Coordinating Agents by Role Based Social Constraints and Conversation Plans

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

Abstract

We explore the view that coordinated behavior is explained by the social constraints 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. 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.

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 set as "goals" 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. Details follow.

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 coordinated 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 (`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 (for example `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-inquiry
 :obliged coordinator
 :authority customer
 :condition (received-inquiry
             :from (agent-playing customer)
             :by (agent-playing coordinator))
 :goal Reply-to-sender
 :enforced (max-reply-time 5))
```

The above requires the `coordinator` agent (the `:obliged` party) to reply to an inquiry from the `customer` (the `:authority` party) in at most five units of time (a constraint on the goal `Reply-to-sender`). This generic obligation becomes active when its condition is satisfied, in this case when the `coordinator` receives an inquiry from the `customer`. When this happens, an actual obligation is created linking the `coordinator` to the `customer` and applying to the actual inquiry received (if many inquiries 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 OIP-s) of a role are inherited by sub-roles.

Goal. Like OIP-s, goals are described in both a generic and an actual form. In the generic form, a goal specification comprises a list of super-goals, lists of goals incompatible with it, any disjunctive coverage (when a number of subgoals are all the possible sub-

goals of a goal), the constraints that can be meaningfully applied to a goal, the optimizations that can be meaningfully requested and, finally, the theories that can be applied to reason about the constraints. For example:

```
(def-goal Reply-to-sender
   :constraints
     (max-reply-time req-max-reply-time)
   :optimizations (time)
   :theories (T1))
```

This specifies that goal `Reply-to-sender` 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 time. Finally, to reason about whether constraints are satisfied or not, theory `T1` can be used.

Theory. A theory is essentially a set of Horn clauses that can be applied to determine if constraints are satisfied. For example, suppose that the message from the `customer` requires a reply in at most 7 units of time, which appears in the `customer`'s message as `(req-max-reply-time 7)` and `T1` has a clause `(satisfied-max-reply-time ?t1 ?t2)` $\Leftarrow$ `(req-max-reply-time ?t1) & (max-reply-time ?t2) & (<= ?t2 ?t1)`. By merging the guarantee provided by the `Reply-to-inquiry` obligation, `(max-reply-time 5)`, with the request `(req-max-reply-time 7)` and with the above clause we can prove `(satisfied-max-reply-time ?t1 ?t2)`, that is the request is within the agent's obligation. This warrants the creation of an actual obligation for the goal `Reply-to-sender` constrained by `(req-max-reply-time 7)`.

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 `Reply-to-sender` 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.

Reasoning About What To Do (or Not)

Suppose now the `coordinator` receives a message from the `customer` in which the latter expresses its goal of inquiring about the possibility of 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 ((inquiry :product widget
                        :amount 1000
                        :due-date 17-sept-97)
               :satisfy (req-max-reply-time 7)
               :optimize (accuracy)))
```

This matches the `Reply-to-inquiry` obligation of `Logistics` and hence `Logistics` builds an actual obligation to `Customer` for replying to the inquiry. From this an actual goal to `Reply-to-sender` in less than 7 units of time and with (preferred) maximal accuracy is generated. When proving that an obligation applies, the framework allows the use of constraint theories to also infer the sanction associated to the obligation. Thus, if an obligation is in contradiction with an interdiction, the agent can compare the two sanctions and decide for the smallest one. To satisfy the obligation, the agent next looks for a plan that can achieve the goal with its attached constraints and (preferably) optimizations. In our case, the plan must produce the estimate quickly and (preferably) with high accuracy. To find the appropriate plan, we use the theories again to prove that the constraints guaranteed by a plan satisfy those requested by the goal. One such plan for example may have `Logistics` use scheduling software to plan the production of the order within the requested time frame. The execution of this software must be necessarily quick and as accurate as possible, so `Logistics` will set the parameters and the order of actions in the plan to reflect these (see later how).

In this process there are several stages where various forms of reasoning about obligations etc. are carried out. We summarize them as follows (where *Beliefs* is the agent's beliefs, *Theory* are relevant constraint theories, *Obligation*, *Interdiction* and *Permission* are constraints guaranteed by some obligation, interdiction, permission and, finally, *Request* is the requested constrained goal):

1. $Beliefs \cup Theory \cup Obligation \supset Request$. The agent is obliged to satisfy request.
2. $Beliefs \cup Theory \cup Interdiction \supset Request$. The agent is forbidden to do request.
3. $Beliefs \cup Theory \cup Permission \supset Request$. The agent has latitude to do or not request.
4. $Beliefs \cup Theory \cup Plan \supset Request$. The plan is usable for request.
5. $Theory \cup Plan \supset Obligation$. Plan is always usable for obligation.
6. $Theory \cup Plan \supset Interdiction$. Plan usage is always forbidden.

Finally, the operational architecture keeps track of how obligations are derived from events like received messages, how goals are derived from obligations and

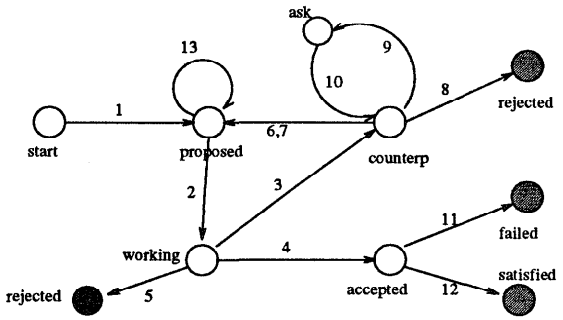


Figure 1: Graph representation of Customer-conversation.

```
(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))
```

Figure 2: Conversation rule.

how plans have been selected for goals, using a truth maintenance system. This allows agents to retract obligations, goals, etc. when agents change decisions, like retracting an order or dropping an interdiction because of a more important incompatible obligation.

Plan Specification and Execution

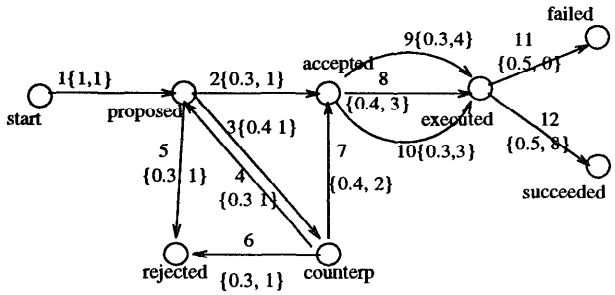
Conversation plans (Barbuceanu & Fox 96) are descriptions of how an agent *acts* and *interacts* in certain situations. A conversation plan consists of states (with distinguished initial and final states) and rule governed transitions together with a control mechanism and a local data-base that maintains 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 (but the approach is essentially independent from KQML).

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, a hierarchical organization of conversations allowing parent conversations to control the execution of child conversations, a more complex typology of rules including various forms of event/condition triggered rules, and, finally, a mechanism allowing a conversation to be suspended (with preserved state), other conversation to proceed and the suspended conversation to be resumed when certain conditions related to the agent environment and conversations are satisfied. All these provide flexible control handles allowing the use of conversations as generalized processes that capture both interaction and local processing.

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 in-



Ordering produced by value iteration: proposed: 2,3,5 accepted: 9, 8, 10
counterp: 7,4,6 executed: 12, 11

Figure 3: Using value iteration to reorder rules.

definite 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 π 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 π at any state s can be computed by solving this system of linear equations. A policy π is optimal if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in S$ and all policies π'. 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.

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.

Back to the Supply Chain

One typical round of interactions starts with the **Customer** sending an inquiry about some order to **Logistics**. To answer it, **Logistics** sets up an appropriate 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.

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. As objective forces determining behavior in organizations, social constraints are *necessary* components of any account of organizational behavior. This is not the case for *mental states* that are externally ascribed and produce animistic explanations which are not logically necessary.

Social constraints have been largely ignored previously, with some exceptions like (Shoham and Tennenholtz 95) who study their general utility, (Castelfranchi 95) who stresses the importance of obligations in organizations but does not advance operational architectures and AOP (Shoham 93) where obligations are defined but not really exploited.

Up to now, our focus has been on prototyping our ideas into systems that can be quickly evaluated in applications. Existing evaluations include several problems, ranging from puzzles like n-queens to supply chain coordination projects carried out in cooperation with industry. 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 hours or 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.

As for future work, it is clear that a system like this also needs clear semantics. Very briefly, we are using the reduction of deontic logic to dynamic logic due to (Mayer 88) in a multi-agent framework. We define obligation, interdiction and permission as follows, where V_α^{ij} denotes a violation by i of a constraint imposed by j wrt α (associated with a cost to be paid):

- (For i j α)=$[\alpha]^i V_\alpha^{ij}$: i is forbidden by j to execute α.
- (Per i j α)=$\neg$(For i j α): i is permitted by j to execute α.
- (Obl i j α)=(For i j -α): i is obliged by j to execute α.

With this, we can 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)=
1. (Obl 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, namely the agent in authority can relieve the obliged agent from the obligation.

Other work planned for the near future include workflow modeling and enactment (Medina-Mora et al 92) as well as using the wide range of representations provided to build multi-level explanations of agent behavior in an industrial setting.

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, 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, R. E. 1957. *Dynamic Programming.* Princeton University Press, Princeton.

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

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

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

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

Mayer, 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, R., Winograd, T., Flores, R. and Flores, F. 1992. The Action Workflow Approach to Workflow Management Technology. In CSCW 92 Proceedings, 281-288.

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.