Commitment Protocol Generation

Akın Günay[1,*], Michael Winikoff[2], and Pınar Yolum[1]

[1] Computer Engineering Department, Bogazici University, Istanbul, Turkey
{akin.gunay,pinar.yolum}@boun.edu.tr
[2] Department of Information Science, University of Otago, Dunedin, New Zealand
michael.winikoff@otago.ac.nz

Abstract. Multiagent systems contain agents that interact with each other to carry out their activities. The agents' interactions are usually regulated with protocols that are assumed to be defined by designers at design time. However, in many settings, such protocols may not exist or the available protocols may not fit the needs of the agents. In such cases, agents need to generate a protocol on the fly. Accordingly, this paper proposes a method that can be used by an agent to generate commitment protocols to interact with other agents. The generation algorithm considers the agent's own goals and capabilities as well as its beliefs about other agents' goals and capabilities. This enables generation of commitments that are more likely to be accepted by other agents. We demonstrate the workings of the algorithm on a case study.

1 Introduction

Interaction is a key element of many multiagent systems. Agents need to interact for various reasons such as coordinating their activities, collaborating on tasks, and so on. These interactions are generally regulated by interaction protocols that define the messages that can be exchanged among agents. Traditionally, agents are supplied with interaction protocols at design time. Hence, they do not need to worry about which protocol to use at run time and can just use the given protocol as they see fit.

However, in open agent systems, where agents enter and leave, an agent may need to interact with another agent for which no previous interaction protocol has been designed. For example, a buyer may know of interaction protocols to talk to a seller, but may not be aware of an interaction protocol to talk to a deliverer. If these two agents meet, they need to figure out a protocol to complete their dealing. Additionally, even if there is an existing interaction protocol, the interaction protocols that are designed generically may make false assumptions about agents' capabilities, which would make the interaction protocol unusable in a real setting. For example, assume that an e-commerce protocol specifies that a buyer can pay by credit card upon receiving goods from a seller. If the buyer does not have the capability to pay by credit card, this protocol will not achieve its purpose. Even when the capabilities of the agents are aligned with those expected by the interaction protocol, the current context of the agents may not be appropriate to engage in the protocol. Following the previous example, an agent who

* Akın Günay is partially supported by TÜBİTAK Scholarships 2211 and 2214 and Pınar Yolum is partially supported by a TÜBİTAK Scholarship 2219.

can pay by credit card might have a current goal of minimizing bank transactions for that month and thus may find it more preferable to pay cash. That is, based on its current goals and existing commitments, the interactions that it is willing to engage in may differ. Therefore an interaction protocol that is blind to agents' current needs would not be applicable in many settings.

Accordingly, we argue that an agent needs to generate appropriate interaction protocols itself at run time. Since the agent would know its own capabilities, goals, and commitments precisely, it can generate an interaction protocol that respects these. However, for the interaction protocol to be successful, it should also take into account the participating agents' context.

Many times, even though the goals, commitments, or the capabilities of the other agents may not be known in full, partial information will exist. For example, agents may advertise their capabilities especially if they are offering them as services (e.g., selling goods). Existing commitments of the other agents may be known if the agent itself is part of those commitments (e.g., the agent has committed to deliver, after payment). The partial goal set of the participating agents may be known from previous interactions (e.g., the agent is interested in maximizing cash payments), or from domain knowledge (e.g. merchants in general have the goal of selling goods and/or services). Hence, the other agents' context can be approximated and using this approximate model a set of possible interaction protocols can be generated.

To realize this, we propose a framework in which agents are represented with their capabilities, goals, and commitments. The interactions of the agents are represented using commitments [3,14] and the interaction protocols are modeled as commitment protocols. Commitments offer agents flexibility in carrying out their interactions and enable them to reason about them [9,19,21]. An agent that wants to engage in an interaction considers its own goals, makes assumptions about the other agents' goals, and proposes a set of commitments such that, if accepted by the other agent, will lead the initial agent to realize its goal. While doing this generation, the agent also considers its own capabilities, so that it generates commitments that it can realize. Note that even with a good approximation of the other agent, the proposed protocol may not be acceptable. For this reason, the agent generates a set of alternative protocols rather than a single one. The exact protocol that will be used is chosen after deliberations with other agents. Having alternative protocols is also useful for recoverability. That is, if a protocol is chosen by the agents, but if one of the agents then violates a commitment, the goals will not be realized as expected. In this case, agents can switch to an alternative protocol. This work is novel in that it situates commitment-based protocols in the larger context of agents by relating commitments to the agents goals, capabilities, and their knowledge of other agents' goals and capabilities.

The rest of this paper is organized as follows. Section 2 describes our technical framework in depth. Section 3 introduces our algorithm for generating commitment protocols based on agents' goals and capabilities. Section 4 applies the algorithm to a case study. Section 5 explains how our approach can be used in a multiagent system. Finally, Section 6 discusses our work in relation to recent work.

2 Technical Framework

In this section we define formally the necessary concepts: agents which have goals that they want to fulfill, and certain capabilities (formalized as propositions that they are able to bring about). We also define the notion of a social commitment between agents (in line with existing approaches, e.g. [21]). The concepts are captured using the following syntax, where *prop* is a proposition, and *agent* is an agent identifier.

$$\begin{array}{ll} \textit{commitment} & \rightarrow \textit{C(agent, agent, prop, prop)}^{\textit{cstate}} \\ \textit{goal} & \rightarrow \textit{G}_{\textit{agent}}\textit{(prop, prop, prop)}^{\textit{gstate}} \\ \textit{service} & \rightarrow \textit{S}_{\textit{agent}}\textit{(prop, prop)} \\ \textit{belief} & \rightarrow \textit{BG}_{\textit{agent}}\textit{(agent, prop, prop)} \mid \textit{BS}_{\textit{agent}}\textit{(agent, prop, prop)} \\ \textit{cstate} & \rightarrow \textit{Null} \mid \textit{Requested} \mid \textit{Active} \mid \textit{Conditional} \mid \textit{Violated} \mid \textit{Fulfilled} \mid \textit{Terminated} \\ \textit{gstate} & \rightarrow \textit{Inactive} \mid \textit{Active} \mid \textit{Satisfied} \mid \textit{Failed} \end{array}$$

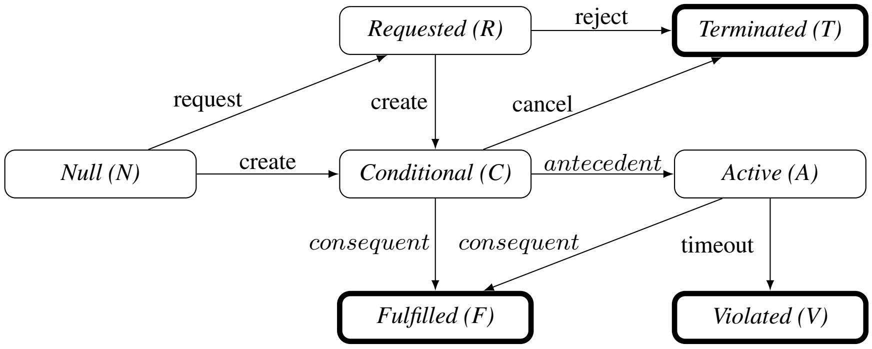

Fig. 1. Life cycle of a commitment

Commitments. A *commitment* $C(debtor, creditor, antecedent, consequent)^{state}$ expresses the social contract between the agents $debtor$ and $creditor$, such that if the $antecedent$ holds, then the $debtor$ is committed to the $creditor$ to bring about the $consequent$. Each commitment has a $state$ that represents the current state of the commitment in its life cycle. The state of a commitment evolves depending on the state of the antecedent and the consequent and also according to the operations performed by the debtor and the creditor of the commitment. We show the life cycle of a commitment in Fig. 1. In this figure, the rectangles represent the states of the commitment and the directed edges represent the transitions between the states. Each transition is labeled with the name of the triggering event. A commitment is in *Null* state before it is created. The create operation is performed by the $debtor$ to create the commitment and the state of the commitment is set to *Conditional*. If the $antecedent$ already holds while creating the commitment, the state of the commitment becomes *Active* immediately. It is also possible for the $creditor$ of a commitment in *Null* state to make a request to the $debtor$ to create the commitment. In this case, the state of the commitment is *Requested*. The $debtor$ is free to create the requested commitment or reject it, which makes the commitment *Terminated*. A *Conditional* commitment becomes *Active* if the $antecedent$ starts to hold, *Fulfilled* if the $consequent$ starts to hold or *Terminated* if the $debtor$ cancels the commitment. An *Active* commitment becomes *Fulfilled* if the $consequent$ starts to hold, *Violated* if the $debtor$ cancels the commitment or *Terminated* if the $creditor$ releases the $debtor$ from its commitment. *Fulfilled*, *Violated* and *Terminated* states are terminal states (depicted with thicker borders in Fig. 1)

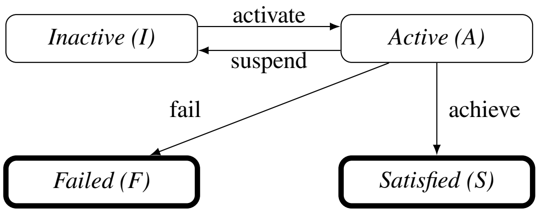

Fig. 2. Life cycle of a goal

Goals. A *goal* $G_{agent}(precondition, satisfaction, failure)^{state}$ represents an aim of an agent such that the $agent$ has a goal to achieve $satisfaction$ if $precondition$ holds and the goal fails if $failure$ occurs (adapted from [20]). The state of the goal is represented by $state$. We show the life cycle of a goal in Fig. 2. A goal is in *Inactive* state if its $precondition$ does not hold. An inactive goal is not pursued by the agent. A goal is in *Active* state if its $precondition$ holds and neither $satisfaction$ nor $failure$ holds. An active goal is pursued by the agent. A goal is *Satisfied*, if $satisfaction$ starts to hold while in the *Active* state. A goal is *Failed*, if $failure$ occurs while in the *Active* state. An active goal may also be suspended, if the precondition ceases to hold. The *Satisfied* and *Failed* states are terminal states.

Capabilities. A *capability* $S_{agent}(precondition, proposition)$ states that an agent has the capability of performing an action (or actions) that will make $proposition$ true. However, this is only possible if the $precondition$ holds. Note that we use the terms "**capability**" and "**service**" interchangeably: in a context where an agent does something for itself "capability" makes more sense, but when an agent acts for another agent, then "service" is more appropriate.

Beliefs. Agents have their own *beliefs* about other agents' goals and capabilities. $BG_{agent_i}(agent_j, condition, satisfaction)$ represents that $agent_i$ believes $agent_j$ has the goal $satisfaction$ if $condition$ holds. Note that beliefs about other agents' goals do not include information about the failure conditions. Similarly $BS_{agent_i}(agent_j, condition, proposition)$ represents that $agent_i$ believes $agent_j$ is able to bring about the $proposition$, if the $condition$ holds. Beliefs about other agents' capabilities essentially correspond to services provided by other agents and interpreted as $agent_i$ believes that $agent_j$ provides a service to bring about $proposition$, if $condition$ is brought about (most probably by an effort of $agent_i$). As discussed in Section 1, although in general other agents' goals and capabilities are private, some information will be available. Although it is possible that advertised services may differ from the actual capabilities of the agent. For example, certain capabilities may not be advertised, or some advertised services may in fact be realized by a third party (e.g. a merchant delegating delivery to a courier).

Agents and **Multiagent system**. An *agent* is a four tuple $A = \langle \mathcal{G}, \mathcal{S}, \mathcal{C}, \mathcal{B} \rangle$, where $\mathcal{G}$ is a set of goals that agent A has, $\mathcal{S}$ is a set of services (aka capabilities) that agent A can provide, $\mathcal{C}$ is a set of commitments that agent A is involved in and $\mathcal{B}$ is a set of beliefs that agent A has about other agents. A *multiagent system* $\mathcal{A}$ is a set of agents $\{A_1, \ldots, A_n\}$. We write $a.X$ to denote the X component of the agent, e.g. writing $a.\mathcal{G}$ to denote the agent's goals, $a.\mathcal{C}$ to denote its commitments etc.

Protocol. We adopt the definition of commitment protocols [7,21] in which a *protocol* P is a set of (conditional) commitments. Hence, we do not have explicit message orderings. Each agent can manipulate the commitments as it sees fit. The manipulations of the commitments lead to state changes in the lifecycles of the commitments as depicted in Fig. 1. Unlike traditional approaches to capturing protocols, such as AUML, this approach, using social commitments, aims to provide minimal constraints on the process by which the interaction achieves its aims [15]. We emphasise that a set of commitments is a protocol in the sense that it allows for a range of possible concrete interactions, unlike the notion of contract used by Alberti *et al.* [1] which represents a single specific concrete interaction.

Definition 1 (Proposition Support). *Given a set Γ of propositions that hold, and a proposition p, the agent $a = \langle \mathcal{G}, \mathcal{S}, \mathcal{C}, \mathcal{B} \rangle$ supports p, denoted as $a \Vdash p$, iff at least one of the following cases holds:*

- *base case: $\Gamma \models p$, i.e. p already holds*
- *capability: $\exists S_a(pre, prop) \in \mathcal{S} : \{prop \rightarrow p \wedge a \Vdash pre\}$, i.e. the agent is able to bring about p (more precisely, a condition prop which implies p) itself, and the required condition is also supported*
- *commitment: $\exists C(a', a, \top, cond)^A \in \mathcal{C} : \{cond \rightarrow p\}$, i.e. there is an active commitment from another agent to bring about p*
- *conditional: $\exists C(a', a, ant, cond)^C \in \mathcal{C} : \{cond \rightarrow p \wedge a \Vdash ant\}$, i.e. there is a conditional commitment from another agent to bring about p, and the antecedent of the commitment is supported by agent a*

The *capability* case states that p can be made true by agent a if p is one of the agent's capabilities. This is the strongest support for p, since p can be achieved by the agent's own capabilities. The *commitment* case states that the agent has a commitment in which it expects p to become true (because it is the creditor of an active commitment). Note that this is weaker than the *capability* condition since the commitment may be violated by its debtor. In the *conditional* case, the agent first needs to realize the antecedent for p to be achieved.

Definition 2 (Goal Support). *A goal $g = G_a(pre, sat, fail)^A$ is supported by the agent $a = \langle \mathcal{G}, \mathcal{S}, \mathcal{C}, \mathcal{B} \rangle$, denoted as $a \Vdash g$, if $a \Vdash sat$.*

Theorem 1. *If a proposition p (respectively goal g) is supported by agent a, then the agent is able to act in such a way that p (resp. g) eventually becomes true (assuming all active commitments are eventually fulfilled).*

Proof: *Induction over the cases in Definition 2 (details omitted).*

3 Commitment Protocol Generation Algorithm

We present an algorithm that uses the agent's capabilities, commitments and also beliefs about other agents, to generate a set of alternative commitment protocols[1] such that each generated protocol supports the given agent's set of goals. That is, for each given goal of the agent, either the agent is able to achieve the goal by using its own capabilities, or the agent is able to ensure that the goal is achieved by relying appropriately on a commitment from another agent which has the goal's satisfaction condition as its consequent. More precisely, if an agent a cannot achieve a desired proposition p using its own capabilities, then the algorithm generates a proposed commitment such as $C(a', a, q, p)^R$ (ensuring q is supported by a) to obtain (conditional) proposition support for p, which implies goal support for goal $g \equiv G_a(pre, p, fail)$.

Note that in general, we can only expect to be able to obtain *conditional* support (in terms of Definition 1). Obtaining *capability* support amounts to extending the agent's capabilities, and obtaining *commitment* support amounts to getting an active commitment $C(a', a, \top, q)^A$ which, in general, another agent a' would not have any reason to accept. Thus, the algorithm proposes commitments that are likely to be attractive to a' by considering its beliefs about the goals of a' and creating a candidate commitment $C(a', a, q, p)^R$ where q is a proposition that is believed to be desired by a' (i.e. satisfies one of its goals). Clearly, there are situations where a given goal cannot be supported (e.g. if no other agents have the ability to bring it about, or if no suitable q can be found to make the proposed commitments attractive), and hence the algorithm may not always generate a protocol.

We divide our algorithm into four separate functions (described below) for clarity:

- *generateProtocols* takes an agent and the set of proposition that hold in the world as arguments, and returns a set of possible protocols $\mathcal{P} = \{P_1, \ldots, P_n\}$, where each protocol is a set of proposed commitments (i.e. it returns a set of sets of commitments).
- *findSupport* takes as arguments an agent, a queue of goals, a set of propositions that are known to hold, and a set of commitments that are known to exist (initially empty); and does the actual work of computing the possible protocols, returning a set of possible protocols $\mathcal{P}$.
- *isSupported* takes as arguments an agent, a proposition, a set of propositions known to hold, and a set of commitments known to exist; and determines whether the proposition is supported, returning a Boolean value.
- *updateGoals* is an auxiliary function used by the main algorithm, and is explained below.

The *generateProtocols* function (see Algorithm 1) is the entry point of the algorithm. It has as parameters an agent a and a set of propositions Γ that hold in the world. Γ is meant to capture a's current world state. The algorithm finds possible, alternative protocols such that when executed separately, each protocol ensures that all of the goals of that agent are achievable.

[1] In practice, we may want to generate the set incrementally, stopping when a suitable protocol is found.

Algorithm 1. $\mathcal{P}$ generateProtocols(a, Γ)

```
Require: $a$, the agent that the algorithm runs for
Require: $\Gamma$, set of propositions known to be true
1: queue $\mathcal{G}' \leftarrow \{g|g \in a.\mathcal{G} \wedge g.state = Active\}$
2: return findSupport($a, \mathcal{G}', \Gamma, \emptyset$)
```

The *generateProtocols* function copies the agent's active goals into a queue structure $\mathcal{G}'$ for further processing and then calls the recursive function *findSupport* providing a (the agent), $\mathcal{G}'$ (its currently active goals), Γ (the propositions that currently hold), and $\emptyset$ (initial value for Δ) as arguments. The *generateProtocols* function returns the result of *findSupport*, which is a set of commitment protocols ($\mathcal{P}$), i.e. a set of sets of commitments. Recall that we use $a.\mathcal{G}$ to denote the goals $\mathcal{G}$ of agent a, and that for goal g we use $g.state$ to denote its state.

The main function is *findSupport* (see Algorithm 2). The function recursively calls itself to generate alternative commitment protocols which support every given goal of the agent a. The function takes as arguments an agent a, the queue of the agent's goals $\mathcal{G}'$ that need to be supported, a set Γ of propositions that are known to be true, and a set Δ of commitments that are known to exist. The function first defines sets $\mathcal{BG}$ and $\mathcal{BS}$ of (respectively) the beliefs of agent a about the goals and the services of other agents. It then pops the next goal g from the goal queue $\mathcal{G}'$ (Line 3). If all goals are considered (i.e. $g =$ Null), then there is no need to generate extra commitments. Hence, the algorithm simply returns one protocol: the set of the commitments already proposed. This corresponds to the base case of the recursion (Lines 4–5). If the agent already supports g (determined by *isSupported* function, see Algorithm 3), then the algorithm ignores g and calls itself for the next goal in $\mathcal{G}'$ (Line 8).

Otherwise, the function searches for one or more possible sets of commitments that will support the goal g. It first initializes the set of alternative protocols $\mathcal{P}$ to the empty set (Line 10). Then the algorithm searches for candidate commitments that will support g. As a first step it checks whether it has any capabilities that would support this goal if the precondition of the capability could be achieved through help from other agents (Line 11). Note that if the preconditions could be achieved by the agent itself then the algorithm would have detected this earlier in Line 3. Hence, here the specific case being handled is that the precondition of a capability cannot be achieved by the agent itself, but if it were achieved through other agents, then the capability would enable the agent to reach its goal g. For each such capability, we make the precondition pre a new goal for the agent, add it to the list of goals $\mathcal{G}'$ that it wants to achieve, and recursively call *findSupport* to find protocols.

After checking its own capabilities for achieving g, the agent then also starts looking for another agent with a known service $s' \in \mathcal{BS}$ such that s' achieves the satisfaction condition of the goal g (Line 14). For any such service s', we generate a proposed commitment of the form $C(a', a, sat', prop)^R$ (Line 16), where a' is the agent that is believed to be provide the service s', a is the agent being considered by the call to the function (its first argument), $prop$ implies the satisfaction condition of the desired goal g (i.e. $prop \rightarrow sat$), and sat' is an "attractive condition" to the proposed debtor agent (a'). The notion of "attractive to agent a'" is defined in line 15: we look for a condition

Algorithm 2. $\mathcal{P}$ findSupport$(a, \mathcal{G}', \Gamma, \Delta)$

```
Require: $a$, the agent that the algorithm runs for
Require: $\mathcal{G}'$, queue of agent's (active) goals
Require: $\Gamma$, set of propositions known to be true
Require: $\Delta$, set of commitments already generated (initially called with $\emptyset$)
 1: define $\mathcal{BG} \equiv \{b | b \in a.\mathcal{B} \wedge b = BG_a(a', gc, s)\}$
 2: define $\mathcal{BS} \equiv \{b | b \in a.\mathcal{B} \wedge b = BS_a(a', c, p)\}$
 3: $g \leftarrow$ pop$(\mathcal{G}')$
 4: if $g =$ Null then
 5:     return $\{\Delta\}$
 6:     // else $g = G_a(gpre, sat, fail)^A$
 7: else if isSupported$(a, sat, \Gamma, \Delta)$ then
 8:     return findSupport$(a, \mathcal{G}', \Gamma, \Delta)$
 9: else
10:     $\mathcal{P} = \emptyset$
11:     for all $\{s \mid S_a(pre, prop) \in a.\mathcal{S} \wedge prop \rightarrow sat\}$ do
12:         $\mathcal{P} \leftarrow \mathcal{P} \cup$ findSupport$(a, \{G_a(\top, pre, \bot)^A\} \cup \mathcal{G}', \Gamma, \Delta)$
13:     end for
14:     for all $\{s' \mid BS_a(a', cond, prop) \in \mathcal{BS} \wedge prop \rightarrow sat\}$ do
15:         for all $\{g' \mid BG_a(a', pre', sat') \in \mathcal{BG} \wedge$ isSupported$(a, pre', \Gamma, \Delta)\}$ do
16:             $c \leftarrow C(a', a, sat', prop)^R$
17:             $\mathcal{G}'' \leftarrow$ updateGoals$(sat', prop, a.\mathcal{G}, \mathcal{G}')$
18:             if $\neg$isSupported$(a, sat', \Gamma, \Delta)$ then
19:                 $\mathcal{G}'' \leftarrow \{G_a(\top, sat', \bot)^A\} \cup \mathcal{G}''$
20:             end if
21:             if $\neg$ isSupported$(a, cond, \Gamma, \Delta)$ then
22:                 $\mathcal{G}'' \leftarrow \{G_a(\top, cond, \bot)^A\} \cup \mathcal{G}''$
23:             end if
24:             $\mathcal{P} \leftarrow \mathcal{P} \cup$ findSupport$(a, \mathcal{G}'', \Gamma, \Delta \cup \{c\})$
25:         end for
26:     end for
27:     return $\mathcal{P}$
28: end if
```

sat' that is believed to be a goal of agent a'. Specifically, we consider the known goals $\mathcal{BG}$ of other agents, and look for a $g' \in \mathcal{BG}$ such that $g' = BG_a(a', pre', sat')$ where pre' is already supported by agent a.

Next, having generated a potential commitment $C(a', a, sat', prop)^R$ where the debtor, a', has a service that can achieve the desired condition $prop$ and has a goal to bring about sat' (which makes the proposed commitment attractive), we update the goals of the agent (discussed below) and check whether (1) the promised condition sat' is supported by agent a, and (2) the precondition $cond$ for realizing $prop$ is supported by agent a. If they are supported, then a does not need to do anything else. Otherwise, it adds the respective proposition to the list of goals $\mathcal{G}''$ (Lines 19 and 22), so that appropriate support for these propositions can be obtained.

Finally, the agent calls the function recursively to deal with the remainder of the goals in the updated goal queue $\mathcal{G}''$. When doing this, it adds the currently created commitment c to the list of already generated commitments Δ. The result of the function call is added to the existing set of possible protocols $\mathcal{P}$ (line 24). Once the agent has completed searching for ways of supporting g, it returns the collected set of protocols $\mathcal{P}$. Note that if the agent is unable to find a way of supporting its goals, then $\mathcal{P}$ will be empty, and the algorithm returns the empty set, indicating that no candidate protocols could be found.

Algorithm 3. $\{$**true** | **false**$\}$ isSupported(a, p, Γ, Δ)

```
Require: $a$, agent to check for support of $p$
Require: $p$, property to check for support
Require: $\Gamma$, set of propositions known to be true
Require: $\Delta$, set of commitments already generated
 1: if $\Gamma \models p$ then
 2:     return true
 3: end if
 4: for all $s = S_a(pre, prop) \in a.\mathcal{S}$ do
 5:     if $prop \rightarrow p \wedge$ isSupported$(a, pre, \Gamma, \Delta)$ then
 6:         return true
 7:     end if
 8: end for
 9: for all $\{c \mid C(a', a, cond, prop) \in (a.\mathcal{C} \cup \Delta)\}$ do
10:     if $c.state = Active \wedge prop \rightarrow p$ then
11:         return true
12:     else if $(c.state = Conditional \vee c.state = Requested) \wedge prop \rightarrow p \wedge$
        isSupported$(a, cond, \Gamma, \Delta)$ then
13:         return true
14:     end if
15: end for
16: return false
```

Algorithm 3 defines the *isSupported* function. This algorithm corresponds to Definition 1 and returns true if the given proposition p is supported by the given agent a, and false otherwise. The first case (line 1) checks whether the proposition is known to be true. The second case checks capability support. That is, whether p is supported by a capability s of the agent. More precisely, if the proposition $prop$ of s implies p and the precondition pre of s is supported by the agent (Lines 4-8). The third case checks commitment support by checking whether a has (or will have) an active commitment c, in which a is the creditor and the consequent $prop$ implies p (Lines 10-11). In the last case, the algorithm checks conditional support by checking whether a has (or will have) a conditional commitment c, in which a is the creditor, the consequent $prop$ implies p and a supports the antecedent $cond$ (Lines 12-14). If none of the above cases hold, then the algorithm returns false, indicating that p is not supported by a.

Algorithm 4 defines the *updateGoals* function. This function is called when a new commitment is generated to support goal g of agent a. It takes propositions ant and $cons$ corresponding respectively to the antecedent and consequent of the new commitment. The function also takes as arguments the goals $\mathcal{G}$ of agent a, and the queue of

Algorithm 4. $\mathcal{G}''$ updateGoals(ant, $cons$, $\mathcal{G}$, $\mathcal{G}'$)

```
Require: $ant$, the antecedent of the new commitment
Require: $cons$, the consequent of the new commitment
Require: $\mathcal{G}$, set of agent's goals
Require: $\mathcal{G}'$, the current queue of (potentially) unsupported goals
1: create new queue $\mathcal{G}''$
2: $\mathcal{G}'' \leftarrow$ copy of $\mathcal{G}'$
3: for all $\{g \mid G_a(pre, sat, fail) \in \mathcal{G}\}$ do
4:    if $g.state = Inactive \wedge (ant \rightarrow pre \vee cons \rightarrow pre)$ then
5:       $g.state \leftarrow Active$
6:       push($\mathcal{G}'', g$)
7:    end if
8: end for
9: return $\mathcal{G}''$
```

currently unsupported goals $\mathcal{G}'$. The algorithm assumes that both ant and $cond$ will be achieved at some future point due to the generated commitment. Accordingly, the algorithm assumes that currently inactive goals which have ant or $cond$ as their precondition will be activated at some future point. Hence, these goals also need to be able to be achieved, i.e. to be supported by agent a. The algorithm thus generates these additional goals, and adds them to a (new queue) $\mathcal{G}''$. The algorithm first creates a new queue $\mathcal{G}''$ and copies into it the current contents of $\mathcal{G}'$ (Line 2). Then the goals in $\mathcal{G}$ that are inactive but will be activated are pushed into $\mathcal{G}''$ as active goals (Lines 3-8). Finally, $\mathcal{G}''$ is returned. Instead of pushing the goals that are assumed to be activated directly into $\mathcal{G}'$, the algorithm creates a new queue. This is done because every recursive call in line 24 of Algorithm 2 is related to a different commitment, which activates different goals depending on its antecedent and consequent. Hence each recursive call requires a different goal queue.

The algorithms presented are sound in the sense of Theorem 1: for any generated protocol, the agent is able to act in such a way as to ensure that the desired goal becomes achieved, without making any assumptions about the behaviour of other agents, other than that they fulfill their active commitments. The algorithms in this section have been implemented (available from `http://mas.cmpe.boun.edu.tr/akin/cpgen.html`), and have been used to generate protocols for a number of case studies, including the one we present next, which took 0.6 seconds to generate protocols (on a 2.7GHz Intel Core i7 machine with 4 GB RAM running Ubuntu Linux).

4 Case Study

We illustrate our commitment generation algorithm's progress through an e-commerce scenario. In this scenario there is a customer (Cus), a merchant (Mer) and a bank ($Bank$). The goal of the customer is to buy some product from the merchant. The customer also has a goal of being refunded by the merchant, if the purchased product is defective. The customer is capable of making payment orders to the bank to pay to the merchant. The customer can also use a gift card, instead of payment. The merchant's goal is to be paid or to receive a gift card and the bank's goal is to get payment orders to earn commissions. We discuss the scenario from the customer's point of view, who runs our algorithm to generate a protocol in order to satisfy her goals. We first describe the propositions that we use and their meanings:

- *Delivered*: The purchased product is delivered to the customer.
- *Paid*: The merchant is paid.
- *HasGiftCard*: The customer has a gift card.
- *GiftCardUsed*: The customer uses the gift card.
- *Defective*: The delivered product is defective.
- *Returned*: The delivered product is returned to the merchant.
- *Refunded*: The customer is refunded.
- *PaymentOrdered*: The bank receives a payment order.

The customer has the following goals and capabilities: g_1 states that the goal of the customer is to have the product be delivered (without any condition) and g_2 represents the goal of the customer to be refunded, if the delivered product is defective, s_1 states that the customer is able to make payment orders (without any condition), and s_2 states that the customer is able to use a gift card (instead of payment), if she has one. Finally, s_3 states that the customer is capable of returning a product, if it is defective.

- $g_1 = G_{Cus}(\top, Delivered, \neg Delivered)$
- $g_2 = G_{Cus}(Defective, Refunded, \neg Refunded)$
- $s_1 = S_{Cus}(\top, PaymentOrdered)$
- $s_2 = S_{Cus}(HaveGiftCard, GiftCardUsed)$
- $s_3 = S_{Cus}(Defective, Returned)$

The customer has the following beliefs about the other agents: b_1 and b_2 state that the customer believes that the merchant provides a service to deliver a product, if the merchant is paid or a gift card is used, respectively. b_3 represents the belief that the merchant will give a refund, if a product is returned, and b_4 is the belief about the service of the bank to perform a money transaction for payment, if the bank receives such a request. The customer also believes that the goal of the merchant is to be paid (b_5) or to receive a gift card (b_6) and refund the customer if a sold product is defective (b_7), in order to ensure customer satisfaction. The goal of the bank is to receive payment orders (b_8), so that it can earn a commission from payment orders.

- $b_1 = BS_{Cus}(Mer, Paid, Delivered)$
- $b_2 = BS_{Cus}(Mer, GiftCardUsed, Delivered)$
- $b_3 = BS_{Cus}(Mer, Returned, Refunded)$
- $b_4 = BS_{Cus}(Bank, PaymentOrdered, Paid)$
- $b_5 = BG_{Cus}(Mer, \top, Paid)$
- $b_6 = BG_{Cus}(Mer, \top, GiftCardUsed)$
- $b_7 = BG_{Cus}(Mer, Defective, Returned)$
- $b_8 = BG_{Cus}(Bank, \top, PaymentOrdered)$

Figure 3 summarises the case study. Ovals are used to denote services, and rectangles denote propositions. Solid arrows (e.g. between the proposition $Paid$ and the service $Delivered$ in the Merchant) indicate the preconditions of a service. Dashed arrows show where a service in one agent is able to bring about a precondition that is desired by another agent.

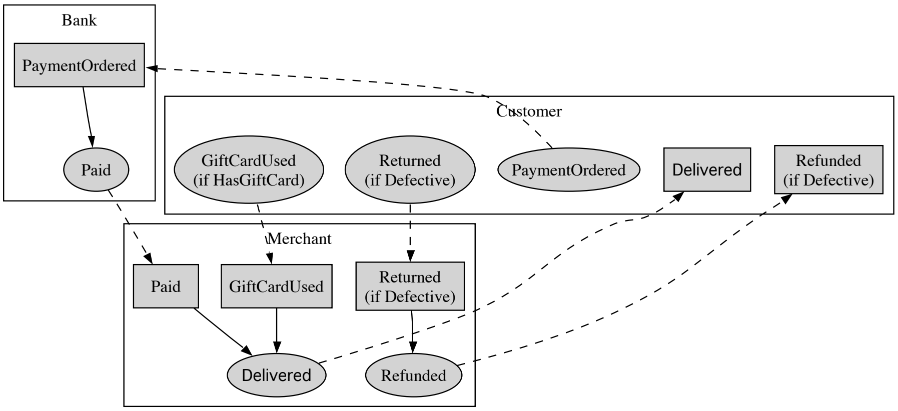

Fig. 3. Case Study

Let us first discuss the states of the merchant's goals g_1 and g_2. The algorithm considers both goals as active. g_1 is active, since its condition is $\top$. On the other hand, $Defective$ actually does not hold initially, which means g_2 should not be active. However, the algorithm assumes that $Defective$ holds, since its truth value is not controlled by any agent and therefore may or may not be true while executing the protocol. Using this assumption, the algorithm aims to create necessary commitments to capture all potential future situations during the execution of the protocol.

Let us walk through the protocol generation process. The algorithm starts with g_1. To support $Delivered$, which is the satisfaction condition of g_1, the algorithm generates the commitment $c_1 = C(Mer, Cus, Paid, Delivered)^R$ using the belief b_1, which is about the service to provide $Delivered$ and b_5, which is the goal of the merchant. However, the antecedent $Paid$ of c_1 is not supported by the customer. Hence, the algorithm considers $Paid$ as a new goal of the customer and starts to search for support for it. It finds the belief b_4, which indicates that the bank can bring about $Paid$ with a condition $PaymentOrdered$, which is also a goal of the bank due to b_8. $PaymentOrdered$ is already supported, since it is a capability of the customer (s_1). Hence, the algorithm generates the commitment $c_2 = C(Bank, Cus, PaymentOrdered, Paid)^R$. At this point, everything is supported to achieve g_1. The algorithm continues for g_2, which is achieved, if $Refunded$ holds. $Refunded$ can be achieved by generating the commitment $c_3 = C(Mer, Cus, Returned, Refunded)^R$ using the service b_3 and the goal b_7 of the merchant. The antecedent $Returned$ is a capability of the customer with a supported condition $Defective$. Hence, everything is supported to achieve g_2 and the algorithm returns the protocol that contains commitments c_1, c_2, and c_3.

Let us examine the protocol. c_1 states that the merchant is committed to deliver the product if the customer pays for it. However, the customer is not capable of payment (cannot bring about $Paid$ by itself). c_2 handles this situation, since the bank is committed to make the payment if the customer orders a payment. Finally, c_3 guarantees a refund, if the customer returns the product to the merchant. Note that the customer returns the product only if it is defective (s_2), hence there is no conflict with the goal (b_5) of the merchant.

Although the above protocol supports all the goals of the customer, the algorithm continues to search for other alternative protocols, since our aim is to generate all possible protocols to achieve the goals. Hence, it starts to search for alternative protocols that support the goals of the customer. It finds that it is possible to support g_1 also by using the service b_2. Accordingly, the algorithm initiates a new alternative protocol and generates the commitment $c_{2-1} = C(Mer, Cus, GiftCardUsed, Delivered)^R$ using the beliefs b_2 and b_6. However, the antecedent $GiftCardUsed$ of c_{2-1} is not supported by the customer, since $HasGiftCard$, which is the condition of service s_2, does not hold. The algorithm searches for support for $HasGiftCard$, but it fails, since neither the customer nor any other agent is able to bring it about.

Note that our algorithm also generates other protocols, which, due to information about other agents not being complete or correct, may be inappropriate. For instance, such a protocol may include a commitment such as $C(Mer, Cus, Paid, Refunded)^R$. This happens because the algorithm considers all believed goals of the other agents while creating commitments. Specifically, to satisfy her goal $Refunded$, the customer considers the known goals of the merchant, and finds three options to offer to the merchant in return: $Paid$, $GiftCardUsed$ and $Returned$. Hence the algorithm creates three alternative commitments using each of these three goals of the merchant and each commitment is considered as an alternative protocol. Another example of this is a situation where the merchant actually replaces a defective product instead of refunding money (i.e. b_2 is incorrect). We deal with inappropriate protocols by requiring all involved agents to agree to a proposed protocol (see below). Specifically in this case when the customer requests the commitment from the merchant, the merchant would not accept the request.

5 Using Generated Protocols

The algorithm presented in the previous section generates candidate protocols, i.e. possible sets of proposed commitments that, if accepted, support the achievement of the desired propositions. In this section we consider the bigger picture and answer the question: *how are the generated candidate protocols used?*

The process is described in Algorithm 5, which uses two variables: the set of candidate protocols ($\mathcal{P}$), and the set of commitments (in the current candidate protocol, P) that agents have already accepted ($\mathcal{C}$). We begin by generating the set of protocols $\mathcal{P}$ (line 1). Next, we need to select one of the protocols[2] (line 2). The selected protocol is removed from $\mathcal{P}$. We then propose each commitment in the protocol to its debtor.

[2] For the present we assume that the selection is done based on the simple heuristic that fewer commitments are preferred.

Algorithm 5. generateAndUseProtocols(a, Γ)

```
Require: $a$, the agent that the algorithm runs for
Require: $\Gamma$, set of propositions known to be true
 1: $\mathcal{P} \leftarrow generateProtocols(a, \Gamma)$
 2: select $P \in \mathcal{P}$
 3: $\mathcal{P} \leftarrow \mathcal{P} \setminus \{P\}$
 4: $\mathcal{C} \leftarrow \emptyset$
 5: for all $C(x, y, p, q)^R \in P$ such that $x \neq a$ do
 6:     Propose $C(x, y, p, q)^R$ to agent $x$
 7:     if Agent $x$ declines then
 8:         for all $C(x, y, p, q)^R \in \mathcal{C}$ do
 9:             Release agent $x$ from the commitment $C(x, y, p, q)^R$
10:         end for
11:         Go to line 2
12:     else
13:         $\mathcal{C} \leftarrow \mathcal{C} \cup \{C(x, y, p, q)^R\}$
14:     end if
15: end for
16: Execute Protocol $P$
```

This is needed because, as noted earlier, domain knowledge about other agents' goals may not be entirely correct or up-to-date. If any agent declines the proposed commitment then we cannot use the protocol, and so we clean up by releasing agents from their commitments in the protocol, and then try an alternative protocol. If all agents accept the commitments, then the protocol is executed.

Note that, since agents may not always fulfill their active commitments, we need to monitor the execution (e.g. [10]), and in case a commitment becomes violated, initiate action to recover. There are a range of possible approaches for recovery including simply abandoning the protocol and generating new protocols in the new state of the world; and using compensation [18].

6 Discussion

We developed an approach that enables agents to create commitment protocols that fit their goals. To achieve this, we proposed to represent agents' capabilities and commitments in addition to their goals. Agents reason about their goals as well as their beliefs about other agents' capabilities and goals to generate commitments. Our experiments on an existing case study showed that an agent can indeed generate a set of commitment protocols that can be used among agents. Hence, even agents who do not have any prior protocols among them can communicate to carry out their interactions.

While we primarily discuss how our approach can be used at runtime, many of the underlying ideas can be used at design time as well. That is, a system designer who is aware of some of the goals and capabilities of the agents that will interact at runtime, can use the algorithm to generate protocols for them. This will enable a principled approach for designing commitment-based protocols.

Goals and commitments have been both widely studied in the literature. On the goals side, Thangarajah *et al.* [17] study relations and conflicts between goals. van Riemsdijk *et al.* [13] study different types of goals and propose to represent them in a unifying framework. On the commitments side, El-Menshawy *et al.* [8] study new semantics for commitments. Chopra and Singh [5,6] study the interoperability and alignment of commitments. However, the interaction between goals and commitments has started to receive attention only recently.

Chopra *et al.* [4] propose a formalization of the semantic relationship between agents' goals and commitment protocols. Their aim is to check whether a given commitment protocol can be used to realize a certain goal. To do this, they define a capability set for each agent and first check if an agent can indeed carry out the commitments it participates in. This is important and can be used by agents to choose among possible commitment protocols. Chopra *et al.* assume that the commitment protocols are already available for agents. By contrast, in our work, we are proposing a method for the agents to generate a commitment protocol that they can use to realize their goals from scratch.

Işıksal [11] studies how an agent can create a single commitment to realize its goal with the help of other agents' in the system. She proposes reasoning rules that can be applied in various situations and she applies these rules on an ambient intelligence setting. She does not generate a set of alternative protocols and does not consider beliefs about other agents' goals as we have done here.

Desai *et al.* [7] propose Amoeba, a methodology to design commitment based protocols for cross-organizational business processes. This methodology enables a system designer to specify business processes through the participating agents' commitments. The methodology accommodates useful properties such as composition. Desai *et al.* model contextual changes as exceptions and deal with them through metacommitments. Their commitment-based specification is developed at design time by a human, based on the roles the agents will play. In this work, on the other hand, we are interested in agents generating their commitments themselves at run time. This will enable agents to interact with others even when an appropriate protocol has not been designed at design time.

Telang *et al.* [16] develop an operational semantics for goals and commitments. They specify rules for the evolution of commitments in light of agents' goals. These practical rules define when an agent should abandon a commitment, when it should negotiate, and so on. These rules are especially useful after a commitment protocol has been created and is in use. In this respect, our work in this paper is a predecessor to the execution of the approach that is described by Telang *et al.*, that is, after the protocol has been generated, the agents can execute it as they see fit, based on their current goals.

The work of Marengo *et al.* [12] is related to this work. Specifically, our notion of support (Definition 1) is analogous to their notion of control: intuitively, in order for an agent to consider a proposition to be supported, it needs to be able to ensure that it is achieved, i.e. be able to control its achievement. However, whereas the aim of their work is to develop a framework for reasoning about control and safety of given protocols, our aim is to derive protocols.

There are a number of directions for future work:

- A key direction is the development of means for ranking generated alternative protocols.
- A second direction is to explore how well our algorithms manage to generate appropriate protocols in situations where the agent's beliefs about other agents' goals and capabilities are incomplete or inconsistent.
- When generating protocols, it may be possible to reduce the search space by interleaving protocol generation with checking the acceptability of the protocol. Rather than waiting until a complete protocol is constructed, whenever a commitment is proposed, we could check with the proposed debtor whether that commitment is acceptable. In general, a commitment's acceptability may depend on the rest of the protocol, but there may be some commitments that are clearly unacceptable regardless of context, and in these cases we can terminate the generation of protocols including that commitment.
- Our representation of protocols follows the "traditional" approach to commitment-based protocols. It has been argued that the representation ought to be extended with the ability to represent regulative temporal constraints [2], and one direction for future work is to extend our protocol generation framework and algorithm to support such constraints.

References

1. Alberti, M., Cattafi, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., Torroni, P.: A Computational Logic Application Framework for Service Discovery and Contracting. International Journal of Web Services Research (IJWSR) 8(3), 1–25 (2011)
2. Baldoni, M., Baroglio, C., Capuzzimati, F., Marengo, E., Patti, V.: A Generalized Commitment Machine for 2CL Protocols and its Implementation. In: Baldoni, M., Dennis, L., Mascardi, V., Vasconcelos, W. (eds.) DALT 2012. LNCS (LNAI), vol. 7784, pp. 96–115. Springer, Heidelberg (2013)
3. Castelfranchi, C.: Commitments: From Individual Intentions to Groups and Organizations. In: Lesser, V.R., Gasser, L. (eds.) ICMAS, pp. 41–48. The MIT Press (1995)
4. Chopra, A.K., Dalpiaz, F., Giorgini, P., Mylopoulos, J.: Reasoning about Agents and Protocols via Goals and Commitments. In: International Conference on Autonomous Agents and Multiagent Systems, AAMAS, pp. 457–464 (2010)
5. Chopra, A.K., Singh, M.P.: Constitutive Interoperability. In: International Conference on Autonomous Agents and Multiagent Systems, AAMAS, pp. 797–804 (2008)
6. Chopra, A.K., Singh, M.P.: Multiagent Commitment Alignment. In: International Conference on Autonomous Agents and Multiagent Systems, AAMAS, pp. 937–944 (2009)
7. Desai, N., Chopra, A.K., Singh, M.P.: Amoeba: A Methodology for Modeling and Evolving Cross-organizational Business Processes. ACM Transactions on Software Engineering and Methodology 19, 6:1–6:45 (2009)
8. El-Menshawy, M., Bentahar, J., Dssouli, R.: A New Semantics of Social Commitments Using Branching Space-Time Logic. In: WI-IAT 2009: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, pp. 492–496 (2009)
9. Fornara, N., Colombetti, M.: Operational Specification of a Commitment-Based Agent Communication Language. In: International Conference on Autonomous Agents and Multiagent Systems, AAMAS, pp. 536–542 (2002)

10. Günay, A., Yolum, P.: Detecting Conflicts in Commitments. In: Sakama, C., Sardina, S., Vasconcelos, W., Winikoff, M. (eds.) DALT 2011. LNCS, vol. 7169, pp. 51–66. Springer, Heidelberg (2012)
11. Işıksal, A.: Use of Goals for Creating and Enacting Dynamic Contracts in Ambient Intelligence. Master's thesis, Bogazici University (2012)
12. Marengo, E., Baldoni, M., Baroglio, C., Chopra, A.K., Patti, V., Singh, M.P.: Commitments with Regulations: Reasoning about Safety and Control in REGULA. In: International Conference on Autonomous Agents and Multiagent Systems, AAMAS, pp. 467–474 (2011)
13. van Riemsdijk, M.B., Dastani, M., Winikoff, M.: Goals in Agent Systems: A Unifying Framework. In: International Conference on Autonomous Agents and Multiagent Systems, AAMAS, pp. 713–720 (2008)
14. Singh, M.P.: An Ontology for Commitments in Multiagent Systems. Artificial Intelligence and Law 7(1), 97–113 (1999)
15. Singh, M.P.: Information-Driven Interaction-Oriented Programming: BSPL, the Blindingly Simple Protocol Language. In: International Conference on Autonomous Agents and Multiagent Systems, AAMAS, pp. 491–498 (2011)
16. Telang, P.R., Yorke-Smith, N., Singh, M.P.: A Coupled Operational Semantics for Goals and Commitments. In: 9th International Workshop on Programming Multi-Agent Systems, ProMAS (2011)
17. Thangarajah, J., Padgham, L., Winikoff, M.: Detecting & Avoiding Interference Between Goals in Intelligent Agents. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 721–726 (2003)
18. Torroni, P., Chesani, F., Mello, P., Montali, M.: Social Commitments in Time: Satisfied or Compensated. In: Baldoni, M., Bentahar, J., van Riemsdijk, M.B., Lloyd, J. (eds.) DALT 2009. LNCS, vol. 5948, pp. 228–243. Springer, Heidelberg (2010)
19. Winikoff, M., Liu, W., Harland, J.: Enhancing Commitment Machines. In: Leite, J., Omicini, A., Torroni, P., Yolum, p. (eds.) DALT 2004. LNCS (LNAI), vol. 3476, pp. 198–220. Springer, Heidelberg (2005)
20. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative & procedural goals in intelligent agent systems. In: KR, pp. 470–481 (2002)
21. Yolum, P., Singh, M.P.: Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In: International Conference on Autonomous Agents and Multiagent Systems, AAMAS, pp. 527–534 (2002)