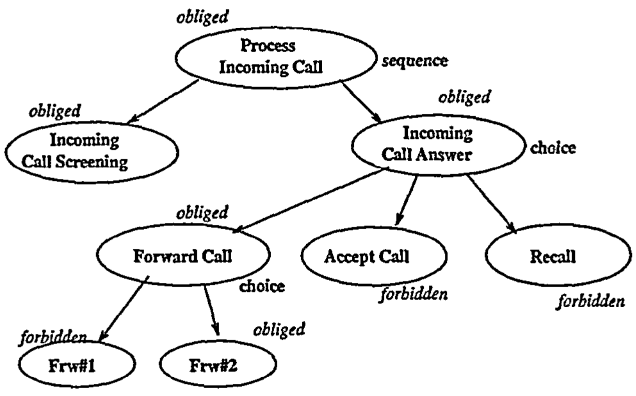


Figure 12: Deontic propagation applied to feature interaction.

Call Screening is scheduled first and executed by retrieving and activating a plan for it. The plan places an interdiction for Accept Call and Recall in the current context, because A is on the black list. Deontic propagation activated again by the plan infers that Forward Call is obliged. As this is a choice whose first subgoal Frw#1 (forward to #1) is forbidden by the caller, deontic propagation also infers that the second subgoal, Frw#2 (forward to #2) is obliged. That leaves Frw#2 as the only obliged subgoal in the action network. Its execution completes the execution of the action network. In conclusion, we have obtained the desired behavior: B does not accept the call and does not set a callback to A if busy. B forwards the call to its second number, because the first is not acceptable to A.

Many other useful negotiation strategies based on this framework can be conceived.

1. Suppose first that A has Outgoing Call Screening to both #1 and #2. In this case A sends to B interdictions for both numbers. Deontic propagation will reveal that B can not forward the call (theorem 2) as both alternatives of the choice are now forbidden. B will then reply with a meesage in which it states that the interdictions on both #1 and #2 can not be satisfied. A is then free to choose: either it drops one of these or it drops the entire call. Alternatively, if A has enough authority it may increase the cost of obliging B to accept the call, in order to force B to override its own interdiction of talking to A directly. In this case B would have to accept the call and, if busy, commit to recalling.

2. Assume now that A does not want B to know that it does not want to be forwarded to #1. In this case, A will not send the corresponding interdiction. B will have to choose between forwarding to #1 and #2. By modifying the protocol, we can have B inform A that it can forward the call to either #1 or #2. If there are only two alternatives (like in this case), A may call the desired one directly, without asking B to forward the call, and thus making sure B has no idea where the call went. If there are more than 2 alternatives, A can explicitely request to be forwarded to one of them, making it uncertain for B which number was avoided. In the former case, A may pay for the call itself, while in the latter B normally pays. Also, in this case it is B who discloses more information to A regarding its forwarding possibilities.

3. Finally, we can have A send together with its constraints some utility value for each of the constraint, thus informing B about its preferences about which constraints should be satisfied first. For example, there may be a utility of 0.8 for forwarding to #2 and only 0.2 for forwarding to #1 (we have assumed that A knows where its call may be forwarded, which is not necessarily the case). B may then try to satisfy those constraints that maximize A's total utility. In this case A does not have enough authority to force B to take a given action, but relies on B's goodwill to satisfy A's request as much as possible. One conclusion we draw from these examples is that the space of coordination patterns that can be described and executed with this framework is very large and more work is needed to explore and exploit it.

5 Coordinating Teamwork in the Supply Chain

The second application deals with work coordination in the supply chain of global enterprises. The supply chain of a modern enterprise 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 to achieve coordinated behavior in dynamic systems of this kind by applying our agent coordination technology.

We have built several demonstration systems addressing supply chain issues. In this his section we briefly review one of them, focused on modeling team formation, team monitoring and team modification. The supply chain in this case consists of a `Customer` agent, a `Logistics` agent coordinating the joint effort and several plants and transportation agents that participate in the joint work. One typical round of interactions in this setup starts with the `Customer` sending an RFQ (request for quotation, in which an inquiry is made about the cost of an order) to `Logistics`. To answer it, `Logistics` sets up an appropriate run of its (local) 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, `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 actual obligations and conversations, showing the flexibility and adaptability of the representation.

6 Conclusions, Related and Future Work

We believe we have demonstrated the feasibility of agents that can represent social constraints in the form of obligations and interdictions and that can efficiently reason about them to find courses of action that either do not violate them or violate them in a 'necessary' way as imposed by the participating agents authorities or priorities. We have shown how a representation of obligation founded on dynamic deontic logic and a constraint propagation inference method for it can be integrated in a *practically useful* agent development language that covers the spectrum from the definition of organizations, roles, agents, obligations, goals, conversations, to inferring and executing actual, coordinated agent behaviors in applications. One consequence of the approach is that it allows agents to be 'talked to' by giving them a list of obligations and interdictions and then trusting the agent to figure out how to actually fulfil them, including which of them to break if necessary! In particular, agents can talk to each other in this way, leading to a clean approach to negotiation which we have illustrated in the context of service provisioning in telecommunications.

Social constraints have been addressed to some extent previously. [18] describes a theory of coordination within social structures built from roles among which permissions and responsibilities are defined. [15] study the general utility of social laws. [6] stresses the importance of obligations in organizations but does not advance operational architectures. AOP [14] defines obligations locally, but does not really exploit them socially. [11] argues for the necessity of artificial agents with normative positions in today's Internet world. [4] uses norms for enhancing the decision making capability of agents, but not for enhancing coordination and negotiation. Finally, [2] schetches an architecture for using obligations in coordination, but without the basic deontic constraint propagation process described here and without specifying how deontic conflicts can be addressed using violation costs.

Up to now, our focus has been on prototyping our ideas into systems that can be quickly evaluated and "falsified" in applications. Evaluations based on several supply chain systems (of which we have reviewed one) as well as on service provisioning systems for telecommunications have shown that 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 and other engineers interested in their own domain. For example, our latest supply chain system consisting of about 40 agents modeling a realistic enterprise that has several plants, distribution centers and transportation facilities has been 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 took place has been built in about 3 months.

As future work, on the application side, we are currently exploring the space of negotiation strategies based on the deontic constraint exchange approach. In telecommunications for example, we are using the flexibility allowed by the system in describing such strategies to customize the services provided (e.g. if a caller does not want calees to know that it does not want to be forwarded to a certain number, we will modify the service to behave as required). Also, we are extending the application of the approach to global supply chain management addressing the management of many horizontal and vertical levels of interaction among agents with many different rights and obligations. On the system development side, we are working on the direct integration of time in the deontic propagation process, allowing to infer obligations and interdictions for specific time intervals, rather than assuming that all obligations apply to the entire horizon. Another extension that would strenghten the system is using first principles planners e.g. like [7, 17] to construct sequences of actions for achieving goals. The agent may initially have incomplete sequences (containing for example only the initial and final action as required by UCPOP [17], or an abstract one, as allowed by O-PLAN [7]) that the agent's planner would refine into a complete plan, maybe also looking at which actions are permitted from a social laws point of view. This would lead to a more complete architecture that integrates social laws reasoning with classical planning and scheduling.

7 Acknowledgments

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

[1] Barbuceanu, M. and Fox, M. S. 1997. Integrating Communicative Action, Conversations and Decision Theory to Coordinate Agents. *Proceedings of Automomous Agents'97*, 47-58, Marina Del Rey, February 1997.

[2] Barbuceanu, M. 1977. Coordinating Agents by Role Based Social Constraints and Conversation Plans. *Proceedings of AAAI'97*, 16-21, Providence, RI.

[3] Beck, C. et al. Texture-Based Heuristics for Scheduling Revisited. *Proceedings of AAAI-97*, 241-248, July 1997, Providence RI.

[4] Boman, M. 1997. Norms as Constraints on Real-Time Autonomous Agent Action. In *Multi-Agent Rationality* (Proceedings of MAAMAW'97), Boman and Van de Welde (eds) Springer Verlag.

[5] Cameron, E.J., N.D. Griffeth, Y.J. Lin, M.E. Nilson, W.K. Schnure, and H. Velthuijsen. A Feature Interaction Benchmark for for IN and Beyond. In L.G. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunication Systems*, 1-23, Amsterdam, May 1996. IOS Press.

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

[7] Currie, K and Tate, A. O-Plan: The Open Planning Architecture. Artificial Intelligence 52(1):49-86, November 1991.

[8] Levesque, H, Cohen, P.R. and Nunes, J. On Acting Together. *Proceedings of AAAI'90*, Menlo Park, CA.

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

[10] Jennings, N. 1995. Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems Using Joint Intentions. *Artificial Intelligence* 75.

[11] 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.

[12] McAllester, D. 1980. An Outlook on Truth Maintenance. Memo 551, MIT AI Laboratory.

[13] 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.

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

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

[16] Tambe, M. 1997. Agent Architectures for Flexible, Practical Teamwork. *Proceedings of AAAI'97*, Providence, RI, 22-28.

[17] Weld, D.S. An Introduction to Least Commitment Planning. AI M agazine, Summer/Fall 1994.

[18] 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.