Agents That Work in Harmony by Knowing and Fulfiling Their 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

Societies 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[1], 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.

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 Cohen-Levesque account of teamwork (Levesque, Cohen & Nunes 90) for example, and the implemented systems based on it (Jennings 95; Tambe 97) assume that all members of a team have essentially a single, implicit, obligation towards a common mutual goal.

In this paper we build on a different model of social interaction, one that explicitely represents and integrates multiple obligations.

1. At the social level, our model assumes that societies and organizations constrain the social behavior of agents by imposing social laws, representable as networks of mutual obligations and interdictions amongst agents. Not fulfiling 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 motivating social behavior and to a large extent determine the 'attitudes' 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.
2. At the individual agent decision level, each agent decides what behavior to adopt to satisfy the applicable social laws as well as its own goals and priorities. In particular, at this level agents determine how to solve conflicting obligations and interdictions.
3. Having decided on the general behavior in terms of what to do or not, agents need to plan/schedule the activities that compose the selected behavior. This determines the precise sequencing of actions to be executed, consistent with time, resource and possibly other constraints on action execution.
4. Finally, actions have to be executed as planned, with provisions for handling exceptions and violations. These may be dealt with at any of the above levels, for example by retrying, replanning, deciding on different actions or even (in an extreme case that we do not deal with) trying to modify the social laws.

To integrate obligations in this framework we rely 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 obliga-

tions and to decide among conflicting ones. 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. An important consequence of the approach is the way it supports negotiation as exchange of obligations and interdictions among agents. We illustrate this, and the entire system, with an agent negotiated solution to the feature interaction problem in the telecommunications industry, one of the industries we work with directly in applying our system. We end with conclusions, a review of related work and future work hints.

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 (Meyer 88) 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.
- $P^{ij}\ \alpha \equiv \neg F^{ij}\ \alpha$: 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 (Meyer 88), 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, are the first step toward applying an efficient constraint propagation method to reason about OPI-s in action networks. Both of these are necessary 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.

$\models \mathrm{F}(\alpha;\beta) \equiv [\alpha]\mathrm{F}\beta$ (1)
$\models \mathrm{F}(\alpha \cup \beta) \equiv \mathrm{F}\alpha \wedge \mathrm{F}\beta$ (2)
$\models (\mathrm{F}\alpha \vee \mathrm{F}\beta) \supset \mathrm{F}(\alpha \& \beta)$ (3)
$\models \mathrm{O}(\alpha;\beta) \equiv (\mathrm{O}\alpha \wedge [\alpha]\mathrm{O}\beta)$ (4)
$\models (\mathrm{O}\alpha \vee \mathrm{O}\beta) \supset \mathrm{O}(\alpha \cup \beta)$ (5)
$\models \mathrm{O}(\alpha \& \beta) \equiv (\mathrm{O}\alpha \wedge \mathrm{O}\beta)$ (6)
$\models \mathrm{P}(\alpha;\beta) \equiv < \alpha > \mathrm{P}\beta$ (7)
$\models \mathrm{P}(\alpha \cup \beta) \equiv (\mathrm{P}\alpha \vee \mathrm{P}\beta)$ (8)
$\models \mathrm{P}(\alpha \& \beta) \supset (\mathrm{P}\alpha \wedge \mathrm{P}\beta)$ (9)
$\models \mathrm{O}(\alpha \cup \beta) \wedge \mathrm{F}\alpha \wedge \mathrm{P}\beta \supset \mathrm{O}\beta$ (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.

While providing an understanding of what OPI-s are in a dynamic framework where agents' behavior can be described with sequential, parallel and choice compositions, this model does not allow to compare obligations in conflicting situations. Next, we show (1) how the model can be given a constraint propagation formulation and (2) how in this format it can be extended to handle conflict resolution.

Deontic Constraint Propagation. We start with representing possible behavior as acyclic networks where nodes represent goals, and arcs relate goals to subgoals. Figure 1 shows a somewhat arbitrary such network in which `par1` and `par2` are parallel goals, `seq1` to `seq4` are sequences, `g1` to `g5` are atomic and `ch1` is a choice. All subgoals of `ch1` except `g2` are negated in `ch1`, shown by having their connecting arcs to `ch1` labeled with a '-'. That means that `ch1` is a choice between *not doing* `g1`, doing `g2`, not doing `g3`, etc.

Assume we have initially asserted `(forbidden ch1)` and `(obliged par2)`. For each of these assertions the propagation process traverses the network along supergoal and subgoal links and applies the theorems listed previously. Thus, `(obliged par2)` implies `(obliged seq3)` and `(obliged seq4)` cf. theorem (6). Forbidding `ch1` makes each component forbidden, cf. (2). But as `g1` is negated in `ch1`, this means `-g1` is forbidden, hence `g1` is obliged, and similarly `g3, g4, g5`. Since `g2` is forbidden (being non-negated in `ch1`) it follows that `seq1` and `seq2` will become eventually forbidden, cf. (1). Then `par1` will also become forbidden, cf. (3).

Integrating violation costs. With violation costs, the propagation process can no longer be described by Meyer's theorems in their given form. The purpose of

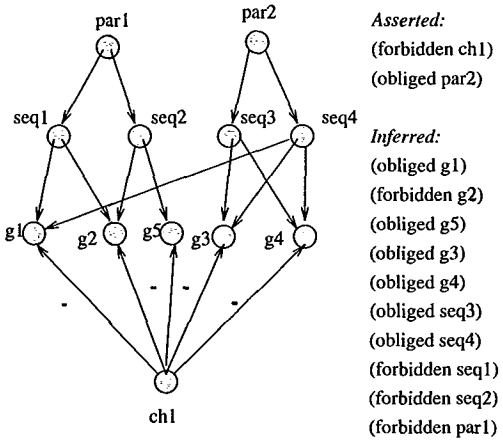
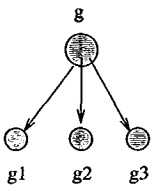


Figure 1: Deontic propagation in a goal network.

violation costs is to allow us to compare obligations and, in conflicting cases, select those that incur a smaller cost to pay. Consider figure 2 and assume g is a choice. Asserting all its subgoals forbidden with violation costs as shown, results in propagating g as forbidden (cf. theorem (2)) with a violation cost that is the *minimum* of the violation costs of subgoals (in our case `low`). This *minimum cost propagation rule* (from subgoals to supergoal) is justified because to execute a choice, at least one subgoal must be executed. If g is asserted as obliged with a cost greater than the minimum interdiction cost of subgoals, then at least the minimum cost subgoal can be turned to obliged with the obligation cost of g. In figure 2, since g is obliged with cost `med`, there is only one subgoal with a smaller interdiction cost, `g1`, and this is turned into obliged with cost `med`, thus solving the conflict by having the agent execute `g1`. If there is more than one subgoal with smaller interdiction costs than the obligation cost of g, any of them are executable as part of the choice, and the agent has freedom to choose from among them.

Assume now g is a parallel goal. To execute a parallel goal, all subgoals must be executed. Thus, if some subgoals are forbidden, (which would make g forbidden as well cf. (3)) g must be obliged with a cost higher than the *maximum* interdiction cost of subgoals in order to become overall obliged, and in this case all forbidden subgoals become obliged with this cost. This situation justifies the *maximum cost propagation rule* (from subgoals to supergoal), and is illustrated in the figure. If g was a sequential goal, then the previous propagation would have taken place in the same way, using the maximum cost propagation from subgoals to supergoal. But in this case the execution semantics would be quite different in that sequences can not be executed with time overlapping of their subgoals, while parallel goals can. This is explained later on, when we address the issue of action scheduling.

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.



g is choice	g is parallel
Asserted:	*Asserted:*
(forbidden g1 low)	(forbidden g1 low)
(forbidden g2 high)	(forbidden g2 med)
(forbidden g3 high)	(obliged g high)
(obliged g med)	
Inferred:	*Inferred:*
(forbidden g low)	(forbidden g med)
(obliged g1 med)	(obliged g1 g2 g3 high)

Figure 2: Deontic propagation with costs and conflicts.

Deontic Propagation Algorithm. The propagation algorithm uses Mayer's theorems extended with cost propagation as above, formulated as a collection of propagation rules inside a recursive invocation mechanism. In figure 3 we show one example of such an extended propagation 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), (5)`g-min` and `c-min` are the subgoal with smallest interdiction cost and that cost respectively and finally (6) `g-min` is the only subgoal whose interdiction cost is smaller than `s0`. In this case, `g-min` becomes obliged or, if it occurs negated in g, forbidden.

Labelings consist of multiple, independently justified propositions of type `(obliged <goal> <cost>)` or `(forbidden <goal> <cost>)`. These propositions are stored in a LTMS (McAllester 80) and are justified by the other propositions that make the rules applicable. This allows us to implement non-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. A goal is never visited more than once for the same initial assertion because any subsequent visit would produce either identical labelings or incorrect ones, circularly justified.

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 ac-

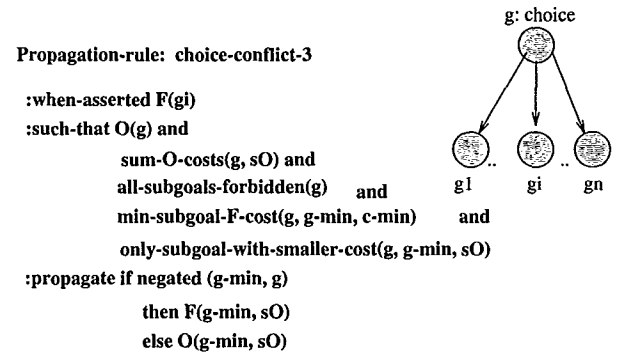
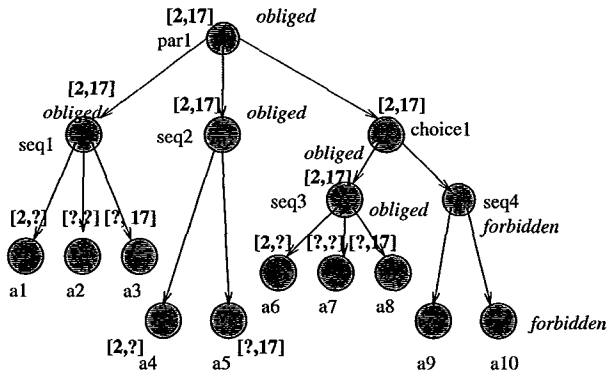


Figure 3: Deontic propagation rule.



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

tions. 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, $time - window(par1, [2, 17])$. 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 (Beck 97). 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.

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 `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 5 `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

```
(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)))

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

Figure 5: Organizations, agents and roles

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`).

Situations. A situation is a specific combination of occuring 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 6. This is a description of a situation in which an agent in the `help-desk-attendant` 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,

```
(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 6: A Situation

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 (Barbuceanu & Fox 97), 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 7 shows how 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 lib-

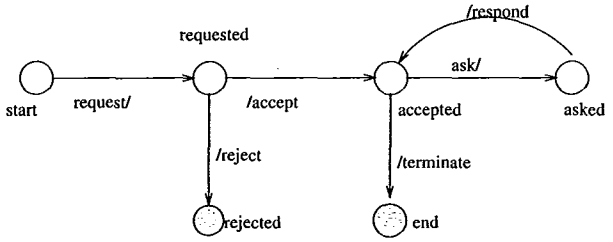


Figure 7: The Customer-conversation

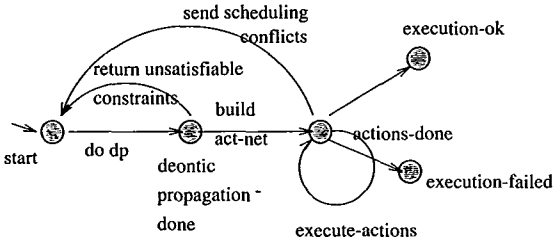


Figure 8: Conversation plan doing deontic propagation, scheduling and action execution

eral form of KQML (Finin 92), but any communicative action language is usable.

To see how conversation plans are used to specify agents' behavior in given situations, the conversation plan in figure 8 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 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 invoked, 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 invoked inside conversation plans tackling situations, as shown in figure 8.

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.

Coordination by Exchanging Deontic Constraints

A basic building block of social interaction is the ability of agents to request things from other agents and to execute other agents requests. Requests normally consist of things an agent wants another to do or to refrain from doing. Thus, they can be described as sets of obligations and interdictions an agent wants another to satisfy. 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.

To illustrate the use of this approach, we consider the feature interaction problem, a general service creation problem in the telecommunications industry (Cameron 96), 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. In our example, *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 9 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 8) performs deontic propagation, schedules and then executes the action network. `Incoming 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

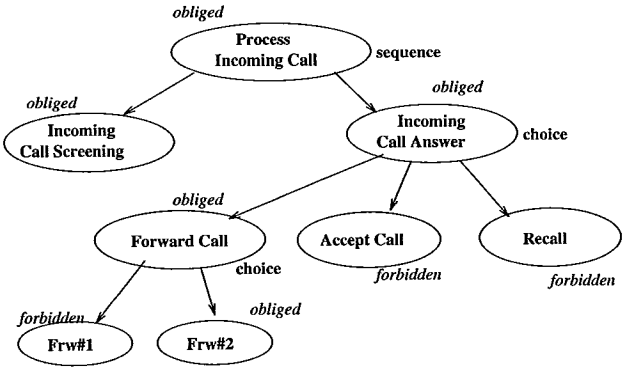


Figure 9: Deontic propagation applied to feature interaction.

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.

Suppose now 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.

Conclusions and Related 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 an initially 'theoretical' representation of obligation founded on dynamic deontic logic could be extended with violation costs, reformulated as a constraint propagation method and 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. In discourse modeling, (Traum &Allen 94) make a strong case for the need of obligations, together with intentions and goals, to understand social interaction. (Jameson & Weis 95) extended their approach with penalty costs and time limitations on obligations. None of them uses the principled deontic constraint propagation method we use, and the requirements of natural language dialogue are different (more complexity, more dynamic conversation turns, but also less quality of service guarantees) from those of artificial agent interaction. (Werner 89) describes a theory of coordination within social structures built from roles among which permissions and responsibilities are defined. (Shoham & Tennenholtz 95) study very general computational properties 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. (Boman 97) uses norms to improve local decison making, but not coordination.

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 looking at strategies that limit and control the amount of information disclosed in exchanges (e.g. if the caller does not want the calee to know that it does not want to be forwarded to a certain number, how should the negotiation proceed?). Also, we are applying the approach to global supply chain management where many horizontal and vertical levels of interaction have to be managed 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. By further allowing action sequences to be produced by planning systems we aim at a more complete architecture that integrates social laws reasoning with classical planning and scheduling.

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

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.

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

Boman, M. 1997. Norms as Constraints on Real-Time Autonomous Agent Action. In *Multi-Agent Rationality*, Boman and Van de Welde (eds) Springer Verlag.

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.

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

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

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

Jameson, A. and Weis, T. 1995. How to Juggle Discourse Obligations. *Proceedings of the Symposium on Conceptual and Semantic Knowledge in Language Generation*, Heidelberg, November 1995.

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

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

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.

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

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

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

Traum, D.R. and Allen, J.F. 1994. Discourse Obligations in Dialogue Processing. *Proceedings of the 32th Annual Meeting of the ACL*, Las Cruces, NM, 1-8.

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.