Hierarchical Planning about Goals and Commitments

Pankaj R. Telang
Dept. of Computer Science
North Carolina State University
Raleigh, NC 27695-8206
prtelang@ncsu.edu

Felipe Meneguzzi
School of Computer Science
Pontifical Catholic University of Rio Grande do Sul
Porto Alegre, RS 90619-900
Brazil
felipe.meneguzzi@pucrs.br

Munindar P. Singh
Dept. of Computer Science
North Carolina State University
Raleigh, NC 27695-8206
singh@ncsu.edu

ABSTRACT

We consider the problem of relating an agent's internal state (its beliefs and goals) and its social state (its commitments to and from other agents) as a way to develop a comprehensive account of decision making by agents in a multiagent system. We model this problem in terms of hierarchical task networks (HTNs) and show how HTN planning provides a natural representation and reasoning framework for goals and commitments. Our approach combines a domain-independent theory capturing the lifecycles of goals and commitments, generic patterns of reasoning, and domain models. Specifically, our approach shows how each agent may take into account its capabilities, costs, and preferences as it plans its interactions (captured as operations on commitments) with other agents to attempt to achieve its goals.

Categories and Subject Descriptors

H.1.0 [**Information Systems**]: Models and Principles—*General*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms

Keywords

Commitments, Goals, Planning, HTN, SHOP2

1. INTRODUCTION

Modeling interactions among agents is an important concern in multiagent systems. Specifically, researchers employ high-level constructs such as social commitments for modeling agent interactions. Such a model characterizes interactions in terms of their (social) meanings. Its well-defined set of commitment operations yields enactment flexibility to the agents. Further, an agent's social commitments relate naturally to its goals [3, 5, 17]. For example, an agent may create a commitment toward another to satisfy its goal, or adopt a goal to satisfy its commitment.

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.

Goals and commitments are declarative notions amenable to automated reasoning. The operations that an agent performs on goals and commitments are naturally hierarchical. For example, a goal may be decomposed into multiple conjunctive or disjunctive subgoals. The agent may satisfy a subgoal on its own, or may create a commitment toward another agent to satisfy it. If the commitment expires or is violated, then the agent may create a commitment toward some other agent, and sanction the violating agent. It is natural that an agent's plans be hierarchical: not only can operations on commitments feature within plans, but also to satisfy a commitment may require another hierarchical plan. Although multiple proposals for multiagent modeling based on goals and commitments exist, few approaches operationalize the resulting models. As an example, Winikoff [21] employs classical BDI plans to operationalize commitment-based interactions. However, Winikoff's approach ignores goals and fails to take advantage of the natural hierarchy of goal and commitment operations.

Approach.. This paper proposes a novel approach based on Hierarchical Task Network (HTN) planning for operationalizing goals and commitments using the semantics developed in our previous work [17]. HTN planning has been in used in significant practical scenarios such as job scheduling [12]. Our approach exploits the natural hierarchical structure of goals and commitments. We show how HTNs intuitively capture the plans that consider joint goal and commitment semantics.

Contributions.. Our proposed HTN-based approach offers three important advantages. First, our proposed approach offers clearer knowledge engineering than in the past. Second, our approach takes into account a notion of preference and cost that has been largely ignored in previous work on generating protocol enactments, though a few works, e.g., [10, 26], discuss preferences for analysis. We desire flexible protocols because they offer multiple choices to agents but clearly all the choices are not equal from an agent's perspective. Being able to handle costs is crucial to practical reasoning by agents, yet is ignored by current approaches, so our approach enables one to model the overall cost incurred by each realization of a set of goals and commitment to all agents within a system. Third, our approach lends itself to supporting modular abstractions by which increasingly flexible protocols may be generated by a layering of modules. In particular, it enables agent designers to easily evolve the plans in the light of changing needs.

Organization. Section 2 provides some historical motivation for our approach. Section 3 introduces goals, commitments, and HTN planning. Section 4 describes our approach for operationalizing goals and commitments via HTN planning. Section 5 applies our proposal to a well-known scenario. Section 6 discusses related work and outlines future directions.

2. HISTORICAL CONTEXT

We now briefly review the relevant literature to highlight the contributions of this paper. Traditional distributed computing representations of interaction focus on operational details. Such approaches, e.g., (message) sequence diagrams derived from agent-based extensions of the Unified Modeling Language (UML) [2, 14], violate the tenet of agent autonomy since they over-constrain how the agents exchange messages. This is a natural consequence of the procedural meaning such modeling constructs have in the underlying UML semantics. Because such approaches lack any business-level meaning for the messages, an agent has no basis for deviating from a prespecified flow without falling out of compliance [19]. Although traditional languages support expressing alternative operational executions, e.g., via the ALT construct in sequence diagrams, they do not provide any support for the generation of correct alternative executions. And, capturing multiple alternatives purely in operational terms becomes unwieldy fast. As a result, the specifications produced in practice are highly constrained, surprisingly often seen as a single sequence of messages.

Venkatraman and Singh [19] introduced the idea of a commitment protocol in conceptual terms along with the idea of each agent maintaining a view of its commitments to others and vice versa, as a way to verifying compliance. Yolum and Singh [23] developed the idea of explicit reasoning about commitments using the notion of a *commitment machine*, which supports the generation of a finite state machine. An agent could then participate in a commitment protocol without having to reason explicitly about commitments: thereby trading off flexibility for ease of implementation. Winikoff et al. [22] and Chopra and Singh [6] expanded the notion of commitment machines. Yolum and Singh [24, 25] introduced a commitment reasoner based on the event calculus with Chesani et al. [3] producing an approach for tracking commitment states based on the reactive event calculus. Recently, there has been increasing interest on relating goals and commitments [4, 5, 17], which seeks to close the gap between agents' goals (representing their mental state) and their commitments (representing their social state).

In simple terms, whereas the above works study the conceptual aspects of commitment protocols and goals, we show how to approach the subject from a more intuitive yet powerful technique. Specifically, reasoning about commitments and goals brings up the challenge of decision making by an agent as it adopts and enacts various commitment protocols so as to advance its goals. Although the earliest works on commitments, e.g., [23], did not model goals explicitly, they showed how an agent could reason about commitments, for example, by applying implemented frameworks such as the event calculus or causal logic [7, 24, 25]. The newer works on goals and commitments, e.g., [17], are abstract in that they define an operational semantics of the relation between goals and commitments, but do not provide algorithms to automatically process their operational semantics.

3. BACKGROUND

We now provide some essential background necessary to understand the contributions of this paper.

3.1 Running Example: Purchase Scenario

We illustrate our approach via a simple purchase scenario involving a merchant and a customer. The merchant has a goal of getting paid, and can provide goods, and the customer has a goal to get the goods, and can pay. The merchant and the customer can achieve their goals in various ways. As an example, the merchant may commit to the customer to providing the goods if the customer pays. Eventually the merchant may provide the goods to the customer (satisfying customer's goal for goods), and the customer may pay the merchant (satisfying merchant's goal to get paid) either before or after the merchant provides the goods. In another example, to achieve its goal for goods, the customer may manufacture the goods itself, and to achieve its goal of getting paid, the merchant may deposit funds in a bank and receive interest payments from the bank. However, it may be more costly for the customer to manufacture the goods than to procure the goods from the merchant, and the merchant's interest payments may not be as much as the merchant's profit by selling the goods to the customer.

Our HTN formalization enables the merchant and customer to generate a joint feasible plan to satisfy their goals. If the merchant or the customer prefer not to employ the generated plan, our formalization is capable of generating alternative plans.

3.2 Logic Language

We use a first-order logic language consisting of an infinite set of symbols for predicates, constants, functions, and variables, obeying the usual formation rules of first-order logic and following its usual semantics when describing planning domains [12].

Definition 1 (Term). *A term, denoted generically as τ, is a variable w, x, y, z (with or without subscripts); a constant a, b, c (with or without subscripts); or a function term $f(\tau_0, \ldots, \tau_n)$, where f is a n-ary function symbol applied to (possibly nested) terms $\tau_0, \ldots, \tau_n$.*

Definition 2 (Atomic formula). *A (first-order) atomic formula, denoted as φ, is a construct of the form $p(\tau_0, \ldots, \tau_n)$, where p is an n-ary predicate symbol and $\tau_0, \ldots, \tau_n$ are terms. A first-order formula Φ is recursively defined as $\Phi ::= \Phi \wedge \Phi' | \neg\Phi | \varphi$.*

We assume the usual abbreviations: $\Phi \vee \Phi'$ stands for $\neg(\neg\Phi \wedge \neg\Phi')$; $\Phi \rightarrow \Phi'$ stands for $\neg\Phi \vee \Phi'$ and $\Phi \leftrightarrow \Phi'$ stands for $(\Phi \rightarrow \Phi') \wedge (\Phi' \rightarrow \Phi)$. Additionally, we also adopt the equivalence $\{\Phi_1, \ldots, \Phi_n\} \equiv (\Phi_1 \wedge \cdots \wedge \Phi_n)$ and use these interchangeably. Our mechanisms use first-order unification [1], which is based on the concept of substitutions.

Definition 3 (Substitution). *A substitution σ is a finite and possibly empty set of pairs $\{x_1/\tau_1, \ldots, x_n/\tau_n\}$, where $x_1, \ldots, x_n$ are distinct variables and each τ_i is a term such that $\tau_i \neq x_i$.*

Given an expression E and a substitution $\sigma = \{x_1/\tau_1, \ldots, x_n/\tau_n\}$, we use $E\sigma$ to denote the expression obtained from E by simultaneously replacing each occurrence of x_i in E with τ_i, for all $i \in \{1, \ldots, n\}$.

Unifications can be *composed*; that is, for any substitutions $\sigma_1 = \{x_1/\tau_1, \ldots, x_n/\tau_n\}$ and $\sigma_2 = \{y_1/\tau'_1, \ldots, y_k/\tau'_k\}$, their composition, denoted as $\sigma_1 \cdot \sigma_2$, is defined as $\{x_1/(\tau_1 \cdot \sigma_2), \ldots, x_n/(\tau_n \cdot \sigma_2), z_1/(z_1 \cdot \sigma_2), \ldots, z_m/(z_m \cdot \sigma_2)\}$, where $\{z_1, \ldots, z_m\}$ are those variables in $\{y_1, \ldots, y_k\}$ that are not in $\{x_1, \ldots, x_n\}$. A substitution σ is a *unifier* of two terms τ_1, τ_2, if $\tau_1 \cdot \sigma = \tau_2 \cdot \sigma$.

Definition 4 (Unify Relation)**.** *Relation* $unify(\tau_1, \tau_2, \sigma)$ *holds iff* $\tau_1 \cdot \sigma = \tau_2 \cdot \sigma$*. Moreover,* $unify(p(\tau_0, \ldots, \tau_n), p(\tau'_0, \ldots, \tau'_n), \sigma)$ *holds iff* $unify(\tau_i, \tau'_i, \sigma)$*, for all* $0 \leq i \leq n$*.*

Thus, two terms τ_1, τ_2 are related through the *unify* relation if there is a substitution σ that makes the terms syntactically equal. In our representation and algorithms, we adopt Prolog's convention [1] and use strings starting with a capital letter to represent variables and strings starting with a small letter to represent constants.

3.3 Commitments

Social commitments are extensively studied in multiagent literature [9, 11, 20]. Specifically, a commitment C(DEBTOR, CREDITOR, antecedent, consequent) means that a DEBTOR agent commits to a CREDITOR agent to bring about the consequent if the antecedent holds [16]. For example, in the purchase scenario, the customer commits to the merchant to paying if the merchant provides the goods: C(CUSTOMER, MERCHANT, goods, pay).

Figure 1 shows the commitment lifecycle [17]. Before a commitment is created, it is in state *null*. When the debtor creates a commitment, the commitment enter the state *active*, which consists of two substates: conditional and detached. An active commitment is *conditional* when its antecedent is false, and *detached* when its consequent is true. An active commitment expires if its antecedent fails. If the consequent of an active commitment is brought about, then the commitment satisfies. An active commitment becomes *pending* when the debtor suspends it (e.g., to redirect its resources to another more pressing goal or commitment), and a pending commitment becomes active when the debtor reactivates it. If the debtor cancels or the creditor releases a conditional commitment, the commitment is *terminated*. If the debtor cancels a detached commitment, then the commitment is *violated*.

3.4 Goals

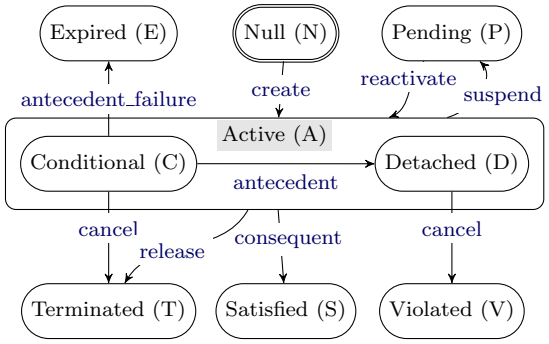

Figure 1: Commitment life cycle as a state transition diagram.

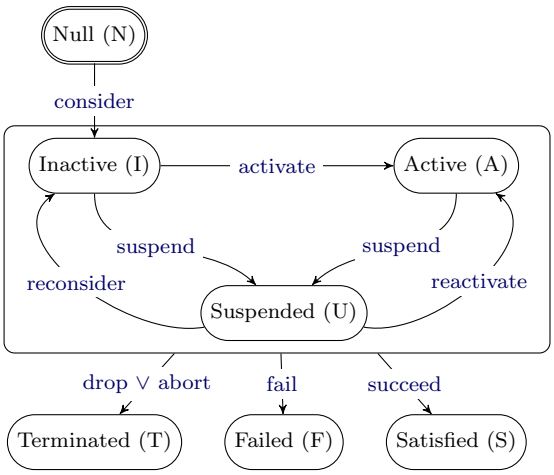

Figure 2: Goal lifecycle as a state transition diagram.

A goal is a state of the world that an agent wishes to bring about. We formulate a goal as: $G = \mathsf{G}(x, pg, s, f)$, where x is an agent, pg is the precondition of the goal, the truth of which is required for the goal to be considered [17, 18]. s is the success condition of G, and f is the failure condition. A goal succeeds if s holds without f holding. For example, in the purchase scenario, the customer has a goal of procuring the goods before a deadline: G(CUSTOMER, needsGoods, goods, deadline). We note that, to improve readability, we have slightly simplified the formulation of goals from [17] by conflating the *success condition* of the goal with its effect, as well as the *in-condition* with the negation of the failure condition.

Figure 2 shows our goal lifecyle [17]. A goal is in state *null* before it is considered, if the goal's precondition is true, then a goal may be considered. When the agent considers a goal, it becomes *inactive*. The agent may activate an inactive goal, in which case the goal becomes *active*. The agent may suspend an inactive or an active goal, which will transition the goal to the *suspended* state. If the agent reconsiders a suspended goal, the goal transitions to the inactive state, and if the agent reactivates a *suspended* goal, the goal transitions to the active state. When the goal is inactive, active, or suspended: (a) if the agent drops or terminates the goal, then the goal transitions to state *terminated*; (b) if the failure condition holds, then the goal transitions to the state *failed*; and (c) if the success condition holds, then the goal transitions to the state *satisfied*.

3.5 Classical and HTN Planning

STRIPS-style planning defines a problem in terms of an *initial state* and a *goal state*—both specified as sets of ground atoms—and a set of *operators*. An operator has a *precondition* encoding the conditions under which the operator can be used, and a *postcondition* encoding the outcome of applying the operator. Planning is concerned with sequencing actions obtained by instantiating operators describing state transformations. In our representation, an *operator* o is a five-tuple $\langle name(o), pre(o), del(o), add(o), cost(o)\rangle$, where (1) $name(o) = act(\vec{x})$, the name of the operator, is a symbol followed by a vector of distinct variables such that all free

variables in $pre(o), del(o)$, and $add(o)$ also occur in $act(\vec{x})$; (2) $pre(o), del(o)$ and $add(o)$ called, respectively, the precondition, *delete-list* and *add-list*, are sets of atoms where the add and delete lists are disjoint; and (3) $cost(o)$ is a numeric expression representing the cost of executing an operator. The delete-list specifies which atoms should be removed from the state of the world when the operator is applied, and the add-list specifies which atoms should be added to the state of the world when the operator is applied.

A Hierarchical Task Network (HTN) planner generates a plan by successive refinements of sets of *tasks*. Tasks are classified into *primitive* (à la individual operators in STRIPS-style planning) and *compound* (abstract high-level tasks). An HTN planner recursively decomposes compound tasks by using a designer-specified library of *methods* until only primitive tasks remain. Methods are elements of domain knowledge that describe how a higher-level task can be decomposed into more primitive tasks, they constrain the search space making HTN planning more efficient. For example, in the purchase scenario, the customer's higher-level task, *achieveGoal*, to achieve its goal for procuring the goods could be decomposed into primitive tasks of creating a commitment, C(CUSTOMER, MERCHANT, goods, pay), toward the merchant, and of paying the merchant after the merchant provides the goods.

Formally [12], an HTN *planning problem* $\mathcal{P}$ is a tuple $(d, \mathbf{I}, \mathcal{D})$, where (1) d is a *task network*, (2) $\mathbf{I}$ is an *initial state*, and (3) $\mathcal{D}$ is an HTN *planning domain*. The planning domain $\mathcal{D}$ is a tuple $(\mathcal{A}, \mathcal{M})$, respectively, finite sets of operators and methods. A *task network* $\mathcal{H}$ is a tuple (T, C), where T is a finite set of tasks (primitive and compound), and C is a set of partial ordering constraints on tasks in T. A constraint specifies the order in which certain tasks can be executed, and can be either a precedes or a succeeds relation. For example, $t_i \prec t_j$ (equivalently $t_i \succ t_j$) means that t_i must be executed before t_j. A task has a precondition that must hold under some substitution before the task can be executed. Corresponding to each primitive task t, an operator exists in the planning domain, that is, $\mathcal{A} \subseteq T$. The operator specifies the effect on the world state if the task is executed. Corresponding to each compound task t, a method exists in the planning domain, that is, $\mathcal{A} \subseteq M$. A method m is a tuple $(t, s, \mathcal{H}')$, where s is a precondition that must hold for a task t to be refined into another task network $\mathcal{H}' = (T', C')$. Note that $\mathcal{A} \cup \mathcal{M} = T$: all operators in $\mathcal{A}$ and all methods in $\mathcal{M}$ are in the task set T.

4. FORMALIZATION WITHIN HTN

We now formalize Telang et al.'s [17] operational semantics as an HTN planning domain. Doing so, together with domain-specific operations, enables us to use an HTN planner such as JSHOP2 [15] to automatically synthesize correct protocols to achieve an individual agent's goals either in isolation (Section 4.1) or via commitments among multiple agents (Section 4.2). We finish with the formalization of the HTN methods required to generate goal-commitment protocols in Section 4.3. We adopt JSHOP2's convention of naming primitive tasks with an initial exclamation mark (!).

4.1 Goal and Commitment Dynamics

As agents interact, their commitments and goals progress in a systematic manner. In essence, the *dynamics* of commitments and goals are captured using their lifecycles, as presented in Figures 1 and 2. Note that only some of the transitions in Figures 1 and 2 are under explicit control of an agent. For example, an agent can explicitly *create* a commitment, thus transitioning it to the *active* state. However, once a commitment is active, transitions to the *satisfied* and *expired* occur based on what transpires or fails to transpire in the environment. Notice that the discharge of a commitment depends solely on the consequent becoming true, which could happen because of an action by any of the agents or through some environmental process. In settings where we wish to ensure that the debtor performs the action that brings about the consequent, we could specify the consequent to incorporate the debtor, e.g., paid(John, $1) instead of paid($1).

We formalize the dynamics using the HTN planning framework. This formalization seeks to support operations that an agent can explicitly execute to manipulate its internal representations of the current state of a goal or commitment. To capture the above intuition regarding the agent's control, we define the dynamics in two parts: (1) a set of logical axioms characterizing the states of a goal and of a commitment; and (2) a set of planning operators that induce the state changes over which the agent has direct control.

The states of a commitment C are defined logically via (1) five predicates corresponding to lifecycle states ($null(C)$, $pending(C)$, $canceled(C)$, $released(C)$, and $expired(C)$) and (2) six axioms corresponding to state transitions, where D represents the debtor of the commitment, and A represents the creditor of the commitment:

⟨**operator** : !$create(C, D, A)$,
pre :($commitment(C, D, A) \wedge null(C)$),
del :($null(C)$), **add** :(),
cost :0⟩
⟨**operator** : !$suspend(C, D, A)$,
pre :($commitment(C, D, A) \wedge active(C)$),
del :(), **add** :($pending(C)$),
cost :1⟩
⟨**operator** : !$reactivate(C, D, A)$,
pre :($commitment(C, D, A) \wedge pending(C)$),
del :($pending(C)$), **add** :(),
cost :1⟩
⟨**operator** : !$expire(C, D, A)$,
pre :($commitment(C, D, A) \wedge conditional(C) \wedge timeout(C)$),
del :(), **add** :($expired(C)$),
cost :5⟩
⟨**operator** : !$cancel(C, D, A)$,
pre :($commitment(C, D, A) \wedge active(C)$),
del :(), **add** :($canceled(C)$),
cost :10⟩
⟨**operator** : !$release(C, D, A)$,
pre :($commitment(C, D, A) \wedge active(C)$),
del :(), **add** :($released(C)$),
cost :1⟩

Here, $p(C)$ and $q(C)$ capture the antecedent and consequent, respectively, of C being true: these would be substituted by domain-specific predicates, e.g., goods may feature as the antecedent in some commitment. Note that the *terminal* axiom is merely a shortcut for the set of states from which a commitment cannot move out. Moreover, commitment (and later goal) types are introduced to allow the existence of multiple instances of a commitment with the same conditions.

$p(C) \leftarrow commitment(C, D, A) \wedge commitmentType(C, c1) \wedge payc$
$q(C) \leftarrow commitment(C, D, A) \wedge commitmentType(C, c1) \wedge goodsc$

The remaining commitment states are defined via the following axioms:

$$\begin{aligned}
\mathit{conditional}(C) &\leftarrow \mathit{active}(C) \wedge \neg p(C)\\
\mathit{detached}(C) &\leftarrow \mathit{active}(C) \wedge p(C)\\
\mathit{active}(C) &\leftarrow \neg \mathit{null}(C) \wedge \neg \mathit{terminal}(C) \wedge\\
&\quad \neg \mathit{pending}(C) \wedge \neg \mathit{satisfied}(C)\\
\mathit{terminated}(C) &\leftarrow (\neg p(C) \wedge \mathit{canceled}(C)) \vee \mathit{released}(C)\\
\mathit{violated}(C) &\leftarrow (p(C) \wedge \mathit{canceled}(C)) \vee \neg p(C)\\
\mathit{satisfied}(C) &\leftarrow \neg \mathit{null}(C) \wedge \neg \mathit{terminal}(C) \wedge p(C) \wedge q(C)\\
\mathit{terminal}(C) &\leftarrow \mathit{commitment}(C, D, A) \wedge\\
&\quad (\mathit{canceled}(C) \vee \mathit{released}(C) \vee \mathit{expired}(C))
\end{aligned}$$

Next, we formalize the dynamics of goals. As for commitments, the states of a goal G from agent A are defined in terms of (1) five predicates corresponding to lifecycle states ($\mathit{null}(G)$, $\mathit{activatedG}(G)$, $\mathit{suspendedG}(G)$, $\mathit{dropped}(G)$, and $\mathit{aborted}(G)$) and (2) five axioms corresponding to state transitions:

$\langle$**operator** : !$\mathit{consider}(G, A)$,
pre :($\mathit{goal}(G, A) \wedge \mathit{null}(G) \wedge \mathit{pg}(G)$),
del :($\mathit{null}(G)$), **add** :(),
cost :1$\rangle$
$\langle$**operator** : !$\mathit{activate}(G, A)$,
pre :($\mathit{goal}(G, A) \wedge \mathit{inactiveG}(G)$),
del :(), **add** :($\mathit{activatedG}(G)$),
cost :1$\rangle$
$\langle$**operator** : !$\mathit{suspend}(G, A)$,
pre :($\mathit{goal}(G, A) \wedge \neg \mathit{terminalG}(G) \wedge \neg \mathit{null}(G)$),
del :($\mathit{activatedG}(G)$), **add** :($\mathit{suspendedG}(G)$),
cost :1$\rangle$
$\langle$**operator** : !$\mathit{reconsider}(G, A)$,
pre :($\mathit{goal}(G, A) \wedge \mathit{suspendedG}(G) \wedge \neg \mathit{terminalG}(G) \wedge$
$\neg \mathit{null}(G)$),
del :(), **add** :($\mathit{suspendedG}(G)$),
cost :1$\rangle$
$\langle$**operator** : !$\mathit{reactivate}(G, A)$,
pre :($\mathit{goal}(G, A) \wedge \mathit{suspendedG}(G) \wedge \neg \mathit{terminalG}(G) \wedge$
$\neg \mathit{null}(G)$),
del :($\mathit{activatedG}(G)$), **add** :($\mathit{suspendedG}(G)$),
cost :1$\rangle$
$\langle$**operator** : !$\mathit{drop}(G, A)$,
pre :($\mathit{goal}(G, A) \wedge \neg \mathit{terminalG}(G) \wedge \neg \mathit{null}(G)$),
del :(), **add** :($\mathit{dropped}(G)$),
cost :1$\rangle$
$\langle$**operator** : !$\mathit{abort}(G, A)$,
pre :($\mathit{goal}(G, A) \wedge \neg \mathit{terminalG}(G) \wedge \neg \mathit{null}(G)$),
del :(), **add** :($\mathit{aborted}(G)$),
cost :1$\rangle$

Here, $pg(G)$, $s(G)$ and f respectively capture the precondition, success condition, and failure condition of G. For example, for goal $G(\mathit{payc}, \mathit{goodsc}, \mathit{deadlinec})$, we could encode the following axioms:

$$\begin{aligned}
\mathit{pg}(G) &\leftarrow \mathit{goal}(G, A) \wedge \mathit{goalType}(G, g3) \wedge \mathit{payc}\\
\mathit{s}(G) &\leftarrow \mathit{goal}(G, A) \wedge \mathit{goalType}(G, g3) \wedge \mathit{goodsc}\\
\mathit{f}(G) &\leftarrow \mathit{goal}(G, A) \wedge \mathit{goalType}(G, g3) \wedge \mathit{deadlinec}
\end{aligned}$$

The remaining goal states are defined via these axioms:

$$\begin{aligned}
\mathit{inactiveG}(G) &\leftarrow \neg \mathit{null}(g) \wedge \neg f(G) \wedge \neg s(G) \wedge\\
&\quad \neg \mathit{terminalG}(G) \wedge \neg \mathit{suspendedG}(G) \wedge\\
&\quad \neg \mathit{activeG}(G)\\
\mathit{activeG}(G) &\leftarrow \mathit{activatedG}(G) \wedge \neg f(G) \wedge \neg \mathit{satisfiedG}(G)\\
&\quad \wedge \neg \mathit{terminalG}(G) \wedge \neg \mathit{suspendedG}(G)\\
\mathit{satisfiedG}(G) &\leftarrow \neg \mathit{null}(G) \wedge \neg \mathit{terminal}(G) \wedge \mathit{pg}(G)\\
&\quad \wedge s(G) \wedge \neg f(G)\\
\mathit{failedG}(G) &\leftarrow \neg \mathit{null}(G) \wedge f(G)\\
\mathit{terminatedG}(G) &\leftarrow \neg \mathit{null}(G) \wedge (\mathit{dropped}(G) \vee \mathit{aborted}(G))\\
\mathit{terminalG}(G) &\leftarrow \mathit{goal}(G, A) \wedge (\mathit{dropped}(G) \vee \mathit{aborted}(G))
\end{aligned}$$

4.2 Relating Goals to Commitments

We now define the methods corresponding to the practical reasoning rules for goals and commitments [17]. Each rule is written $\frac{guard}{transition}$, where $guard$ is a query on the state, and $transition$ consists of one or more transitions in the state of a goal or commitment. To map such a rule to the HTN formalism. we define a compound task for each rule identifier and express each rule as an HTN method that decomposes this task into primitive transitions to the state of a goal or commitment. The guard of the rule becomes the precondition of the method, and the task network contains the primitive action corresponding to the transition. (Below, the superscript on a C or a G refers to the state in the corresponding lifecycle as shown in Figures 1 and 2, respectively.) For example, the ENTICE rule

$$\frac{\langle G^A, C^N \rangle}{\mathit{create}(C)}\text{ENTICE}$$

becomes the following method:

(**method** : $\mathit{entice}(G, C, D, A)$,
pre :($\mathit{goal}(G, D) \wedge \mathit{activeG}(G) \wedge$
$\mathit{commitment}(C, D, A) \wedge \mathit{null}(C) \wedge (s(G) \leftrightarrow p(C))$),
tn :(!$\mathit{create}(C, D, A)$))
(**method** : $\mathit{deliver}(G, C, D, A)$,
pre :($\mathit{goal}(G, D) \wedge \mathit{null}(G) \wedge$
$\mathit{commitment}(C, D, A) \wedge \mathit{detached}(C)$),
tn :(!$\mathit{consider}(G, D)$, !$\mathit{activate}(G, D)$))

For brevity, we omit the methods for the remaining practical rules though we use them in our running examples.

4.3 Bringing it all Together

For an HTN planner to generate valid plans to achieve an agent's goals individually or through commitment protocols, we need methods to decompose an agent's goals. Table 1 formalizes methods for decomposing all achievable goals in a domain into commitment protocols. The first method in Table 1 recursively tries to activate all possible goals for an agent. Notice that we omit domain dependent methods to achieve goals by individual agents without help from others, but in many domains, agents will have a choice between executing a single agent plans and using commitments to get other agents to help them.

If a certain goal is already active, the second method tries to decompose such a goal into an individually initiated plan to achieve that goal. Such a plan may consist of an individual plan: a designer must create a domain-dependent plan through an additional method to decompose the *achieveGoal* compound task. Or, the plan may consist

of a generic commitment protocol: use the last method in Table 1 to create a commitment protocol based on enticing another agent A_2 to adopt a commitment whose antecedent satisfies the goal, and whose consequent is a goal of A_2. If A_2 achieves A_1's goal, this commitment detaches, in which case A_1 delivers the promised consequent. Finally, the third method in Table 1 caters to two agents A_1 and A_2 whose goals can be achieved by mutual commitments. If both agents commit as appropriate, when A_1 detaches its commitment, the goal of A_2 is achieved, and vice versa.

Table 1: Methods to achieve goals through commitments.

(**method** : $achieveGoals$,
  pre :$(goal(G,A) \wedge pg(G) \wedge \neg activeG(G))$,
  tn :$(!consider(G,A), !activate(G,A), achieveGoals))$
(**method** : $achieveGoals$,
  pre :$(goal(G,A) \wedge activeG(G))$,
  tn :$(achieveGoal(G,A)))$
(**method** : $achieveGoals$,
  pre :$(goal(G_1,A_1) \wedge activeG(G_1) \wedge$
    $goal(G_2,A_2) \wedge activeG(G_2) \wedge$
    $commitment(C_1,A_1,A_2) \wedge (s(G_1) \leftrightarrow p(C_1)) \wedge$
    $commitment(C_2,A_2,A_1) \wedge (s(G_2) \leftrightarrow p(C_2)))$,
  tn :$(\{entice(G_1,C_1,A_1,A_2), entice(G_2,C_2,A_2,A_1)\},$
    $\{detach(C_1), detach(C_2)\}))$
(**method** : $achieveGoals$,
  pre :$(\top)$,
  tn :$())$
(**method** : $achieveGoal(G_1,A_1)$,
  pre :$(goal(G_1,A_1) \wedge activeG(G_1) \wedge$
    $commitment(C,A_1,A_2) \wedge (s(G_1) \leftrightarrow p(C)) \wedge$
    $goal(G_2,A_1) \wedge (s(G_2) \leftrightarrow q(C)) \wedge (G_1 \neq G_2))$,
  tn :$(entice(G_1,C,A_1,A_2), detach(C), deliver(G_2,C,A_1,A_2),$
    $achieveGoal(G_2,A_2)))$

5. APPLYING THE HTN FORMALIZATION

We illustrate the HTN formalization on the purchase scenario. Table 3 summarizes the goals and commitments from this example. The HTN formalization for the purchase scenario reuses the domain-independent methods, operators, and axioms from Section 4. We add four domain-dependent operators to the formalization.

manufactureGoods, with success condition *goods*: the customer manufactures goods on its own.

earnInterest, success condition *pay*: the merchant deposits money in a bank to earn interest.

sendPayment, with success condition *pay*: the customer pays the merchant.

sendGoods, with success condition *goods*: the merchant sends goods to the customer.

For simplicity, we use an absolute number for cost instead of an expression. We set the cost of manufactureGoods higher than sendPayment and of earnInterest higher than sendGoods. Here we assume the costs are centrally set; normally it is ill-founded to compare preferences across agents.

Figure 3 shows a portion of the HTN tree rooted at the achieveGoals method, showing it invokes two methods. achieveGoal(C, G_{cg}) and achieveGoal(M, G_{mp}) are the customer and merchant's methods to achieve their goals G_{cg} and G_{mp}, respectively. There are two ways for the customer to decompose achieveGoal(C, G_{cg}): (1) manufactureGoods or manufacture the goods on its own; and (2) ENTICE(G_{cg}, C_{cm}, C, M) or entice the merchant to manufacture the goods. The ENTICE method invokes the *create*(C_{cm}) operator to create a commitment. Since the commitment C_{cm} is active, and the merchant has a goal G_{mp} to get paid, the detach(C_{cm}) method is invoked, which invokes the operator *activate*(G_{mg}), which in turn invokes the domain-dependent operator goods.

Table 2: Domain-specific axioms and methods for the merchant scenario.

$pg(G) \leftarrow goal(G,A) \wedge goalType(G,G_{mp}) \wedge \top$
$s(G) \leftarrow goal(G,A) \wedge goalType(G,G_{mp}) \wedge pay$
$f(G) \leftarrow goal(G,A) \wedge goalType(G,G_{mp}) \wedge deadline$
$pg(G) \leftarrow goal(G,A) \wedge goalType(G,G_{cg}) \wedge needsgoods$
$s(G) \leftarrow goal(G,A) \wedge goalType(G,G_{cg}) \wedge goods$
$f(G) \leftarrow goal(G,A) \wedge goalType(G,G_{cg}) \wedge deadline$
$pg(G) \leftarrow goal(G,A) \wedge goalType(G,G_{mg}) \wedge pay$
$s(G) \leftarrow goal(G,A) \wedge goalType(G,G_{mg}) \wedge goods$
$f(G) \leftarrow goal(G,A) \wedge goalType(G,G_{mg}) \wedge deadline$
$pg(G) \leftarrow goal(G,A) \wedge goalType(G,G_{cp}) \wedge goods$
$s(G) \leftarrow goal(G,A) \wedge goalType(G,G_{cp}) \wedge pay$
$f(G) \leftarrow goal(G,A) \wedge goalType(G,G_{cp}) \wedge deadline$
$p(C) \leftarrow commitment(C,D,A) \wedge commitmentType(C,C_{mc}) \wedge pay$
$q(C) \leftarrow commitment(C,D,A) \wedge commitmentType(C,C_{mc}) \wedge goods$
$p(C) \leftarrow commitment(C,D,A) \wedge commitmentType(C,C_{cm}) \wedge goods$
$q(C) \leftarrow commitment(C,D,A) \wedge commitmentType(C,C_{cm}) \wedge pay$

$\langle$**operator** : $!sendGoods(A,D)$,
  pre :$(\neg goods \wedge (A=m) \wedge (D=c))$,
  del :$(), $**add** :$(goods)$,
  cost :$1\rangle$
$\langle$**operator** : $!sendPayment(A,D)$,
  pre :$(\neg payc \wedge (A=c) \wedge (D=m))$,
  del :$(), $**add** :$(payc)$,
  cost :$1\rangle$
$\langle$**operator** : $!manufactureGoods(A)$,
  pre :$(\neg goodsc \wedge (A=c))$,
  del :$(), $**add** :$(goods)$,
  cost :$10\rangle$
(**method** : $achieveGoal(G)$,
  pre :$(G = G_{cg} \wedge goal(G,c) \wedge activeG(G))$,
  tn :$(!manufactureGoods(c)))$
(**method** : $achieveGoal(G)$,
  pre :$(G = G_{mg} \wedge goal(G,m) \wedge activeG(G))$,
  tn :$(!sendGoods(m,c)))$
(**method** : $achieveGoal(G)$,
  pre :$(G = G_{cp} \wedge goal(G,c) \wedge activeG(G))$,
  tn :$(!sendPayment(c,m)))$
(**method** : $detach(C)$,
  pre :$(C = C_{mc} \wedge active(C))$,
  tn :$(!sendPayment(c,m)))$
(**method** : $detach(C)$,
  pre :$(C = C_{cm} \wedge active(C))$,
  tn :$(!sendGoods(c,m)))$

Table 3: Goals and commitments from the purchase example.

Id	Commitment or Goal
G_{mp}	G(MERCHANT, $\top$, pay, deadline)
G_{cg}	G(CUSTOMER, needsGoods, goods, deadline)
G_{mg}	G(MERCHANT, pay, goods, deadline)
G_{cp}	G(CUSTOMER, goods, pay, deadline)
C_{mc}	C(MERCHANT, CUSTOMER, pay, goods)
C_{cm}	C(CUSTOMER, MERCHANT, goods, pay)

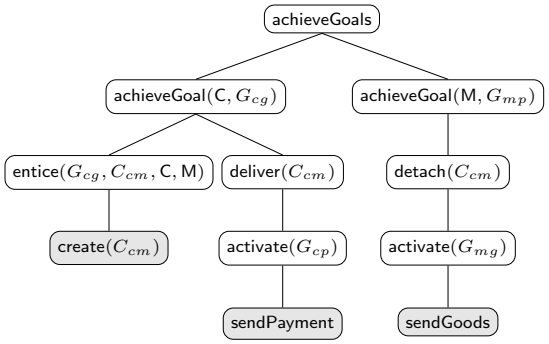

Figure 3: A decomposition tree for the purchase scenario.

That satisfies G_{mg} and G_{cg}, and detaches C_{cm}. Once the commitment C_{cm} is detached, the deliver(C_{cm}) method is invoked, which in turn invokes the *activate*(G_{cp}) operator. Finally, the domain-dependent operator pay is invoked, which satisfies G_{cp}, G_{mp}, and C_{cm}. Following this path, we obtain the plan: *create*(C_{cm}), *activate*(G_{mg}), goods, *activate*(G_{cp}), pay. The total cost of this plan is four. A second possible plan (not shown in Figure 3) is: manufactureGoods, earnInterest. These two operators satisfy the achieveGoal(M, G_{mp}) and achieveGoal(C, G_{cg}). The cost of this plan is 40.

These plans and their costs represent concrete realizations for the goals and commitments defined in the agent system modeled, and serve two purposes. First, the existence of plans that achieve the goals and satisfy commitments represents a proof of realizability for the specified agent system. Second, when multiple plans exist, their individual costs enable agents to reason about optimal realizations of its goals either by an agent on its own, or through commitments to other agents.

6. CONCLUSIONS AND DIRECTIONS

The main contribution of this paper is a novel HTN planning-based approach for operationalizing goals and commitments. Our formalization for goals and commitments exploits existing off-the-shelf HTN planners to efficiently generate a plan for an individual agent or a commitment protocol. Moreover, using our formalization, once can use such a planner as an efficient validation tool for business protocols as they are designed. Our approach offers advantages over existing approaches, e.g., [13, 23]. First, it supports the generation of human-readable protocols. Second, it reasons about the costs of protocols to optimize them before execution. Third, the methods we define can be converted into plan libraries for traditional agent programming languages, specifically those that use HTN planning [8]. Fourth, although compiling a commitment protocol may result in a large HTN domain, HTN planning itself is recognized as one of the most efficient planning formalisms, being able to generate solutions to planning problems that have extremely large state spaces.

6.1 Related Work

Chopra and Singh [6] employ $\mathcal{C}+$, an action description language, to model commitment protocols so they can be contextually adapted. Günay et al. [13] propose an algorithm to automatically create commitment protocols that would achieve agent goals by matching goals to local capabilities and services from third parties. By contrast, our formalization applies an HTN planner such as JSHOP2 not only to create such plans and commitment protocols, but also to optimize the results based on the cost of the operators. Further, it supports visualizing the generation of a protocol in a readable way. Günay et al. reason about an agent's capabilities explicitly during protocol generation whereas we explicitly enumerate the capabilities or services that satisfy each goal. In this respect, the approach is complementary to our approach.

Chopra et al. [4, 5] develop a semantic relationship between goals and commitments. Their approach can verify if a set of commitments supports achieving a set of agent goals, and if a set of agent goals supports satisfying a set of commitments. In contrast, given a set of goals and commitments, our approach produces operational level and feasible plans that lead to satisfaction of the goals and commitments.

It is instructive to compare our approach to Telang et al.'s [17] approach. We adopt their practical rules but add cost. Telang et al. present a table showing one possible execution, identifying the specific practical rules and agent states that occur at each step in the execution. Our approach offers the following relative advantages. First, we consider all possible executions if we disregard the cost. Second, we identify a jointly optimal plan, which ensures that the sum of the costs for each agent is minimal, and can generate additional executions of increasing cost as needed.

6.2 Directions

Our approach brings cost into consideration for reasoning about goals and commitments. However, it assumes a fully cooperative setting that takes into account the total cost over all participants. A natural and important research challenge is to extend the approach to handle self-interested agents who might wish to optimize for themselves individually even if the overall cost of the plan is increased as a result. Such agents may negotiate with each other or may even disclose their costs strategically. We hope to pursue such considerations in future work.

A natural important challenge is to apply our approach to address the design-time challenge, which involves reasoning about a protocol so as to generate an operational representation. Previous approaches can produce finite state machine representations. Such representations do not provide a basis for incorporating costs to limit the execution paths that would be suitable for a particular agent based on the costs they can bear.

Finally, although our approach can be used to generate protocol realizations when simultaneous goals and commitments do not conflict, and detect when conflicts make realizations impossible, it cannot currently *resolve* such conflicts automatically (e.g., suggest different methods to overcome conflicts). Therefore, we leave the design of planning tools to overcome domain-knowledge conflicts as future work.

Acknowledgments

We thank the anonymous reviewers for their helpful comments.

7. REFERENCES

[1] K. R. Apt. *From Logic Programming to Prolog.* Prentice-Hall, U.K., 1997.

[2] B. Bauer and J. Odell. UML 2.0 and agents: How to build agent-based systems with the new UML standard. *Engineering Applications of Artificial Intelligence*, 18(2):141–157, Mar. 2005.

[3] F. Chesani, P. Mello, M. Montali, and P. Torroni. Commitment tracking via the reactive event calculus. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 91–96, Pasadena, California, 2009. IJCAI.

[4] A. K. Chopra, F. Dalpiaz, P. Giorgini, and J. Mylopoulos. Modeling and reasoning about service-oriented applications via goals and commitments. In *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 417–421, 2010.

[5] A. K. Chopra, F. Dalpiaz, P. Giorgini, and J. Mylopoulos. Reasoning about agents and protocols via goals and commitments. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 457–464, Toronto, 2010. IFAAMAS.

[6] A. K. Chopra and M. P. Singh. Contextualizing commitment protocols. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1345–1352, Hakodate, Japan, May 2006. ACM Press.

[7] A. K. Chopra and M. P. Singh. Contextualizing commitment protocols. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1345–1352, Hakodate, Japan, May 2006. ACM Press.

[8] L. de Silva, S. Sardina, and L. Padgham. First principles planning in BDI systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems – Volume 2*, AAMAS, pages 1105–1112, 2009.

[9] N. Desai, A. K. Chopra, and M. P. Singh. Amoeba: A methodology for modeling and evolution of cross-organizational business processes. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 19(2):6:1–6:45, Oct. 2009.

[10] N. Desai, N. C. Narendra, and M. P. Singh. Checking correctness of business contracts via commitments. In *Proceedings of the 7th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 787–794, Estoril, Portugal, May 2008. IFAAMAS.

[11] N. Fornara and M. Colombetti. Ontology and time evolution of obligations and prohibitions using semantic web technology. In *Proceedings of the 7th AAMAS Workshop on Declarative Agent Languages and Technologies (DALT)*, pages 101–118, 2009.

[12] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

[13] A. Günay, M. Winikoff, and P. Yolum. Commitment protocol generation. In *Proceedings of the 10th AAMAS Workshop on Declarative Agent Languages and Technologies (DALT)*, pages 51–66, 2012.

[14] B. Hofreiter, C. Huemer, P. Liegl, R. Schuster, and M. Zapletal. UN/CEFACT's Modeling Methodology (UMM): A UML profile for B2B e-commerce. In *Proceedings of the 2nd International Workshop on Best Practices of UML (ER)*, pages 19–31, 2006.

[15] O. Ilghami and D. S. Nau. A general approach to synthesize problem-specific planners. Technical report, University of Maryland, 2003.

[16] M. P. Singh. Semantical considerations on dialectical and practical commitments. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*, pages 176–181, Chicago, July 2008. AAAI Press.

[17] P. R. Telang, N. Yorke-Smith, and M. P. Singh. Relating goal and commitment semantics. In *Proceedings of the 9th International Workshop on Programming Multiagent Systems (ProMAS 2011)*, volume 7217 of *LNCS*, pages 22–37, Taipei, 2012. Springer.

[18] J. Thangarajah, J. Harland, D. Morley, and N. Yorke-Smith. Operational behaviour for executing, suspending and aborting goals in BDI agent systems. In *Declarative Agent Languages and Technologies VII, Revised Selected and Invited Papers*, volume 6618 of *LNCS*, pages 1–21. Springer, 2011.

[19] M. Venkatraman and M. P. Singh. Verifying compliance with commitment protocols: Enabling open Web-based multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, Sept. 1999.

[20] M. Verdicchio and M. Colombetti. Commitments for agent-based supply chain management. *SIGecom Exchanges*, 3(1):13–23, 2002.

[21] M. Winikoff. Implementing commitment-based interaction. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 868–875, Honolulu, May 2007. IFAAMAS.

[22] M. Winikoff, W. Liu, and J. Harland. Enhancing commitment machines. In *Proceedings of the 2nd International Workshop on Declarative Agent Languages and Technologies (DALT)*, volume 3476 of *LNAI*, pages 198–220, Berlin, 2005. Springer.

[23] P. Yolum and M. P. Singh. Commitment machines. In *Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL 2001)*, volume 2333 of *LNAI*, pages 235–247, Seattle, 2002. Springer.

[24] P. Yolum and M. P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 527–534, Bologna, July 2002. ACM Press.

[25] P. Yolum and M. P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence*, 42(1–3):227–253, Sept. 2004.

[26] P. Yolum and M. P. Singh. Enacting protocols by commitment concession. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 116–123, Honolulu, May 2007. IFAAMAS.