Specifying and Applying Commitment-Based Business Patterns

Amit K. Chopra
University of Trento
Via Sommarive, 14 I-38123 Povo, Italy
chopra@disi.unitn.it

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

ABSTRACT

Recent work in communications and business modeling emphasizes a commitment-based view of interaction. By abstracting away from implementation-level details, commitments can potentially enhance perspicuity during modeling and flexibility during enactment.

We address the problem of creating commitment-based specifications that directly capture business requirements, yet apply in distributed settings. We encode important business patterns in terms of commitments and group them into *methods* to better capture business requirements.

Our approach yields significant advantages over existing approaches: our patterns (1) respect agent autonomy; (2) capture business intuitions faithfully; and (3) can be enacted in real-life, distributed settings. We evaluate our contributions using the Extended Contract Net Protocol.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Design, Theory

Keywords

Protocols, Software engineering, Method engineering

1. INTRODUCTION

Commitment-based approaches to agent communication are finding broad traction in specifying interaction protocols. What makes commitments an appealing abstraction is that they naturally capture the business relationships that arise in our everyday life and business interactions, and offer flexibility in realizing them.

The expression C(debtor, creditor, antecedent, consequent) represents a commitment: it means that the debtor is committed to the creditor for ensuring the consequent if the antecedent holds. For example, C(buyer, seller, goods, paid) means that the buyer commits to the seller that if the seller provides the goods the buyer will ensure he is paid. Whereas it is easy enough to come up with commitments, it is not easy to specify the *right* commitments for particular applications. For instance, Desai et al. [4] show how a scenario dealing with foreign exchange transactions may be formalized in multiple ways using commitments, each with different ramifications on the outcomes. This leads us to the main question we address: *How can we guide software engineers in creating appropriate commitment-based specifications?*

Such guidance is often available for operational approaches such as state machines and Petri nets that describe interactions in terms of message order and occurrence. For instance, Figure 1 shows two common patterns expressed as (partial) state machines, which can aid software engineers in specifying operational interactions. Here, b and s are buyer and seller, respectively. (A) says that the seller may accept or reject an order; (B) says the buyer may confirm an order after the seller accepts it.

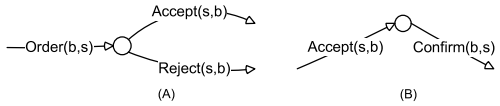

Figure 1: Example operational patterns.

By contrast, commitment protocols abstract away from operational details, focusing on the meanings of messages, not their flow. Clearly, operational patterns such as the above would not apply to the design of commitment protocols. What kinds of patterns would help in the design of commitment protocols? By and large, they would need to be business patterns—characterizing requirements, not operations—that emphasize meanings in terms of commitments. In contrast with Figure 1, business patterns—as we formalize them—describe what it *means* to make, accept, reject, or update an offer, not when to send messages.

We apply our patterns towards creating commitment-based specifications in a manner inspired by situational method engineering (SME) [10]. In SME, a method corresponds to a particular software engineering lifecycle and is composed of reusable fragments selected based on application and organizational requirements. For example, based on its requirements, a development organization may adopt goal-based or scenario-based requirements engineering or omit requirements engineering altogether. Analogously, for us, those developing commitment-based specifications would choose a commitment-based method that composes specific business patterns and that suits their requirements, including those relating to the organizational context [12] in which the sys-

Cite as: Specifying and Applying Commitment-Based Business Patterns, Amit K. Chopra and Munindar P. Singh, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

tem to be will be enacted. In this sense, a method describes a second-order business pattern.

Contributions. Our contributions are as follows. First, we identify business patterns as distinct from semantic and enactment patterns. Whereas semantic patterns encapsulate general commitment reasoning [2] and enactment patterns guide a commitment-based agent design, business patterns support specifying business protocols in cross-organizational settings. Second, we identify semantic antipatterns, which generally reflect a closed system way of thinking and are not suitable for open settings. Third, we identify several business patterns that accommodate common business situations. Fourth, we formulate engineering *methods* as sets of selected patterns and outline a simple approach based on organizational requirements for selecting among methods.

Like any set of patterns, the patterns in this paper reflect intuitions rooted in experience. Our patterns, however, are also motivated by the following requirements.

Autonomy-compatibility Autonomy broadly refers to the lack of control: no agent has control over another agent. To get things done, agents set up the appropriate commitments by interacting. Any expectation from an agent beyond what the agent has explicitly committed to is unreasonable.

Explicit meanings Our patterns make public the aspects of meaning that ought to have been public in the first place, but are often hidden within agent implementations. For example, updating a standing offer would mean replacing an existing commitment with a new one. An operational approach would simply allow for multiple *UpdateOffer* messages. If agents differently assume whether the latest message prevails, misalignment would ensue.

Distributed enactment Our business patterns build up systematically from a core set of communication primitives reflecting established ways to manipulate commitments in distributed settings.

Result. We evaluate our approach via a case study. The main result we obtain is that our approach highlights the critical design decisions and places them at a business level. First, a designer can see what is at stake in those decisions and can choose according to the needs of the business partners and the contextual setting in which they will interact. Second, through its focus on formalizing business meaning, our approach captures exactly what the business needs. In contrast, traditional approaches are guilty of over-specifying on some aspects (leading to rigid enactments) and under-specifying others (leading to potential ambiguity in realistic environments). Their only recourse against the former is to enumerate additional enactments and their only recourse against the latter is to insert additional ad hoc constraints, thus leaning toward over-specification. The overall outcome is excessive complexity.

Organization. The rest of the paper is organized as follows. Section 2 describes the necessary background for computing commitments in distributed settings. It also discusses semantic patterns. Section 3 introduces some business patterns, enactment patterns, and semantic antipatterns. Section 4 applies the patterns toward protocol specification via methods. Section 5 applies our approach to the Extended Contract Net Protocol [15]. Section 6 sums up our approach along with a discussion of the relevant literature.

2. BACKGROUND

We adopt Chopra and Singh's formal framework [2], including their language and reasoning postulates. Table 1 repeats their grammar for commitments and for messages that manipulate commitments. A sender can inform a receiver using a Declare. A commitment is detached when its antecedent becomes true ($\top$), meaning its debtor is unconditionally committed. A commitment is discharged when its consequent becomes true. Table 2 lists some important kinds of commitments that may arise in a fan-selling scenario.

Table 1: Syntax for commitments and messages.

Commitment $\longrightarrow$ C(Agent, Agent, DNF, CNF)
Content $\longrightarrow$ Atom | $\neg$ Atom | Stative(Agent, Agent, DNF, CNF)
Stative $\longrightarrow$ created | released | canceled | violated
DNF $\longrightarrow$ And | And $\vee$ DNF
CNF $\longrightarrow$ Or | Or $\wedge$ CNF
And $\longrightarrow$ Content | Content $\wedge$ And
Or $\longrightarrow$ Content | Content $\vee$ Or
Message $\longrightarrow$ Declare(Agent, Agent, News)
Message $\longrightarrow$ Op(Agent, Agent, DNF, CNF)
News $\longrightarrow$ Atom | Stative(Agent, Agent, DNF, CNF) | Atom $\wedge$ News | Stative(Agent, Agent, DNF, CNF) $\wedge$ News
Op $\longrightarrow$ Create | Cancel | Release | Delegate

Table 2: Commitments in the syntax of Table 1.

Name	**Commitment** (S is seller; B is buyer)
c_A	C(S, B, paid, fan): S commits to B that if payment is made, the fan will be delivered.
c_{UA}	C(S, B, $\top$, fan): The unconditional version of c_A. S commits to B that the fan will be delivered.
c_B	C(S, B, released(S, B, $\top$, fan), created(S, B, $\top$, discount)): S commits to B that if B releases S from the commitment to deliver the fan, S will give B a discount on its next purchase.
c_C	C(S, B, $\neg$fan$\wedge$released(S, B, $\top$, fan), created(S, B, $\top$, discount)): Similar to c_B except that it accounts for the case when S's delivery of the fan and B's release cross in transit—in such a case, S need not give the discount anymore.

The *statives* (except violated) record the history of commitment operations. For example, created(x, y, r, u) is added to an agent's KB when an agent has observed the message Create(x, y, r, u), and so on. We introduce violated to capture that an unconditional commitment has been violated, e.g., because the deadline for bringing about its consequent has passed. Table 3 lists each message along with its sender and receiver, and the effects of the messages (we omit the assignment operation for brevity).

Chopra and Singh's framework uses two kinds of postulates: update postulates (appropriately constrained by the

Table 3: Core messages pertaining to commitments.

Message	Sender	Receiver	Effect
$\mathsf{Create}(x, y, r, u)$	x	y	$\mathsf{C}(x, y, r, u)$
$\mathsf{Cancel}(x, y, r, u)$	x	y	$\neg\mathsf{C}(x, y, r, u)$
$\mathsf{Release}(x, y, r, u)$	y	x	$\neg\mathsf{C}(x, y, r, u)$
$\mathsf{Delegate}(x, y, z, r, u)$	x	z	$\mathsf{C}(z, y, r, u)$
$\mathsf{Declare}(x, y, p)$	x	y	p

conditions listed below) that capture the computation of an agent's state following its observation of a message, and commitment reasoning postulates such as (assuming the same debtor-creditor pair throughout a postulate) $u \rightarrow \neg\mathsf{C}(r, u)$ (captures discharge) and $\mathsf{C}(r \wedge s, u) \wedge s \rightarrow \mathsf{C}(r, u)$ (captures detach), and so on. These postulates encode *semantic patterns*, that is, the domain-independent rules of computing commitments. In Table 3, the effects are *nominal* because they hold only under the following conditions. (A commitment $\mathsf{C}(r, u)$ is stronger than $\mathsf{C}(r', u')$ iff $u \vdash u'$ and $v' \vdash v$.)

Novel Creation $\mathsf{Create}(r, u)$ is a noop if a stronger commitment $\mathsf{C}(s, v)$ holds or has held before (that is, if $\mathsf{created}(\mathsf{s}, \mathsf{v})$ holds).

Complete Erasure $\mathsf{Release}(r, u)$ or $\mathsf{Cancel}(r, u)$ removes all commitments weaker than $\mathsf{C}(r, u)$ provided no $\mathsf{C}(s, v)$ strictly stronger than $\mathsf{C}(r, u)$ holds; otherwise it is a noop.

Accommodation From $\mathsf{Release}(r, u)$ and $\mathsf{Cancel}(r, u)$, infer that each weaker $\mathsf{C}(s, v)$ has held before.

Notification Whenever a creditor learns of a condition that features in the antecedent, it notifies the debtor, and whenever a debtor learns of a condition that features in the consequent, it notifies the creditor.

Priority If two agents may take conflicting actions, the protocol specifies ahead of time whose action has priority.

The principal result that follows from the above conditions is that even when agents communicate asynchronously, they would remain aligned with respect to their commitments (assuming reliable in-order message delivery for every pair of agents—easily supported by common infrastructure such as reliable message queues).

3. COMMITMENT PATTERNS

We discuss three kinds of patterns. *Business patterns* capture the meanings of business communications in terms of commitments, *enactment patterns* specify *when* an agent may enact a particular business communication, and *semantic antipatterns* capture inappropriate patterns. All of our examples are from the fan-selling domain (Table 2).

3.1 Business Patterns

Business patterns encode the common ways in which businesses engage each other. By representing business patterns using Chopra and Singh's framework, we can guarantee alignment even for asynchronous enactments.

The messages of Table 3 correspond to elementary business patterns. Here, $\mathsf{Offer}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u})$ means $\mathsf{Create}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u})$ (the Generic Offer or GO pattern); $\mathsf{CancelOffer}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u})$ means $\mathsf{Cancel}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u})$ (the Cancel Offer or CO pattern); and $\mathsf{RejectOffer}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u})$ means $\mathsf{Release}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u})$ (the Release Offer or RO pattern). However, we can build upon the basic primitives to build more complex business patterns such as for updating, compensation, mutual commitment, and so on. Below, we list some recurring business patterns.

• Basic Offer (BO)
Intent To set up a basic business transaction.
Motivation Captures a basic way of doing business.
Implementation $\mathsf{BasicOffer}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u})$ means $\mathsf{Create}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u})$ where r and u are formulas over atoms (they contain no statives).
Example $\mathsf{BasicOffer}(\mathsf{S}, \mathsf{B}, \mathsf{paid}, \mathsf{fan})$
Consequences For progress, the creditor should be ready to bring about the antecedent.

• Nested Offer (NO)
Intent The debtor wants a commitment from the creditor for something in return for something else.
Motivation To set up a richer (both parties are committed) and more flexible engagement.
Implementation $\mathsf{NestedOffer}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u})$ means $\mathsf{Create}(\mathsf{x}, \mathsf{y}, \mathsf{created}(\mathsf{y}, \mathsf{x}, \top, \mathsf{r}), \mathsf{u})$.
Example $\mathsf{NestedOffer}(\mathsf{S}, \mathsf{B}, \mathsf{paid}, \mathsf{fan})$
Consequences When the antecedent holds, both x and y are unconditionally committed to u and r, respectively. When that happens, each would gain some measure of safety in acting first and discharging its commitment, thus improving flexibility in enactment.

• Mutual Commitment Offer (MCO)
Intent Debtor should have the exact "reciprocal" commitment from the creditor: if the creditor commits to u for r, the debtor commits to r for u.
Motivation To set up a richer and more flexible engagement, wherein both parties are committed.
Implementation $\mathsf{MutualCommitmentOffer}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u})$ means $\mathsf{Create}(\mathsf{x}, \mathsf{y}, \mathsf{created}(\mathsf{y}, \mathsf{x}, \mathsf{u}, \mathsf{r}), \mathsf{created}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u}))$
Example $\mathsf{MutualCommitmentOffer}(\mathsf{S}, \mathsf{B}, \mathsf{paid}, \mathsf{fan})$
Consequences This pattern is less prone to violations than Nested Offer, as only one party could possibly violate its commitment.

• Business Transaction Identifiers (BTI)
Intent To enable an agent to distinguish distinct offers and to relate commitments that coherently fall into the same business transaction.
Motivation It is important (1) not to conflate distinct business transactions, so that commitments from different transactions do not interfere with each other, and (2) preserve logical structure so the reasoning is sound.
Implementation Introduce identifiers in the antecedent, propagating them as needed to the consequent.
Example Writing the identifier as the first parameter of a proposition, $\mathsf{C}(\mathsf{S}, \mathsf{B}, \mathsf{paid}(0), \mathsf{fan}(0))$ occurs in a different transaction from $\mathsf{C}(\mathsf{S}, \mathsf{B}, \mathsf{paid}(1), \mathsf{fan}(1))$.
Consequences We need a clear information model to make sure the commitments pertaining to one transaction do not involve the identifiers of another.

• Compensation (COM)
Intent To compensate the creditor in case of commitment cancellation or violation by the debtor.
Motivation It is not known in advance whether a party will fulfill its commitments; compensation commitments provides some assurance to the creditor in case of violations.
Implementation $\mathsf{Compensate}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u}, \mathsf{p})$ means $\mathsf{Create}(\mathsf{x}, \mathsf{y}, \mathsf{violated}(\mathsf{x}, \mathsf{y}, \mathsf{r}, \mathsf{u}), \mathsf{p})$.

Example Compensate(S, B, paid, fan, discount)

Consequences A commitment (even a compensation commitment) should ideally be supported by compensation; however, at some level, the only recourse is escalation to the surrounding business *context*—for example, the local jurisdiction [12].

• Update (UP)

Intent To update a previously made offer.

Motivation Changing business environments may require debtors to update their commitments.

Implementation Update(x, y, r, u, s, v) means Cancel(x, y, r, u) and Create(x, y, s, v).

Example Update(S, B, paid$12, fan, paid$15, fan)

Consequences One must be careful in applying updates since the creditor may not find the new commitment an acceptable substitute for the old commitment.

• Release Incentive (RI)

Intent To enable the debtor to offer an incentive to the creditor for releasing it from a commitment.

Motivation Due to changing business environments, it may be more profitable for the debtor to offer an incentive to the creditor for releasing it from an existing commitment.

Implementation ReleaseIncentive(x, y, r, u, p) means Create(x, y, $\neg$u $\wedge$ released(x, y, r, u), p) where p represents the incentive. The conjunction in the antecedent is necessary to handle the case where Declare(x, y, u) may cross with Release(x, y, r, u): once u occurs, the debtor is off the hook.

Example ReleaseIncentive(S, B, $\top$, fan, discount)

Consequences The creditor may not take up the incentive offer; the debtor may then consider canceling the commitment unilaterally.

• Delegation Acceptance (DA)

Intent To set up the proper relationship between a delegator and delegatee for effective delegations.

Motivation The debtor may delegate (viewed as a request) a commitment to another party if it sees value in it; however, the delegatee is not bound to accept the delegation.

Implementation DelegationAcceptance(x, y, z, r, u) means Create(z, x, delegated(x, y, z, r, u), created(z, y, r, u)); delegated(x, y, z, r, u) captures the performance of the delegation request.

Example DelegationAcceptance(S, B, S_2, paid, fan)

Consequences The parties should set up additional notifications, for example, when the delegatee has discharged the commitment, for greater confidence.

• Redundancy (RED)

Intent To mitigate risk by assuring the creditor of service by a backup agent in case things go awry.

Motivation A debtor can reduce the risk of violating its commitments by introducing a backup.

Implementation Redundancy(x, y, z, r, u) means Create(x, y, risk(x, y, r, u), created(z, y, r, u)) (x is promising backup service by z to y). Her, risk(x, y, r, u) is a domain-specific predicate that holds when a commitment is at risk of being violated.

Consequences This pattern presumes the backup agent commits to *accepting* delegations from the debtor, for example via Delegate Acceptance.

In the end, all of the above patterns are specializations of the Generic Offer pattern, except Update which is a composite pattern, and yet we are able to capture a rich set of business patterns by appropriately changing the content of the commitments.

3.2 Enactment Patterns

Whereas a business pattern describes the meaning of communication, an enactment patterns describe the conditions under which an agent should enact a business pattern, that is, *when* to undertake the corresponding communication. In general, enactment is agent-specific. Nonetheless, some behaviors are commonly observed in practice, for example, in negotiation. A locus of such enactments may serve as the basic agent skeleton. We highlight two enactment patterns that are built upon the offer business patterns presented earlier.

• Improved Offer

Intent To make improved offers via stronger commitments.

Motivation The creditor has not taken up an earlier offer.

When x makes the offer C(x, y, r, u); y has not taken up the offer, that is, r does not hold. Then, x makes a *stronger* offer C(x, y, r', u') (recall strength from Section 2) in order to entice y into the deal.

Consequences The debtor is committed more strongly; ideally, it must make sure the stronger commitment has at least some positive utility, even if diminished. This pattern represents a concession.

• Counter Offer

Intent One party makes an offer to another, who responds with a modified offer of its own.

Motivation Essential for negotiation.

When Let C(x, y, r, u) be the commitment corresponding to the original offer. Making a counteroffer would amount to creating the commitment C(y, x, u', r') such that $u' \vdash u$ and $r \vdash r'$, in other words, if the consequent is strengthened and the antecedent is weakened. An alternative implementation includes doing Release(x, y, r, u) in addition.

Consequences When $u \equiv u'$ and $r \equiv r'$, the counter offer amounts to a mutual commitment.

3.3 Semantic Antipatterns

Below, we enhance Chopra and Singh's framework with *semantic antipatterns*—forms of representation and reasoning to be avoided because they conflict with the autonomy of the participants or with a logical basis for commitments.

• Commit Another as Debtor

Intent An agent creates a commitment in which the debtor is another agent.

Motivation To capture delegation, especially in situations where the delegator is in a position of power of over the delegatee.

Implementation The sender of Create(y, z, p, q) is x, thus contravening Table 3.

Example Consider two sellers S and S_2. S sends Create(S_2, B, paid, fan) to B.

Consequences A commitment represents a public undertaking by the debtor. A special case is when $x = z$. That is, x unilaterally makes itself the creditor.

Criteria Failed S_2's autonomy is not respected.

Alternative Apply delegation to achieve the desired business relationship, based on prior commitments. In the above example, S_2 could have a standing commitment with S to accept delegations. S can then send a delegate instruction to S_2 upon which S_2 commits to B. See the DELEGATION ACCEPTANCE and REDUNDANCY business patterns in Section 3.1.

• ACKED COMMIT

Intent A commitment may hold only when the creditor has acknowledged its creation to the debtor. That is, the creditor should accept the commitment [9].

Motivation A commitment should be set up only upon the agreement of both parties. This is often based on a misunderstanding of commitments: that the creditor is committed to the antecedent.

Implementation Creditor acknowledges a create message.

Example The seller S enacts $\mathsf{BasicOffer}(\mathsf{S}, \mathsf{B}, \mathsf{paid}, \mathsf{fan})$; however, the offer does not hold until the buyer B acknowledges the offer.

Consequences It rules out unilateral commitment by the debtor such as in a business offer or advertisement for services.

Criteria Failed Autonomy (a debtor shouldn't need a creditor's approval to create a commitment) and generality (as it is unable to capture common scenarios).

Alternative MUTUAL COMMITMENT OFFER.

• COMMITMENT IDENTIFIERS

Intent Gives a unique identifier to every commitment.

Motivation To distinguish transactions and to simplify reasoning about commitments in concurrent settings, e.g., in [5, 11].

Implementation Every commitment has an ID, as in $\mathsf{C}(id, \mathsf{debtor}, \mathsf{creditor}, \mathsf{antecedent}, \mathsf{consequent})$.

Example $\mathsf{C}(id_0, \mathsf{S}, \mathsf{B}, \mathsf{paid}, \mathsf{fan})$ and $\mathsf{C}(id_1, \mathsf{S}, \mathsf{B}, \mathsf{paid}, \mathsf{fan})$

Consequences Reasoning about commitments breaks down. For example, from $\mathsf{C}(x, y, r, u) \wedge \mathsf{C}(x, y, r, v)$, one infers $\mathsf{C}(x, y, r, u \wedge v)$. However, one cannot apply such an inference to $\mathsf{C}(id_0, x, y, r, u) \wedge \mathsf{C}(id_1, x, y, r, v)$. Further, commitment operations must now explicitly refer to the identifiers in addition to the logical content.

Criteria failed Generality, since general reasoning about commitments breaks down.

Alternative BUSINESS TRANSACTION IDENTIFIER.

4. PROTOCOL SPECIFICATION

We explain how the business patterns specified above may be used by protocol designers.

Business protocols are often specified around a central exchange of goods, services, or monies. Although a simple pattern such as BASIC OFFER is usually enough to capture the exchange, typically participants want the protocols to be robust in the following ways. Table 4 summarizes how our business patterns support the robustness requirements.

Creditor Confidence (CC) Inspire confidence in the creditor about the outcome of the interaction: e.g., COMPENSATION and REDUNDANCY.

Debtor Confidence (DC) Inspire confidence in the debtor by requiring commitments from other parties: e.g., NESTED OFFER, MUTUAL COMMITMENT OFFER, and DELEGATION ACCEPTANCE.

Progress (P) Ensure liveness by requiring the involved parties to act or risk being out of compliance, e.g., NESTED OFFER and COMPENSATION (once a violation happens).

Mitigation (M) Mitigate risk for the debtor of a commitment by helping it avoid noncompliance, e.g., RELEASE INCENTIVE and DELEGATION ACCEPTANCE.

Table 4: Business patterns and robustness.

	CC	DC	P	M
NO	–	Yes	Yes	–
MCO	–	Yes	Yes	–
COM	Yes	–	Yes	–
UP	–	–	–	Yes
RI	Yes	–	–	Yes
DA	Yes	Yes	–	–
RED	Yes	–	–	Yes

A bundle of business patterns is a reusable *method* for addressing certain requirements. For example, the method $\langle \text{MCO}, \text{COM} \rangle$ addresses the requirements of creditor and debtor confidence; $\langle \text{MCO}, \text{COM}, \text{DA} \rangle$ does the same job better; $\langle \text{MCO}, \text{COM}, \text{DA}, \text{RED} \rangle$ fares even better. Alternatively, a protocol designer could choose the method $\langle \text{NO}, \text{RI} \rangle$ in order to support progress as well as mitigation. In essence, the patterns can be grouped according to the required level of robustness.

However, in selecting a method, a protocol designer would take into account not only the robustness requirements, but also the intended organizational setting. The resources of the organization and its policies would affect the method selected. For example, a fan seller ModernFans might not want to use delegation as a mitigation strategy for competitive reasons. It might also want to makes offers which its customers may take advantage of directly by making payments; in such a case, ModernFans would select a method that includes BO instead of NO or MCO. Further, the more robust a method the more computational resources the agents would need to devote during enactments—another reason a less robust method may be selected. In general, a protocol designer must make judgments about robustness versus organizational policies and resource usage.

5. CASE STUDY

We now apply the patterns to the Extended Contract Net Protocol (xCNP) formalized by Vokřínek et al. [15]. xCNP involves two roles: contractor and contractee. Vokřínek et al.'s extensions enable the negotiation of penalties in case one of the parties is unable to fulfill its end of the bargain. The xCNP protocol has three distinct phases: contract formation (similar to the traditional CNP), contract decommitment (negotiation of penalties in case one of the parties wants out, that is, before the actual violation of the contract), and contract resolution (negotiation of penalties in case of an actual violation).

Vokřínek et al. formalize xCNP in a procedural manner via a state machine (Figure 2). Many enactments are possible. For example, a contract may be reached or a penalty may be successfully negotiated; the parties could negotiate back and forth many times before reaching an agreement; they could fail to arrive at an initial contract; one of them could propose decommitment and then take back the proposal, and so on.

Table 5: Commitments used to model an xCNP-like setting.

Label	D	C	Antecedent	Consequent
pr	cte	ctr	created(ctr, cte, built(0), paid(0))	created(cte, ctr, paid(0), built(0))
co	ctr	cte	created(cte, ctr, paid(0), built(0) ∧furnished(0))	created(ctr, cte, built(0) ∧ furnished(0), paid(0))
cv	cte	ctr	created(ctr, cte, built(0) ∧ furnished(0), paid(0)) ∧ created(ctr, cte, violated(ctr, cte, ⊤, paid(0)), penalty(0))	created(cte, ctr, paid(0), built(0) ∧ furnished(0))
cvs	cte	ctr	created(ctr, cte, built(0) ∧ furnished(0) ∧driveway(0), paid(0)) ∧ created(ctr, cte, violated(ctr, cte, ⊤, paid(0)), penalty(0))	created(cte, ctr, paid(0), built(0) ∧ furnished(0)∧ driveway(0))
py	ctr	cte	built(0) ∧ furnished(0) ∧ driveway(0)	paid(0)
vio	ctr	cte	violated(ctr, cte, ⊤, paid(0))	penalty(0)
ta	cte	ctr	paid(0)	built(0) ∧ furnished(0) ∧ driveway(0)
in	ctr	cte	¬paid(0) ∧ released(ctr, cte, ⊤, paid(0))	released(cte, ctr, ⊤, built(0) ∧ furnished(0)∧ driveway(0)) ∧ expensesPlusTen(0)
in_R	ctr	cte	⊤	expensesPlusTen(0)

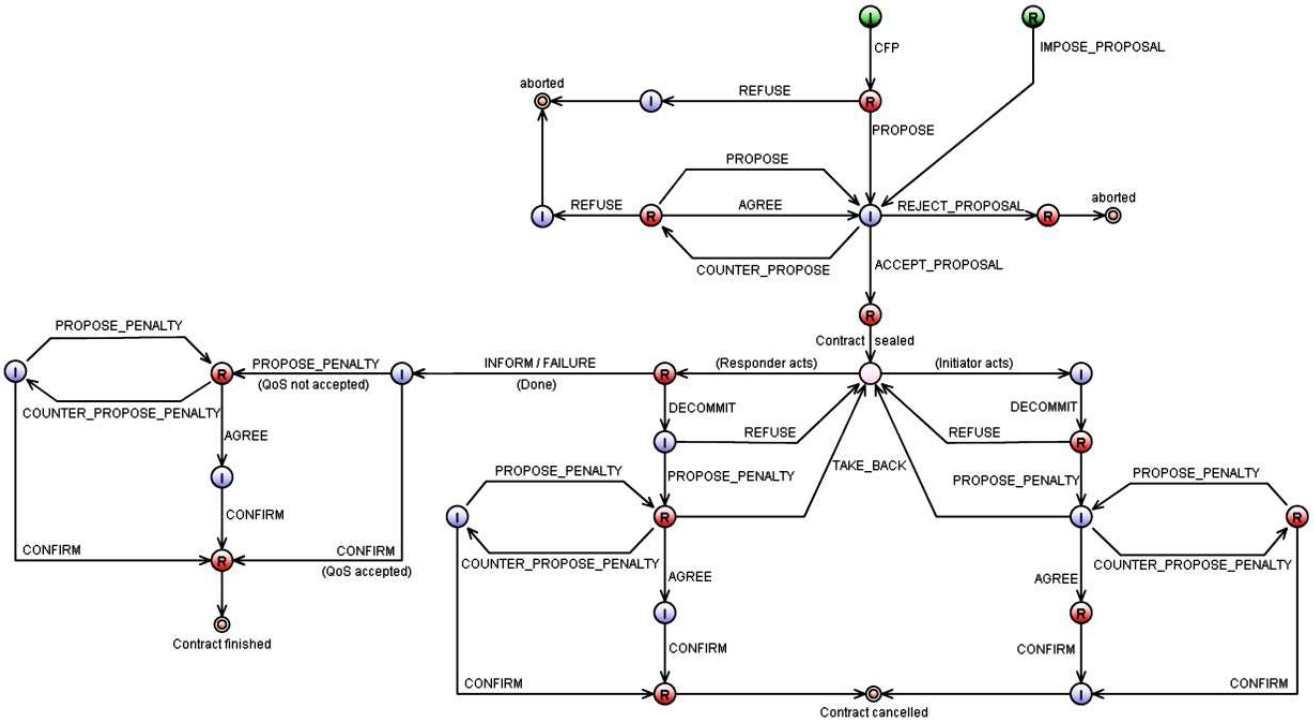

Figure 2: The xCNP protocol [15]. I and R refer to the contractor (ctr) and the contractee (cte), respectively.

5.1 Applying our Approach

We replace the operational model of xCNP with a model based on the appropriate business patterns.

- xCNP emphasizes the synchronizing agree-confirm operational pattern for arriving at any outcome: in contract formation (after one party agrees to a proposal, the other must confirm it), in penalty negotiation, and so on. We instead use Mutual Commitment Offer (MCO) or Nested Offer (NO).
- In order to enable parties to get out of their commitment, xCNP supports decommitment. In our framework, Release Incentive (RI) captures decommitment: the commitment is not yet violated, and the debtor is asking to be released by the creditor in return for a penalty (incentive from the creditor's point of view). An alternative set of patterns for implementing decommitment consists of Cancel Offer (CO) and Compensation (COM, as proposing a penalty for the cancellation).
- xCNP supports penalties for violation to capture contract resolution. We can instead use Compensation.

Thus, to capture a contract protocol, one can choose from the following business pattern methods.

Method 1. ⟨NO, RI, COM⟩
Method 2. ⟨MCO, RI, COM⟩
Method 3. ⟨NO, CO, COM⟩
Method 4. ⟨MCO, CO, COM⟩

As stated earlier, the choice of the method depends not

only upon the robustness criteria but also upon organizational requirements. For example, Method 2 is more robust than Method 4 because cancellation is in effect a violation. However, a business partner could still choose Method 4 if it did not care about violations as much as it cared about immediacy (in the sense that it does not have to *wait* to be released by the other party).

The four methods above are just samples; in general, designers could come up with more patterns and methods that meet various requirements.

5.2 Enactment

Table 5 lists the commitments used in xCNP; $name_U$ is the unconditional commitment resulting from *name*. Figure 3 shows an enactment of the contract formation stage using Method 2. The scenario is one where a contractor issues a CFP for an office block construction. Let's consider Figure 3 step by step.

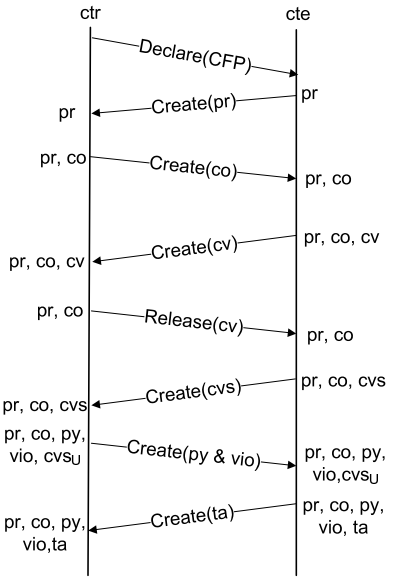

Figure 3: Method 2: Contract formation enactment.

1. Contractor ctr applies the DECLARE pattern in sending the CFP.
2. Contractee cte enacts the MCO pattern in response (to create *pr*): essentially the contractee will do built in return for paid.
3. ctr does a COUNTER OFFER in response (to create *co*): in addition to built, the contractor also wants furnished.
4. cte does a COUNTER OFFER in response (to create *cv*): the contractee is ready to do built and furnished for paid, but wants contractor ctr to commit to paying a penalty in case ctr cannot pay for the services rendered.
5. ctr applies REJECT OFFER in response.
6. cte then applies IMPROVED OFFER: it sweetens the offer by throwing in driveway.
7. ctr then creates the necessary commitments using the BASIC OFFER pattern; presumably the contractor is happy with the improved offer.
8. cte also creates the necessary commitments.

Figure 4 shows an enactment of the contract decommitment stage using Method 2. The figure begins from where the interaction has progressed so the commitments *py* and *ta* hold. Let's look at Figure 4 step by step.

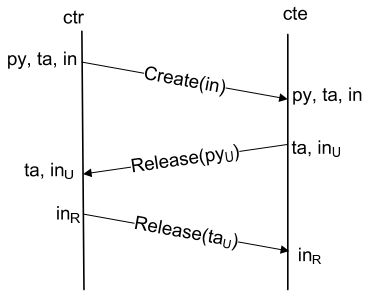

Figure 4: Method 2: Decommitment enactment.

- ctr applies the RELEASE INCENTIVE pattern (to create *in*): if cte releases it from py_U and payment has not yet happened, ctr will release cte from ta_U and reimburse cte to the extent of 110% of the expenses cte has already incurred.
- In response, cte releases it from py_U.
- In response, ctr releases cte from ta_U. At this point, in_R holds: ctr must still pay cte 110% of the expenses.

5.3 Observations and Conclusions

xCNP, as formalized by Vokřínek et al., does not consider the meanings of interactions. For example, it does not formalize what it means to decommit. By contrast, we formalized decommitment via two alternative patterns that have different ramifications for meeting organizational requirements. In addition, one of the alternatives turned out to be a composite pattern (CANCEL OFFER and COMPENSATION). Operational formalizations miss out on such nuances.

We showed four alternative methods that model the enhancements xCNP claims over the traditional CNP. We gave an example requirement of what might drive a protocol designer to choose one method over another. Since our methods are meaning-based, it is natural for a designer to select among them than from among the same number of alternative operational formalizations of xCNP.

All our methods draw from the patterns we introduced earlier, which themselves draw from the basic framework in [2]. Thus business-level interoperability is guaranteed even when agents enact the patterns asynchronously. By contrast, Vokřínek et al.'s xCNP formalization is both over-specified and under-specified. It is over-specified because it is highly synchronous and enforces rigid enactments, such as via the agree-confirm pattern for arriving at any outcome. It is under-specified in net effect because it cannot capture enactments that would be natural. For example, both the contractor and contractee (I and R, respectively in Figure 2) may act concurrently by sending CFP and IMPOSE_PROPOSAL, respectively. Although these transitions are allowed, the resulting state is not captured in the formalization. A similar situation ensues when, after sealing the contract, both parties act concurrently in order to decommit. In general, it is difficult to capture all possible executions paths via operational methods because of their lower level of abstraction.

One could try to repair Vokřínek et al.'s formalization by inserting additional enactment paths to address its rigidity and insert additional synchronizations to address messages crossing in transit (the latter would increase rigidity).

However, such a formalization would be overly complex, unwieldy to maintain, and difficult both for designers and end users to understand.

In conclusion, when the business meanings of interactions are made explicit, (1) designers gain in flexibility in selecting from a range of possible specifications, that is, the methods, and (2) agents gain in flexibility in enacting the specified system because they can reason about meanings and select among alternative courses of action.

6. DISCUSSION

Method engineering is an expanding area of SE. Traditionally, method engineering considers how to engineer and choose among methods in the large, such as Agile or Scrum [10], with selection based on the structure of the given software development organization. In contrast with existing work, we observe that a (modeling) method could be understood in terms of the families of interactions that we wish to support among agents, such as the partners in business processes. We too are concerned with organizations, but emphasize the organization of the business partners during enactment as well as the contextual organization in which the business process takes place. We envisage that suitable methods would be engineered based on features of such organizations as well as the flexibility supported by the business partners' agents. And designers who apply selected methods would create models of interaction that naturally meet those criteria.

Our approach to patterns is layered: method over business over semantic patterns. Lind and Goldkuhl [7] propose a layered approach to business modeling starting with business actions and building up to transactions; however, they overlook the meanings of the business actions themselves.

Conceptually any protocol, no matter how specified, is a reusable pattern of interaction. Existing approaches for protocol composition, e.g., [8, 14], focus on procedural aspects, which though valuable cannot substitute for business meanings. Singh et al. [12] motivate some commitment-based connector patterns, including multiparty ones. However, they do not consider the challenges of distributed enactment.

Traditionally, researchers have used action logics for commitment protocol specification, for example, as in [6]. Chopra and Singh [1] support the application of business patterns, such as for *Return and Refund*, to protocols specified in an action logic. However, these approaches assume synchronous communication and, further, freely mix meaning axioms along with operational constraints such as for message ordering. By contrast, our patterns are purely meaning-based, and they can be enacted asynchronously.

Wang et al. [16] annotate each commitment with types depending on the relative order in which its antecedent and consequent ought to be satisfied. For example, they annotate $\mathsf{C}(\mathsf{merchant}, \mathsf{customer}, \mathsf{payment}, \mathsf{refund})$ *strictly-ordered*: payment must be made before the refund can be made. However, *payment-before-refund* can be an enactment policy—a choice—on the part of the merchant; the ordering is not necessarily an issue of commitment specification. Annotating commitments as Wang et al. do unduly limits flexibility during enactment. In general, it is important to sort out the issues of agent specification from those of protocol specification [3].

Future directions include coming up a rich taxonomy of requirements that pertain to interactions, and providing tool-based support to designers for picking from among the methods in a repository.

Telang and Singh [13] propose a metamodel in which to express cross-organizational business models that includes a set of modeling patterns. They formalize commitments in a simplified temporal semantics assuming synchrony and show how to verify low-level protocols expressed in sequence diagrams with respect to the business models. It would be interesting to reconcile our approach with theirs.

Acknowledgments

We thank the reviewers for their helpful comments. Chopra was supported by a Marie Curie Fellowship.

7. REFERENCES

[1] A. K. Chopra, M. P. Singh. Contextualizing commitment protocols. *AAMAS*, pp. 1345–1352, 2006.

[2] A. K. Chopra, M. P. Singh. Multiagent commitment alignment. *AAMAS*, pp. 937–944, 2009.

[3] A. K. Chopra and F. Dalpiaz and P. Giorgini and J. Mylopoulos. Reasoning about agents and protocols via goals and commitments. *AAMAS*, pp. 457–464, 2010.

[4] N. Desai, A. K. Chopra, M. Arrott, B. Specht, M. P. Singh. Engineering foreign exchange processes via commitment protocols. *IEEE SCC*, pp. 514–521, 2007.

[5] N. Fornara, M. Colombetti. Operational specification of a commitment-based agent communication language. *AAMAS*, pp. 535–542, 2002.

[6] L. Giordano, A. Martelli, C. Schwind. Specifying and verifying interaction protocols in a temporal action logic. *J. Applied Logic*, 5(2):214–234, 2007.

[7] M. Lind, G. Goldkuhl. The constituents of business interaction—generic layered patterns. *Data & Knowledge Engineering*, 47(3):327–348, 2003.

[8] H. Mazouzi, A. E. F. Seghrouchni, S. Haddad. Open protocol design for complex interactions in multi-agent systems. *AAMAS*, pp. 517–526, 2002.

[9] P. McBurney, S. Parsons. Posit spaces: A performative model of e-commerce. *AAMAS*, pp. 624–631, 2003.

[10] A. Qumer, B. Henderson-Sellers. An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and Software Technology*, 50(4):280–295, 2008.

[11] M. Rovatsos. Dynamic semantics for agent communication languages. *AAMAS*, pp. 1–8, 2007.

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

[13] P. R. Telang and M. P. Singh. Specifying and verifying cross-organizational business models. *IEEE Trans. Services Comput.*, 4, 2011.

[14] B. Vitteau, M.-P. Huget. Modularity in interaction protocols. *Proc. ACL*, *LNCS* 2922, pp. 291–309, 2004.

[15] J. Vokřínek, J. Bíba, J. Hodík, J. Vybíhal, M. Pěchouček. Competitive contract net protocol. *SOFSEM: Theory and Practice of Computer Science*, *LNCS* 4362, pp. 656–668, 2007.

[16] M. Wang, K. Ramamohanarao, J. Chen. Reasoning intra-dependency in commitments for robust scheduling. *AAMAS*, pp. 953–960, 2009.