Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

A First-Order Formalization of Commitments and Goals for Planning

Felipe Meneguzzi
Pontifical Catholic University of
Rio Grande do Sul, Brazil
felipe.meneguzzi@pucrs.br

Pankaj R. Telang and **Munindar P. Singh**
North Carolina State University, USA
prtelang@ncsu.edu and *singh@ncsu.edu*

Abstract

Commitments help model interactions in multiagent systems in a computationally realizable yet high-level manner without compromising the autonomy and heterogeneity of the member agents. Recent work shows how to combine commitments with goals and apply planning methods to enable agents to determine their actions. However, previous approaches to modeling commitments are confined to propositional representations, which limits their applicability in practical cases. We propose a first-order representation and reasoning technique that accommodates templatic commitments and goals that may be applied repeatedly with differing bindings for domain objects. Doing so not only leads to a more perspicuous modeling, but also supports many practical patterns.

Introduction

We are concerned with the specification of interactions among agents in a multiagent system and the reasoning by the agents to enact such interactions. In the first regard, we adopt the notion of a *(social) commitment*, which describes an element of the social relationships between two agents in high-level terms. Thus a commitment in this paper is not to be confused with a "psychological" commitment expressing an agent's entrenchment with its intentions (Singh 1991; 2012; Castelfranchi 1995). In the second regard, we adopt the notion of an *(achievement) goal*, which describes a pro-attitude of an agent. Goals in our approach map to consistent desires and can be treated as possibly weaker than intentions, though the subtle distinctions between goals and intentions do not concern us. Existing approaches for commitments (Desai, Chopra, and Singh 2009; Fornara and Colombetti 2009; Verdicchio and Colombetti 2002; Winikoff 2007; Winikoff, Liu, and Harland 2005) as well as for goals and commitments (Chopra et al. 2010a; 2010b; Telang, Yorke-Smith, and Singh 2012) address several subtle situations.

Telang et al. (2013) show how to formalize commitments and goals via Hierarchical Task Networks (HTNs), providing a practical means to specify and verify the realizability of a multiagent system defined in terms of commitments. HTN planning is well suited for operationalizing commitments and goals because it helps capture the natural hierarchical structure of goals and commitments (Ghallab, Nau, and Traverso 2004). Existing HTN planning tools, such as JSHOP2 (Ilghami and Nau 2003), can accommodate real-life practical scenarios such as job scheduling. However, like existing approaches on commitments, such work is defined in a purely propositional framework. As a result, these approaches are cumbersome to apply in practical settings and do not naturally express certain common patterns of commitment reasoning.

Our contribution is twofold. First, we provide a first-order formalization of commitments and goals that enables an off-the-shelf HTN planner to be used in the verification of the realizability of a multiagent system. Second, we identify key limitations in the expressiveness of existing formalizations and reasoning mechanisms and develop reasoning patterns that address these limitations.

The contributions in this paper are motivated by the limitations of encoding the simple purchase scenario that follows. A customer sends a request for quotes for some specified goods to a merchant, who responds with a quote for the specified goods at a specified price. The customer either accepts or rejects it: if the customer accepts the quote, the merchant provides the goods; otherwise, upon receiving the goods, the customer pays the merchant the specified amount. When there are multiple merchants, goods, and prices, theoretical approaches, e.g., (Chopra and Singh 2006), do not handle them, though interestingly practical rule-based approaches, e.g., (Desai et al. 2005), can handle them.

Key Background

Commitments

Commitments are extensively studied in multiagent systems (Desai, Chopra, and Singh 2009; Fornara and Colombetti 2009; Verdicchio and Colombetti 2002). Specifically, a commitment C(DEBTOR, CREDITOR, antecedent, consequent) means that the DEBTOR agent commits to the CREDITOR agent to bring about the consequent if the antecedent holds (Singh 2008). Figure 1 summarizes a commitment lifecycle (Telang, Yorke-Smith, and Singh 2012). Upon creation, a commitment transitions from state *null* to *active*, which consists of two substates: *conditional* (its antecedent

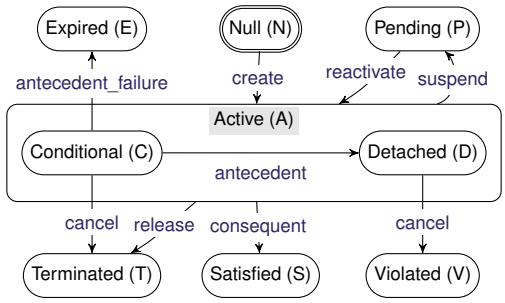

Figure 1: State transition diagram of commitment life cycle.

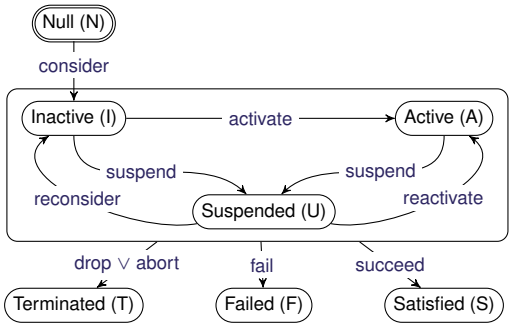

Figure 2: Goal lifecycle as a state transition diagram.

is false) and *detached* (its antecedent is true). An active commitment expires if its antecedent fails. If the consequent of an active commitment is brought about, the commitment is satisfied. An active commitment may be suspended and a pending commitment reactivated. If the debtor cancels or the creditor releases a conditional commitment, the commitment is *terminated*. If the debtor cancels a detached commitment, the commitment is *violated*.

Goals

A goal is a state of the world that an agent wishes to bring about. Formally, a goal $G = \mathsf{G}(x, pg, s, f)$, where x is an agent and pg is G's precondition, whose truth is required for G to be considered (Telang, Yorke-Smith, and Singh 2012; Thangarajah et al. 2011). Since, G's success condition is s and failure condition is f, G succeeds if $s \wedge \neg f$ holds.

Figure 2 shows our goal lifecycle (Telang, Yorke-Smith, and Singh 2012). A goal is in state *null* before it is considered, and if a goal's precondition is true, the agent may consider it, making it *inactive*. The agent may suspend a goal, activate an inactive goal, and reconsider or reactivate a suspended one. When the goal is inactive, active, or suspended: (a) if the agent drops or terminates the goal, the goal transitions to *terminated*; (b) if the failure condition holds, the goal transitions to *failed*; and (c) if the success condition holds, the goal transitions to *satisfied*.

Classical and HTN Planning

We adopt the usual logic language (Apt 1997) underpinning deterministic planning representations, namely, a first-order language with infinitely many variable symbols, no function symbols, and finitely many predicate and constant symbols (Ghallab, Nau, and Traverso 2004, Chapter 2). We omit specific details of the first-order logic used throughout the paper for brevity. Classical planning defines a problem in terms of an *initial state* and a *goal state*—each a set of ground atoms—and a set of *operators*. An operator is applicable when its *precondition* holds, and its execution brings about its *postcondition*. Planning is concerned with sequencing actions, obtained by instantiating operators, which describe concrete state transformations. We represent an *operator* o as a four-tuple $\langle name(o), pre(o), del(o), add(o)\rangle$, 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 $add(o)$ occur in $\vec{x}$; and (2) $pre(o), del(o)$ and $add(o)$ called, respectively, the precondition, *delete-list*, and *add-list*, are sets of atoms where $del(o) \cap add(o) = \emptyset$.

A Hierarchical Task Network (HTN) (Ghallab, Nau, and Traverso 2004) planner generates a plan by successive refinements of *tasks*. Tasks are either *primitive* (equivalent to operators in STRIPS planning) or *compound* (abstract high-level tasks). An HTN planner recursively decomposes compound tasks by applying a set 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 concrete tasks; they constrain the search space, helping improve efficiency. For example, a person may represent the goal of obtaining a certain product as an abstract task, which could be decomposed into the tasks of making an order, sending the payment, and then picking up the goods after shipping. Formally, an HTN *planning domain* $\mathcal{D}$ is a tuple $(\mathcal{A}, \mathcal{M})$, containing, 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. An operator corresponding to each primitive task t exists in the planning domain (that is, $\mathcal{A} \subseteq T$), and each compound task t must have one or more corresponding methods in the planning domain. A method m is a tuple (t, s, H'), where s is a precondition that must hold for a task t to be refined into another task network $H' = (T', C')$. Finally, 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*.

Technical Motivation

Clarifying how to deal with multiple instances is a useful contribution of this paper. An initial motivation for modeling interactions with multiple instances is the need to instantiate a small model of an interaction multiple times and to keep distinct instances from interfering with each other, yet relating to each other as appropriate. For example, a purchase protocol may be instantiated by multiple parties for multiple goods sold at multiple prices. This need can be addressed in a straightforward manner via a first-order representation. Nevertheless, even in such a representation, the overarching

challenge is to capture different ways in which interaction instances can (1) flexibly deviate from the initial specification; (2) split off into two or more instances that together accomplish the original interaction; (3) coalesce into larger interactions. In particular, the above should be accomplished in a modular manner, meaning that we should not have to rewrite an interaction specification but should be able to transform it in a systematic manner to produce the desired interaction. Yolum and Singh (2002) introduced the idea of digressions in protocols and Chopra and Singh (2006) introduced the notion of protocol transformers, though both papers adopted propositional frameworks. Specifically, these frameworks cannot encode domains containing the following patterns of behavior.

Piecemeal progress. The customer may pay the merchant in installments. The challenge to accommodate here is of arithmetic: we would like to handle the situation that, for example, a payment of $6 followed by a payment of $4 is equivalent to a payment of $10. (Note that domain regulations would determine whether payments may be split).

Concession. The merchant may balk at providing the goods (or goods above a certain value) in advance of any payment. Therefore, we might amend the protocol so that the customer makes a partial deposit first, upon which the merchant delivers the goods, upon which the customer makes the remaining payment. Unlike piecemeal progress, this scenario involves altering the structure of the commitments involved: the merchant is committing to providing the goods only upon receiving a deposit and the customer is committing to paying the remaining amount upon receiving the goods. Concession is loosely inspired by Yolum and Singh's (2007) approach, which deals with nesting commitments to reduce the apparent risk to each party in a protocol.

Consolidation. If a customer places two purchase orders in close succession, the merchant may ship both of the ordered goods in the same package. Likewise, the customer may pay for both orders via one check. This is a clear case of flexibility in enactment that multiagent protocols ought to support. To realize it requires a richer representation wherein some actions (e.g., delivery) may be associated with more than one protocol instance.

Compensation. The customer may return goods to the merchant and the merchant would issue a refund. The refund should match the goods returned. This should result from a straightforward application of the first-order representation. Additionally, the protocol should ordinarily ensure that for piecemeal payments, only the amount received may be refunded. Further, the protocol may build in some fraud-resistant measures, such as that a prior refund disables a subsequent refund or that the total amount refunded in successive protocol instances does not exceed some threshold.

Proposed Formal Framework

We now develop the logical rules, operators, and methods in the HTN formalism that operationalize the goal and commitment dynamics introduced above. Existing techniques show that it is straightforward to convert operational business process models into HTN (Pistore et al. 2005), as well as to convert business process languages into planning operators (Hoffmann, Weber, and Kraft 2010). Based on these, we assume that a large part of the domain-specific knowledge used in HTN encoding can be generated from the business processes being validated.

Table 1: Logical rules for commitment dynamics

$$null(C, Ct, \vec{Cv}) \leftarrow \neg var(C, Ct, \vec{Cv})$$
$$conditional(C, Ct, \vec{Cv}) \leftarrow active(C, Ct, \vec{Cv}) \wedge \neg p(C, Ct, \vec{Cv})$$
$$detached(C, Ct, \vec{Cv}) \leftarrow active(C, Ct, \vec{Cv}) \wedge p(C, Ct, \vec{Cv})$$
$$active(C, Ct, \vec{Cv}) \leftarrow \neg null(C, Ct, \vec{Cv}) \wedge \neg terminal(C, Ct, \vec{Cv}) \wedge \neg pending(C, Ct, \vec{Cv}) \wedge \neg satisfied(C, Ct, \vec{Cv})$$
$$terminated(C, Ct, \vec{Cv}) \leftarrow released(C, Ct, \vec{Cv}) \vee (\neg p(C, Ct, \vec{Cv}) \wedge cancelled(C, Ct, \vec{Cv}))$$
$$violated(C, Ct, \vec{Cv}) \leftarrow p(C, Ct, \vec{Cv}) \wedge cancelled(C, Ct, \vec{Cv})$$
$$satisfied(C, Ct, \vec{Cv}) \leftarrow \neg null(C, Ct, \vec{Cv}) \wedge \neg terminal(C, Ct, \vec{Cv}) \wedge q(C, Ct, \vec{Cv})$$
$$terminal(C, Ct) \leftarrow commitment(C, Ct, De, Cr) \wedge (cancelled(C, Ct, \vec{Cv}) \vee released(C, Ct, \vec{Cv}) \vee expired(C, Ct, \vec{Cv}))$$

Commitment Dynamics

A commitment is a tuple $\langle Ct, De, Cr, P, Q, \vec{Cv} \rangle$, where: Ct is the *commitment type*; De is the *debtor* of the commitment; Cr is the *creditor* of the commitment; P is the *antecedent*; Q is the *consequent*, both P and Q are existentially quantified first-order formulas; and, $\vec{Cv}$ is a list $[v_1, \ldots, v_n]$ of variables identifying specific instances of Ct. The first challenge in encoding commitments in a first-order setting is in ensuring that the components of a commitment are connected through their shared variables. In order to accomplish that, we model the entire set of variables of a particular commitment within one predicate. Thus, the number of variables n for a commitment is equivalent to the sum of *arities* of all first-order predicates in P, and Q, so if $P = p_{a0}(\vec{t_{a0}}) \ldots p_{ak}(\vec{t_{ak}})$ and $Q = p_{c0}(\vec{t_{c0}}) \ldots p_{ck}(\vec{t_{ck}})$, then $n = \sum_{i=a0}^{i=ck} |\vec{t_i}|$. Thus, for each commitment $C = \langle Ct, De, Cr, P, Q, \vec{Cv} \rangle$, where P is a formula φ and Q is a formula $\varkappa$ we define the rules below:

$$p(C, Ct, \vec{Cv}) \leftarrow commitment(C, Ct, De, Cr) \wedge \varphi$$
$$q(C, Ct, \vec{Cv}) \leftarrow commitment(C, Ct, De, Cr) \wedge \varkappa$$

Given these two basic formulas from the commitment tuple, we define rules that compute a commitment's state in Table 1, which follow from Figure 1. The *null* state for a commitment is "instance dependent", as each commitment has a number of possible instantiations, depending on the variables of the antecedent. In order to accomplish this, each commitment instance has an associated var predicate containing the commitment type and the list of variables associated to the instance. An *active* commitment is *conditional* if its antecedent (p) is false, and is *detached* otherwise. A commitment is *active* if it is not *null*, *terminal*, *pending*, or *satisfied*. Note that *terminal* is a shortcut for the states *can-*

Table 2: Planning operators for commitment dynamics.

$\langle \textbf{operator } !\textit{create}(C, Ct, De, Cr, \vec{C}v),$
$\quad \textbf{pre}(\textit{commitment}(C, Ct, De, Cr) \wedge \textit{null}(C, Ct, \vec{C}v)),$
$\quad \textbf{del}(), \textbf{add}(var(C, Ct, \vec{C}v))\rangle$
$\langle \textbf{operator } !\textit{suspend}(C, Ct, De, Cr, \vec{C}v),$
$\quad \textbf{pre}(\textit{commitment}(C, Ct, De, Cr) \wedge \textit{active}(C, Ct, \vec{C}v)),$
$\quad \textbf{del}(), \textbf{add}(\textit{pending}(C, Ct, \vec{C}v))\rangle$
$\langle \textbf{operator } !\textit{reactivate}(C, Ct, De, Cr, \vec{C}v),$
$\quad \textbf{pre}(\textit{commitment}(C, Ct, De, Cr) \wedge \textit{pending}(C, Ct, \vec{C}v)),$
$\quad \textbf{del}(\textit{pending}(C, Ct, \vec{C}v)), \textbf{add}()\rangle$
$\langle \textbf{operator } !expire(C, Ct, De, Cr, \vec{C}v),$
$\quad \textbf{pre}(\textit{commitment}(C, Ct, De, Cr) \wedge$
$\qquad \textit{conditional}(C, Ct, \vec{C}v) \wedge timeout(C, Ct, \vec{C}v)),$
$\quad \textbf{del}(), \textbf{add}(\textit{expired}(C, Ct, \vec{C}v))\rangle$
$\langle \textbf{operator } !cancel(C, Ct, De, Cr, \vec{C}v),$
$\quad \textbf{pre}(\textit{commitment}(C, Ct, De, Cr) \wedge active(C, Ct, \vec{C}v)),$
$\quad \textbf{del}(), \textbf{add}(\textit{cancelled}(C, Ct, \vec{C}v))\rangle$
$\langle \textbf{operator } !release(C, Ct, De, Cr, \vec{C}v),$
$\quad \textbf{pre}(\textit{commitment}(C, Ct, De, Cr) \wedge active(C, Ct, \vec{C}v)),$
$\quad \textbf{del}(), \textbf{add}(\textit{released}(C, Ct, \vec{C}v))\rangle$

celled, *released*, or *expired*. A commitment is *terminated* if it is *released* or it is *cancelled* when its antecedent is false. A commitment is *violated* if it is *cancelled* when its antecedent is true. A commitment is *satisfied* if it is not *null* and not *terminal*, and its consequent (q) is true.

Finally, we encode the transitions from Figure 1 as the planning operators in Table 2. For a commitment, the *create* operator adds the *var* predicate if the commitment is null. If a commitment is active, executing *suspend* adds the *pending* predicate. If a commitment is pending, executing *reactivate* deletes the *pending* predicate. If a commitment is *conditional* and a *timeout* has occurred, then executing *expire* adds the *expired* predicate. If a commitment is active, executing *cancel* adds the *cancelled* predicate. If a commitment is active, executing *release* adds the *released* predicate.

Goal Dynamics

We represent a goal as a tuple $\langle Gt, X, Pg, S, F, \vec{G}v\rangle$, where: Gt is the *goal type*; X is the *agent* that has the goal; Pg is the goal *precondition*; S is the *success condition*; F is the *failure condition*; and $\vec{G}v$ is a list of variables identifying specific instances of Gt. Similarly to commitments, the number of variables for a commitment will be equivalent to the sum of *arities* of all first-order predicates in Pg, S and F. Likewise, for each goal $G = \langle Gt, X, Pg, S, F, Gv\rangle$, where Pg is a formula ϖ, S is a formula ς, and F is a formula ϑ we define the following rules:

$$pg(G, Gt, \vec{G}v) \leftarrow goal(G, Gt, X) \wedge \varpi$$
$$s(G, Gt, \vec{G}v) \leftarrow goal(G, Gt, X) \wedge \varsigma$$
$$f(G, Gt, \vec{G}v) \leftarrow goal(G, Gt, X) \wedge \vartheta$$

Table 3 defines rules that compute a goal's state following Figure 2. Finally, Table 4 encodes the goal state transitions from Figure 2 as planning operators. We omit their details for brevity.

Table 3: Logical rules for goal dynamics.

$null(G, Gt, \vec{G}v) \leftarrow \neg var(G, Gt, \vec{G}v)$
$inactiveG(G, Gt, \vec{G}v) \leftarrow \neg null(G, Gt, \vec{G}v)$
$\quad \wedge \neg f(G, Gt, \vec{G}v) \wedge \neg s(G, Gt, \vec{G}v)$
$\quad \wedge \neg terminalG(G, Gt, \vec{G}v) \wedge \neg suspendedG(G, Gt, \vec{G}v)$
$\quad \wedge \neg activeG(G, Gt, \vec{G}v)$
$activeG(G, Gt, \vec{G}v) \leftarrow activatedG(G, Gt, \vec{G}v)$
$\quad \wedge \neg f(G, Gt, \vec{G}v) \wedge \neg satisfiedG(G, Gt, \vec{G}v)$
$\quad \wedge \neg terminalG(G, Gt, \vec{G}v) \wedge \neg suspendedG(G, Gt, \vec{G}v)$
$satisfiedG(G, Gt, \vec{G}v) \leftarrow \neg null(G, Gt, \vec{G}v)$
$\quad \wedge \neg terminal(G, Gt, \vec{G}v) \wedge pg(G, Gt, \vec{G}v)$
$\quad \wedge s(G, Gt, \vec{G}v) \wedge \neg f(G, Gt, \vec{G}v)$
$failedG(G, Gt, \vec{G}v) \leftarrow \neg null(G, Gt, \vec{G}v) \wedge f(G, Gt, \vec{G}v)$
$terminatedG(G, Gt, \vec{G}v) \leftarrow \neg null(G, Gt, \vec{G}v)$
$\quad \wedge (dropped(G, Gt, \vec{G}v) \vee aborted(G, Gt, \vec{G}v))$
$terminalG(G, Gt, \vec{G}v) \leftarrow goal(G, Gt, X)$
$\quad \wedge (dropped(G, Gt, \vec{G}v) \vee aborted(G, Gt, \vec{G}v)$

Formalizing the Patterns

This section applies our approach to capture the patterns from the Technical Motivation section. Table 5 shows the goals and commitments of a customer and a merchant. For example, C_1 is the customer's commitment to the merchant to paying if the merchant provides the goods. In C_1, 123 is the transaction identifier, and \$100 is the payment amount.

Table 6 shows the methods that we employ in formalizing the patterns. For brevity, we only present a subset of the methods and operators. The satisfy(C) method encodes the plans for satisfying a commitment C. If C is of type CT_1 and is detached, then satisfy either invokes the pay method once representing that the customer pays the entire amount to the merchant, or invokes the pay method twice representing that the customer pays the merchant in two installments. If C is of type CT_2 and is detached, then satisfy invokes the $goods$ method representing that the merchant provides the goods to the customer. If C is of type CT_3 and is detached, then satisfy invokes refundpaid representing that the merchant refunds the customer. The satisfy(C_1, C_2) method invokes paytogether if commitments C_1 and C_2 are detached, and paytogether invokes pay for C_1 and C_2. The pay method implements the arithmetic to add up the payments for a transaction identifier. If the customer has paid an installment, then the pay method invokes the updatepaid operator, which deletes the previous paid predicate, and adds a paid predicate with the new amount. Otherwise, the pay method invokes the paid operator, which adds a paid predicate.

Note that our rules and operators from Tables 2–4 are completely general, whereas the methods and operators from Table 6 are specific to the patterns we present.

Piecemeal progress. Figure 3 shows an HTN decomposition tree for piecemeal progress. The customer creates C_1 to achieve its goal G_1 (for clarity of presentation, we omit the goal operations $consider$ and $activate$ in the HTN decomposition trees). The merchant detaches C_1 by sending the goods. This presumes that merchant has a goal to get paid. To satisfy C_1, the customer needs to pay \$100, which the customer may pay either as a lump sum, or in two install-

Table 4: Planning operators for goal dynamics.

⟨**operator** !*consider*$(G, Gt, X, \vec{Gv})$,
pre$(goal(G, Gt, X) \wedge$ *null*$(G, Gt, \vec{Gv}) \wedge pg(G, Gt, \vec{Gv}))$,
del$(),$ **add**$($*var*$(G, Gt, \vec{Gv}))\rangle$
⟨**operator** !*activate*$(G, Gt, X, \vec{Gv})$,
pre$(goal(G, Gt, X) \wedge$ *inactiveG*$(G, Gt, \vec{Gv}))$,
del$(),$ **add**$($*activatedG*$(G, Gt, \vec{Gv}))\rangle$
⟨**operator** !*suspend*$(G, Gt, X, \vec{Gv})$,
pre$(goal(G, Gt, X) \wedge \neg$*terminalG*$(G, Gt, \vec{Gv}) \wedge$
$\neg$*null*$(G, Gt, \vec{Gv}))$,
del$($*activatedG*$(G, Gt, \vec{Gv}))$, **add**$($*suspendedG*$(G, Gt, \vec{Gv}))\rangle$
⟨**operator** !*reconsider*$(G, Gt, X, \vec{Gv})$,
pre$(goal(G, Gt, X) \wedge$ *suspendedG*$(G, Gt, \vec{Gv}) \wedge$
$\neg$*terminalG*$(G, Gt, \vec{Gv}) \wedge \neg$*null*$(G, Gt, \vec{Gv}))$,
del$(),$ **add**$($*suspendedG*$(G, Gt, \vec{Gv}))\rangle$
⟨**operator** !*reactivate*$(G, Gt, X, \vec{Gv})$,
pre$(goal(G, Gt, X) \wedge$ *suspendedG*$(G, Gt, \vec{Gv}) \wedge$
$\neg$*terminalG*$(G, Gt, \vec{Gv}) \wedge \neg$*null*$(G, Gt, \vec{Gv}))$,
del$($*activatedG*$(G, Gt, \vec{Gv}))$, **add**$($*suspendedG*$(G, Gt, \vec{Gv}))\rangle$
⟨**operator** !$drop(G, Gt, X, \vec{Gv})$,
pre$(goal(G, Gt, X) \wedge \neg$*terminalG*$(G, Gt, \vec{Gv}) \wedge$
$\neg$*null*$(G, Gt, \vec{Gv}))$,
del$(),$ **add**$($*dropped*$(G, Gt, \vec{Gv}))\rangle$
⟨**operator** !$abort(G, Gt, X, \vec{Gv})$,
pre$(goal(G, Gt, X) \wedge \neg$*terminalG*$(G, Gt, \vec{Gv}) \wedge$
$\neg$*null*$(G, Gt, \vec{Gv}))$,
del$(),$ **add**$($*aborted*$(G, Gt, \vec{Gv}))\rangle$

Table 5: Goals and commitments for the patterns.

Id	Type	Goal or commitment
G_1	GT_1	G(CUST,needsGoods(123),goods(123),deadline(123))
G_2	GT_1	G(CUST,needsGoods(456),goods(456),deadline(456))
C_1	CT_1	C(CUST, MER, goods(123), paid($100, 123))
C_2	CT_2	C(MER, CUST, pay($20, 123), goods(123))
C_3	CT_1	C(CUST, MER, goods(123), pay($80, 123))
C_4	CT_1	C(CUST, MER, goods(456), pay($200, 456))
C_5	CT_3	C(MER, CUST, return(123), refundpaid(123))

ments. Figure. 3 shows a plan in which the customer pays two installments of $50 each.

Concession. This pattern involves two commitments: the merchant commits (C_2) to providing the goods upon receiving a deposit of $20, and the customer commits (C_3) to the merchant to pay the remaining amount of $80 upon receiving the goods. Figure 4 shows an HTN decomposition tree for concession. The customer and the merchant create C_2 and C_3, respectively. Then the customer detaches C_2 by paying $20. The merchant satisfies C_2 by providing the goods, which also detaches C_3. Next, the customer pays $80 to satisfy C_3. The detach method has a structure similar to the satisfy method.

Consolidation. In this pattern, the customer has a second goal G_2 for goods(456) and C_4 is the commitment from the customer to the merchant to paying $200 if the merchant provides the goods. Figure 5 illustrates the HTN decomposition tree for consolidation, which shows that to achieve its goals G_1 and G_2, the customer creates commitments

Table 6: Methods and operators for the patterns.

⟨**method**(satisfy(C)),
pre$(commitment(C, CT_1, cust, mer) \wedge var(C, CT_1,$
$(cAmount, tID)) \wedge detached(C, CT_1, (cAmount, tID))$,
tn$(pay(cust, mer, cAmount, tID))$,
pre$(commitment(C, CT_1, cust, mer) \wedge var(C, CT_1,$
$(cAmount, tID)) \wedge detached(C, CT_1, (cAmount, tID)))$,
tn$(pay(cust, mer, cAmount/2, tID) \wedge pay(cust, mer,$
$cAmount/2 + cAmount\%2, tID))$
pre$(commitment(C, CT_2, mer, cust) \wedge var(C, CT_2,$
$(cAmount, tID)) \wedge detached(C, CT_2, (cAmount, tID)))$,
tn$(goods(mer, cust, tID))$
pre$(commitment(C, CT_3, mer, cust) \wedge var(C, CT_3,$
$(tID)) \wedge detached(C, CT_2, (tID)))$,
tn(refundpaid$(mer, cust, tID))\rangle$
⟨**method**(satisfy(C_1, C_2)),
pre$(commitment(C_1, CT_1, cust, mer) \wedge var(C_1, CT_1,$
$(c1Amount, t1ID)) \wedge detached(C_2, CT_1, (c1Amount,$
$t1ID)) \wedge commitment(C_2, CT_1, cust, mer) \wedge var(C_2,$
$CT_1, (c2Amount, t2ID)) \wedge detached(C_2, CT_1,$
$(c2Amount, t2ID)))$
tn$(paytogether(cust, mer, amount, t1ID, t2ID))\rangle$
⟨**method**$(paytogether(cust, mer, amount, t1ID, t2ID))$,
pre$(commitment(C_1, CT_1, cust, mer)$
$\wedge var(C_1, CT_1, (c1Amount, t1ID))$
$\wedge detached(C_2, CT_1, (c1Amount, t1ID))$
$\wedge commitment(C_2, CT_1, cust, mer)$
$\wedge var(C_2, CT_1, (c2Amount, t2ID))$
$\wedge detached(C_2, CT_1, (c2Amount, t2ID))$
$\wedge(amount = c1Amount + c2Amount))$
tn$(pay(cust, mer, c1Amount, t1ID)$
$\wedge pay(cust, mer, (amount - c1Amount), t2ID))$
⟨**method**$(pay(cust, mer, amount, tID))$,
pre$(commitment(C, Ct, cust, mer) \wedge var(C, Ct,$
$(cAmount, tID)) \wedge paid(cust, mer, oldAmt, tID))$,
tn$(!updatepaid(cust, mer, (oldAmt + amount), tID))$
pre$(commitment(C, Ct, cust, mer)$
$\wedge var(C, Ct, (cAmount, tID))$,
tn(!paid$(cust, mer, amount, tID))\rangle$
⟨**operator** !$paid(cust, mer, amount, tID)$,
pre$(agent(cust) \wedge agent(mer))$,
del$(),$ **add**$(paid(cust, mer, amount, tID))\rangle$
⟨**operator** !$updatepaid(cust, mer, amount, tID)$,
pre$(agent(cust) \wedge agent(mer)$
$\wedge paid(cust, mer, oldAmount, tID))$,
del$(paid(cust, mer, oldAmount, tID))$,
add$(paid(cust, mer, amount, tID))\rangle$

C_1 and C_4. The merchant detaches C_1 and C_4 by shipping the goods (goods(123) and goods(456)) together using the $shiptogether$ method. The customer satisfies C_1 and C_4 by making a consolidated payment of $300 to the merchant. Table 6 shows the details of the paytogether method, which splits the $300 into $100 and $200, and applies them to the transactions 123 and 456.

Compensation. In this pattern, the merchant commits (C_5) to the customer to refunding the amount paid by the customer if the customer returns the goods. Figure 6 illustrates the HTN decomposition tree for compensation. The customer and the merchant create C_1 and C_5, respectively. The merchant detaches C_1 by providing the goods, and the customer makes a partial payment of $50 to the merchant. Next, the customer returns the goods, which detaches commitment

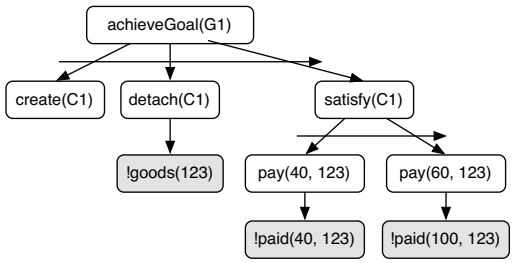

Figure 3: Decomposition tree for the piecemeal pattern.

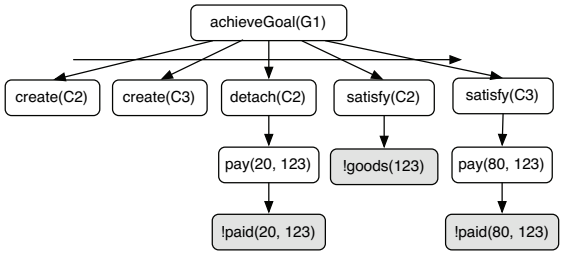

Figure 4: Decomposition tree for the concession pattern.

C_5. The merchant satisfies C_5 by refunding \$50 to the customer. Notice that the refundpaid method identifies and refunds the actual amount paid by the customer.

Discussion

HTN planning provides a promising approach to provide depth of representation and reasoning to conceptualizing the connection between commitments and goals. We motivate some key patterns of reasoning about and enacting commitments and show how our approach naturally accommodates them. We articulate a first-order approach to represent commitments and the generation of plans that lead them to suitable states (satisfied in a happy path). Such a formalization not only affords greater expressivity than currently existing approaches, but also allows one to employ an off-the-shelf HTN planner as a validation tool for a business process. Moreover, since most agent programming languages build agent plan libraries using an HTN-like abstraction, our formalization also creates the possibility of directly implementing business process in agent systems.

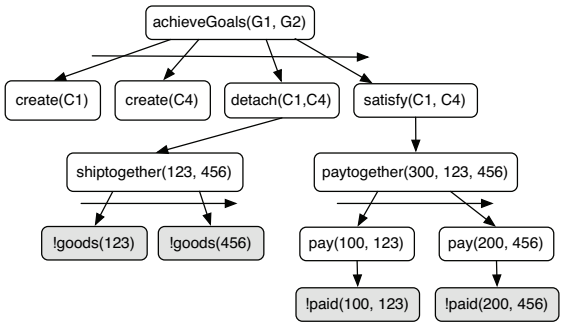

Figure 5: Decomposition tree for the consolidation pattern.

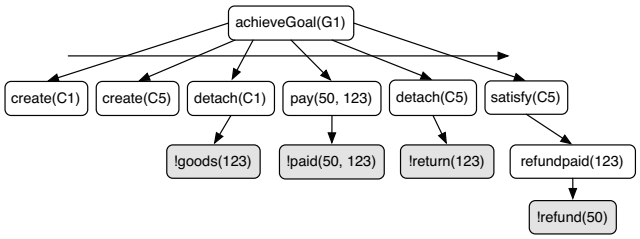

Figure 6: Decomposition tree for the compensation pattern.

A network of commitments among agents specifies the interactive components of a business model. The joint plans generated from such a model via HTN planning describe alternative means to enact the model. These alternatives involve different costs and benefits for the participating agents and even a simple element of a business model can lead to a variety of interactions. Our first-order approach provides a practically viable way to generate such diverse interactions.

We have implemented a conversion tool that automatically generates the HTN domain from a set of goals and commitments and have verified, using JSHOP2 (Ilghami and Nau 2003), that the examples shown in this paper are valid. Our conversion tool is limited in that the domain-specific knowledge still needs to be encoded by hand, but we expect to be able to leverage existing work on planning within business processes to overcome this limitation (Pistore et al. 2005; Hoffmann, Weber, and Kraft 2010).

Future Directions

This approach open some interesting directions for future work. Key among these is grounding the notion of protocol transformations in HTN planning. Specifically, we envisage applying HTN planning not only to operationalize business models but to affect the three aspects of a commitment-based service engagement (Singh, Chopra, and Desai 2009).

First, HTN planning applies at the level of achieving the antecedents and consequents of the relevant commitments to address the idea of *digression* (Yolum and Singh 2002) in commitment protocols: a planner could identify joint plans that support different enactments that result in a commitment being satisfied. Second, HTN planning can apply to the structure of interrelated commitments and thus alter the business model itself. Concession, described above, is an example of a business transactional (Singh, Chopra, and Desai 2009) change. Similar transformations could be identified not only at the business transaction level but also at the organizational structure level (e.g., involving delegation) and the organizational context level. In future work, we plan to remove goal and plan identifier variables (C and G) from our formalization, since they interfere with some of the logical deductions we aim to implement (Chopra and Singh 2011).

References

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

Castelfranchi, C. 1995. Commitments: From individual intentions to groups and organizations. In *Proceedings of the International Conference on Multiagent Systems*, 41–48.

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

Chopra, A. K., and Singh, M. P. 2011. Specifying and applying commitment-based business patterns. In *Proceedings of the 10th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 475–482. Taipei: IFAAMAS.

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

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

Desai, N.; Mallya, A. U.; Chopra, A. K.; and Singh, M. P. 2005. Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering* 31(12):1015–1027.

Desai, N.; Chopra, A. K.; and Singh, M. P. 2009. 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.

Fornara, N., and Colombetti, M. 2009. 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)*, 101–118.

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

Hoffmann, J.; Weber, I.; and Kraft, F. M. 2010. SAP speaks PDDL. In Fox, M., and Poole, D., eds., *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*. Atlanta: AAAI Press.

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

Pistore, M.; Marconi, A.; Bertoli, P.; and Traverso, P. 2005. Automated composition of web services by planning at the knowledge level. In Kaelbling, L. P., and Saffiotti, A., eds., *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, 1252–1259.

Singh, M. P.; Chopra, A. K.; and Desai, N. 2009. Commitment-based service-oriented architecture. *IEEE Computer* 42(11):72–79.

Singh, M. P. 1991. Social and psychological commitments in multiagent systems. In *AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels*, 104–106.

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

Singh, M. P. 2012. Commitments in multiagent systems: Some history, some confusions, some controversies, some prospects. In Paglieri, F.; Tummolini, L.; Falcone, R.; and Miceli, M., eds., *The Goals of Cognition: Essays in Honor of Cristiano Castelfranchi*. London: College Publications. chapter 31, 601–626.

Telang, P. R.; Meneguzzi, F.; and Singh, M. P. 2013. Hierarchical planning about goals and commitments. In *Proceedings of the 12th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. St. Paul, Minnesota: IFAAMAS.

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

Thangarajah, J.; Harland, J.; Morley, D.; and Yorke-Smith, N. 2011. 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*, 1–21. Springer.

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

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

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

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

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