Representing and Reasoning About Commitments in Business Processes*

Nirmit Desai and **Amit K. Chopra** and **Munindar P. Singh**
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
{nvdesai, akchopra, singh}@ncsu.edu

Abstract

A variety of business relationships in open settings can be understood in terms of the creation and manipulation of *commitments* among the participants. These include B2C and B2B contracts and processes, as realized via Web services and other such technologies. *Business protocols*, an interaction-oriented approach for modeling business processes, are formulated in terms of the commitments. Commitments can support other forms of semantic service composition as well. This paper shows how to represent and reason about commitments in a general manner. Unlike previous formalizations, the proposed formalization accommodates complex and nested commitment conditions, and concurrent commitment operations. In this manner, a rich variety of open business scenarios are enabled.

Introduction

Business processes in Web-based IT environments are often complex and interorganizational, and involve rich interactions among services representing autonomous, heterogeneous parties. There is a growing consensus that in order to engineer modern business solutions, we must emphasize business-level elements such as contracts over technical elements such as control and data flows.

Commitments capture the contractual essence of business interactions. They can be thought of as directed obligations that have been reified, and possibly conditionalized (Singh 1999). For example, a merchant may commit to a customer to delivering specified goods *if* a specified payment is received. The commitment can be manipulated: e.g., the merchant may delegate the commitment to delivering to an independent franchise owner. Other such manipulations are possible, leading to perspicuous descriptions of behaviors that bypass low-level details and support the dynamic reconfiguration of businesses (e.g., by introducing new participants such as a franchisee) without altering the overall structure of the merchant-customer interaction. Associated with this power is the need to specify the allowed manipulations precisely and to constrain possible behaviors in a manner that guarantees the business requirements without unnecessarily restricting the autonomy of the participants.

Classically, a process can be specified as an *orchestration* (describing the control and data flows among services that are centrally invoked) or a *choreography* (describing the messages exchanged among services in a distributed manner). Traditional representations, however, either ignore semantics altogether or merely specify the semantics at a low level, i.e., in terms of a finite state machine (e.g., Fu *et al.* (2004)) or a Petri net (van der Aalst & van Hee 2002). Both, however, ignore the "meanings" (in the business sense) that the representation supports. For example, with a low-level representation, we cannot easily determine if introducing a franchisee into a business interaction is sound.

Semantic Web services, e.g., (OWL-S 2004), are a leading AI approach. In simple terms, these provide representations of services that would support composing them via an application of automatic planning based on their IOPEs: inputs, outputs, preconditions, and effects. Conventionally these compositions have been viewed from a single perspective, and can be thought of as *semantic orchestration*. Choreography is more general than orchestration, because it can support the request-response pattern of orchestration as well as other more complex interaction patterns. For this reason, we present our results in the context of choreography. In particular, our interest is in enriching choreographies with meaning, i.e., *semantic choreography*.

Choreographies modeled via commitments correspond to (business) *protocols* (Fornara & Colombetti 2003; Desai *et al.* 2005; Winikoff 2007; Desai & Singh 2007). Although well-motivated and conceptually valuable, existing approaches lack a rigorous semantics for commitments. This limits the applicability of formal methods, for instance. Moreover, the above papers make simplifying assumptions that limit their applicability in practical business scenarios.

It is important to note that commitments are a fundamental abstraction for (contractual) business relationships. Commitments arise independently of the underlying platform. For example, you would be equally committed to paying for your purchase whether you submit your order using remote procedure call (RPC) or messaging. Consequently, if commitments are understood properly, they can be used in RPC-based service composition just as easily as in protocols.

This paper builds on (Chopra & Singh 2006), which pro-

*With partial support from the US National Science Foundation under grant IIS-0139037.

vides a basic formalization of commitment protocols in the $\mathcal{C}+$ action description language (Giunchiglia *et al.* 2004). It goes beyond previous work on representing and reasoning about commitments in two main respects.

Complex commitments The proposed formalization supports the modeling of complex commitments, where the condition of a commitment may itself be a logical formula or another commitment. Complex commitments are common in real life. For instance, an insurance company commits to covering a customer if the premium is paid, where covering the customer in turn means committing to processing claims, providing repair services, and so on.

Concurrent operations The proposed formalization supports concurrent operations on commitments. The need to handle concurrent commitment operations arises from the fact that typically the interacting parties are organizations, and individual agents within such organizations may act concurrently on the extant commitments.

This paper introduces an ontology for commitments including concurrent commitment operations, shows how to apply it in modeling a protocol for ordering goods, and shows how to accommodate complex commitments.

Commitments and Protocols

A *protocol* specifies a set of roles, the messages they exchange, the meanings of the messages, and any constraints on message order or occurrence. Specifically, the meanings of messages include their effects on the commitments among the agents playing the specified roles.

Commitments

Following Singh (1999), we define a commitment $\mathsf{cc}(x, y, p, q)$ as an obligation from a debtor x to a creditor y to satisfying the condition q if p holds. Here p is the precondition and q is the condition of the commitment. Consider, for example, a scenario where a buyer and a seller are exchanging goods for payment. The commitment cc(buyer, seller, goods, payment) denotes an obligation from the buyer to the seller that if the goods are delivered, then the buyer will pay.

Unlike an obligation as traditionally understood, a commitment can be manipulated. Singh describes six operations on commitments: create, discharge, delegate (changing the debtor), assign (changing the creditor), release (creditor releasing the debtor from the commitment), and cancel (debtor canceling the commitment).

A commitment $\mathsf{cc}(x, y, p, q)$ may be *base* (p=T) or *conditional* (p$\neq$T). Conditional commitments yield base commitments via an operation termed toBase. For example, if the precondition goods holds, the above conditional commitment would change to a base commitment that the buyer would make the payment. Conditional commitments can be satisfied if their condition holds, regardless of the precondition.

Specifying Commitments in $\mathcal{C}+$

Listing 1 shows a partial specification of commitments in $\mathcal{C}+$. The object cc(Role,Role,Condition,Condition) (line 15) denotes a set of commitments (each of sort Commitment) ranging over Role and Condition objects. A multivalued inertial fluent comm(Commitment) (line 21) represents the state of each commitment. The domain of this fluent is thus the sort State. The exogenous actions act(Message) (line 19) correspond to protocol messages, and the inertial fluents fl(Message) (line 18) record occurrences of corresponding messages. The causation of commitment conditions is modeled as a set of actions cond(Condition) (line 20), one per condition object. The commitment operations are modeled similarly (lines 23–26).

Commitments properly should be specified with a unique identifier. For example, if x commits twice to y to pay \$1, a single payment of \$1 should discharge only one of them. This can be handled by having an identifier as a parameter in commitment objects and cond action constants. When a cond with a particular ID is caused, only the commitment with a matching ID is discharged. The present formalization skips these details.

Listing 1: Commitment ontology (part 1: declarations)

```
1  :- sorts
2   Role; Slot; Message; State; Commitment; Condition.

4  :- variables
5   msg :: Message;  p,q :: Condition;
6   c1 :: Commitment;  db1,cr1,db2,cr2 :: Role.

8  :- objects
9   NIL, BASE, CONDITIONAL, RELEASED, SATISFIED,
10  BASE_DELEGATED, CONDITIONAL_ASSIGNED, BASE_ASSIGNED,
11  CONDITIONAL_DELEGATED, BASE_ASSIGNED_DELEGATED,
12  CONDITIONAL_ASSIGNED_DELEGATED, CANCELLED :: State;

14  T :: Condition;
15  cc(Role,Role,Condition,Condition) :: Commitment.

17 :- constants
18  fl(Message) :: inertialFluent;
19  act(Message) :: exogenousAction;
20  cond(Condition) :: action;
21  comm(Commitment) :: inertialFluent(State);

23  create(Commitment), discharge(Commitment),
24  toBase(Commitment), delegateTo(Role,Commitment),
25  assignTo(Role,Commitment), cancel(Commitment),
26  release(Commitment) :: action;

28  active(Commitment), initial :: sdFluent.
```

Listing 2 continues the commitment ontology. The static fluent initial is used to obtain an initial state void of any message or commitment fluents (lines 1–3). The static fluent active denotes a commitment's being in one of the states considered active (lines 9–13). Also, each message action occurrence is recorded as an inertial fluent (line 5). The commitment operation actions are disabled by default (lines 15–18).

Listing 2: Commitment ontology (part 2: operations)

```
1  caused initial if initial.
```

```
2  caused −initial if comm(c1)\=NIL.
3  caused −initial if fl(msg).

5  act(msg) causes fl(msg).

7  −cond(p) causes −cond(p).

9  caused −active(c1) if −active(c1).

11 caused active(c1) if
12   comm(c1)\=NIL & comm(c1)\=SATISFIED &
13   comm(c1)\=RELEASED & comm(c1)\=CANCELLED.

15 −create(c1) causes −create(c1).
16 −discharge(c1) causes −discharge(c1).
17 ...
18 −release(c1) causes −release(c1).
```

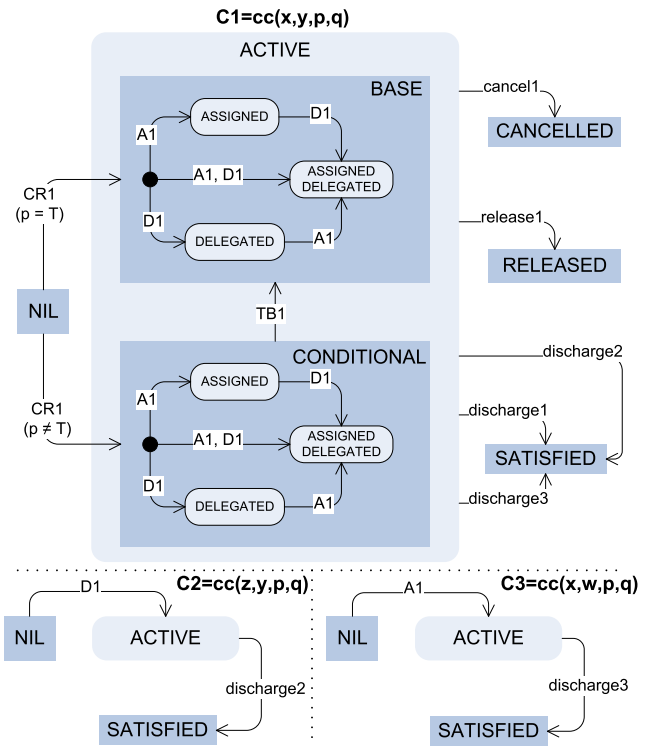

Figure 1: Life cycle of a commitment in three parts

In Figure 1, the state machine for C_1 describes the primary life cycle for a commitment. The delegate and assign transitions on C_1 spawn additional commitments C_2 and C_3, respectively. C_2 and C_3 are full-fledged commitments themselves and thus follow the life cycle indicated by the state machine for C_1. However, the satisfaction of C_2 and C_3 causes the satisfaction of C_1.

Let's now explain Figure 1 in more detail. The abbreviated labels D denote delegate, A assign, CR create, and TB toBase. These labels are appended with the index of the commitment to which they apply. For example, TB1 denotes toBase(C_1). Edges with multiple labels denote concurrent operations. Solid circles denote default states of the respective composite states. Transitions out of composite states can be taken concurrently with the transitions within the composite state. Thus, TB1 can be concurrent with A1, D1, or both. For example, if the current state is the default state of the composite state CONDITIONAL, and TB1 occurs concurrently with A1 and D1, the resulting state is ASSIGNED_DELEGATED in the BASE composite state.

Initially, the state of a commitment is NIL. During enactment, when a commitment is created, its state changes to either BASE (p=T) or CONDITIONAL (p≠T). The commitment may be assigned, delegated, or both. If the commitment is conditional (composite state CONDITIONAL), it changes to a base commitment (composite state BASE), if its precondition holds. An active commitment can be released, cancelled, or discharged, causing state changes.

When a commitment (interpreted as C_1) is delegated, its state is updated and a new commitment (interpreted as C_2) for the delegatee is created. Thus, C_2 becomes active when delegate(C_1) happens. Similarly, in the case of assign, a new commitment for the new creditor is created making C_3 active. Naturally, C_2 and C_3 can themselves be interpreted as C_1, and thus delegated or assigned.

Specifying Commitment Operations in $\mathcal{C}+$

In an organizational setting, a commitment could be acted upon simultaneously by more than one agent. Such independent actions by agents may potentially cause the commitment state to be ambiguous. Thus, it is important to prioritize the operations to obtain a desirable state of the commitment. For example, if the creditor is an organization consisting of agents Alice and Bob, Alice could release the debtor from a conditional commitment, while Bob satisfies the precondition of the same commitment (thereby causing toBase, which results in an inconsistent state of the commitment). Concurrent operations of the debtor and the creditor could also be problematic. Therefore, we handle concurrent commitment operations according to a priority scheme.

Table 1 shows the proposed priority scheme. A '$\succ$' in a cell means the row operation overrides the column operation if they happen concurrently on the same commitment. For example, toBase overrides a concurrent create. A '$\|$' means that the row and column operations may happen concurrently. For example, a concurrent delegate and toBase would result in the delegator's original commitment's state changing to BASE_DELEGATED from CONDITIONAL (à la C_1), and a new commitment (à la C_2) being created in which the delegatee is the debtor, and the state of which is BASE.

Table 1 is configurable, and thus incomplete. For example, prioritizing release over cancel may not always be reasonable. Such situations are marked with a '?' indicating their configurability. A protocol designer may choose a priority scheme depending on domain requirements. The following formalization assumes discharge $\succ$ release and release $\succ$ cancel. Table 1 does not show the parallel relationship among the concurrent transitions TB1, A1, D1. And, the column for assign, which would be similar to that of delegate, is skipped. Listing 3 axiomatizes the commitment operations according to the priority scheme of Table 1 and the life cycle of Figure 1. The axiom of lines 1–2 means that

Table 1: Override relationships among concurrent commitment operations

Operations	create	toBase	delegate	cancel	release
toBase	$\succ$				
delegate	$\succ$	$\|\|$			
assign	$\succ$	$\|\|$	$\|\|$		
cancel	$\succ$	$\succ$	$\succ$		?
release	$\succ$	$\succ$	$\succ$	?	
discharge	$\succ$	$\succ$	$\succ$	$\succ$	?

a discharge occurs when the condition of a commitment is met and the commitment is either active or being created. Similarly, toBase is caused if the precondition of the commitment is met (lines 4–6). These two are the only operations not caused directly by agents.

The axiom of lines 8–15 describes the transition NIL $\rightarrow$ BASE. If create is caused on a NIL commitment, and none of the operations that override create are caused concurrently, then the state of the newly created commitment is BASE (its precondition being T). A similar axiom would model the transition NIL $\rightarrow$ CONDITIONAL. Similarly, axioms for discharge, release, and cancel are defined according to the override relationships.

Listing 3: Commitment ontology (part 3: rules)

```
1  caused discharge(cc(db1,cr1,p,q)) if cond(q) &
2  (active(cc(db1,cr1,p,q)) ++ create(cc(db1,cr1,p,q))).

4  caused toBase(cc(db1,cr1,p,q)) if cond(p) &
5  (active(cc(db1,cr1,p,q)) ++ create(cc(db1,cr1,p,q)))
6  where db1<>cr1 & p<>T.

8  caused comm(cc(db1,cr1,T,q))=BASE if true after
9  create(cc(db1,cr1,T,q)) &
10 -(discharge(cc(db1,cr1,T,q)) ++
11   release(cc(db1,cr1,T,q)) ++
12   delegateTo(db2,cc(db1,cr1,T,q)) ++
13   cancel(cc(db1,cr1,T,q)) ++
14   assignTo(cr2,cc(db1,cr1,T,q))
15  ) & comm(cc(db1,cr1,T,q))=NIL where db1<>cr1 & q<>T.

17 caused comm(c1)=SATISFIED if true after discharge(c1).

19 caused comm(c1)=RELEASED if true after release(c1) &
20 -discharge(c1) & (active(c1) ++ create(c1)).

22 caused comm(c1)=CANCELLED if true after cancel(c1) &
23 -(discharge(c1) ++ release(c1)) &
24 (active(c1) ++ create(c1)).
```

Listing 4 continues with the axioms for commitment operations. The axiom of lines 1–4 means that a commitment in state CONDITIONAL enters state BASE if toBase is caused and the operations overriding toBase are not caused concurrently. Even though delegate can happen concurrently with toBase, this axiom blocks it as the desirable effect in that case would be BASE_DELEGATED instead of BASE as in this case. Similar axioms for other TB1 transitions are skipped.

When an agent x delegates a commitment to another agent y, the original commitment is changed to reflect its DELEGATED state and a new commitment for the new debtor y is created. The axiom of lines 6–10 models delegate by following the pattern and complying to delegate||assign, and delegate||toBase. Similar axioms for transitions D1 are not shown here. The axiom of lines 12–22 captures the creation of a new commitment if the delegate is not being overridden and the original commitment is active or being created and has not been already delegated. The axiom of lines 24–28 handles concurrent toBase and delegate. Another such axiom would capture the transition CONDITIONAL_ASSIGNED $\rightarrow$ BASE_ASSIGNED_DELEGATED.

Axioms relating to assign follow the same pattern as those for delegate and are not shown here. Concurrent delegate|| assign, assign||toBase, and toBase||assign||delegate are handled analogously to toBase||delegate.

Listing 4: Commitment ontology (part 4: rules)

```
1  caused comm(c1)=BASE if true after toBase(c1) &
2  -(discharge(c1) ++ release(c1) ++ cancel(c1) ++
3    delegateTo(db2,c1) ++ assignTo(cr2,c1)
4   ) & comm(c1)=CONDITIONAL.

6  caused comm(c1)=CONDITIONAL_DELEGATED if true after
7  delegateTo(db2,c1) & (active(c1) ++ create(c1)) &
8  -(discharge(c1) ++ release(c1) ++ cancel(c1) ++
9    assignTo(cr2,c1) ++ toBase(c1)
10  ) & comm(c1)=CONDITIONAL.

12 caused create(cc(db2,cr1,p,q)) if
13 delegateTo(db2,cc(db1,cr1,p,q)) &
14 -(discharge(cc(db1,cr1,p,q)) ++
15   release(cc(db1,cr1,p,q)) ++
16   cancel(cc(db1,cr1,p,q)) ++
17   assignTo(cr2,cc(db1,cr1,p,q))) &
18 (active(cc(db1,cr1,p,q)) ++ create(cc(db1,cr1,p,q)))
19 & (comm(cc(db1,cr1,p,q))=BASE ++
20    comm(cc(db1,cr1,p,q))=CONDITIONAL ++
21    comm(cc(db1,cr1,p,q))=BASE_ASSIGNED ++
22    comm(cc(db1,cr1,p,q))=CONDITIONAL_ASSIGNED).

24 caused comm(c1)=BASE_DELEGATED if true after
25 delegateTo(db2,c1) & toBase(c1) &
26 -(discharge(c1) ++ release(c1) ++
27   cancel(c1) ++ assignTo(cr2,c1)) &
28 (active(c1) ++ create(c1)) & comm(c1)=CONDITIONAL.
```

The listings of this paper are posted online (MAS Lab 2007).

A Purchase Example

Consider the simplified purchase process often studied in the literature (e.g., (Desai *et al.* 2005)). The *Order* protocol defines the interactions between a *Customer* and a *Merchant* role in reaching an agreement on a purchase. The *Customer* requests a quote for an item, to which the *Merchant* responds by quoting a price. The meaning of quoting a price is that it creates a commitment from the *Merchant* to the Customer that if the price is paid, the goods would be delivered. The *Customer* can then either accept or reject the quote (the reject case is at (MAS Lab 2007)). The meaning of acceptance

is that the *Customer* commits to paying for the delivered goods. Listing 5 specifies *Order*.

Listing 5: *Order* protocol

```
:- include 'commitment_ontology'.

:- sorts
  Slot >> Item;  Slot >> ItemPrice;
  Role >> Customer;  Role >> Merchant.

:- objects
  reqForQuote(Customer, Merchant, Item) :: Message;
  quote(Merchant, Customer, Item, ItemPrice) :: Message;
  accept(Customer, Merchant, Item, ItemPrice) :: Message;
  deliver(Item) :: Condition;
  payment(ItemPrice) :: Condition;
  c :: Customer;  m :: Merchant;
  myItem :: Item;  myPrice :: ItemPrice.

:- variables
  item :: Item;  price :: ItemPrice.

nonexecutable act(reqForQuote(c,m,item)) if
  fl(reqForQuote(c,m,item)).

nonexecutable act(quote(m,c,item,price)) if
 -fl(reqForQuote(c,m,item)) ++
  fl(quote(m,c,item,price)).

nonexecutable act(accept(c,m,item,price)) if
 -fl(quote(m,c,item,price)) ++
  fl(accept(c,m,item,price)).

act(quote(m,c,item,price)) causes
  create(CC(m,c,payment(price),deliver(item))).

act(accept(c,m,item,price)) causes
  create(CC(c,m,deliver(item),payment(price))).

:- query
  label :: 5;  maxstep :: 4;  0: initial;
  maxstep: fl(accept(c,m,item,price)).
```

The nonexecutable rules control the autonomy of the participants. In this protocol, no message can be exchanged twice with the same parameters. Thus, once the *Customer* requests a quote for an item, it cannot request it again for the same item in the same instance of the protocol. Such repeated messages can be allowed by not restricting the execution of the action even though the corresponding fluent holds. The standard model of *Order* is shown in Figure 2(a).

Listing 6: Adding Shipping and Payment to *Order* protocol

```
nonexecutable act(goods(m,c,item)) if
-fl(accept(c,m,item,price)) ++ fl(goods(m,c,item)).

act(goods(m,c,item)) causes cond(deliver(item)).

nonexecutable act(pay(c,m,price)) if
-fl(accept(c,m,item,price)) ++ fl(pay(c,m,price)).

act(pay(c,m,price)) causes cond(payment(price)).
```

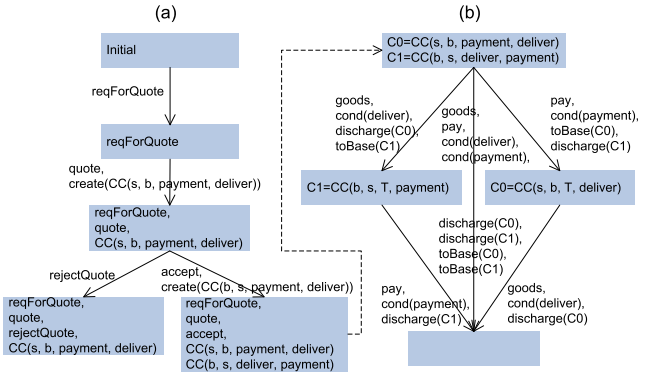

Figure 2: Standard model of *Order* and its extension

For illustration, let's introduce the axioms of Listing 6 into *Order*. Figure 2(b) shows the desired behavior of commitments as their conditions and preconditions are met. For clarity, the states only show the commitment fluents. Notice that discharge and toBase are caused automatically when the conditions and preconditions are respectively met.

Let us examine Figure 2(b) further. According to Listing 6, messages goods and pay can be exchanged concurrently, as denoted by the middle transition from the state labeled C_0 and C_1 (at the top) to the empty state (at the bottom). If discharge did not override toBase, there would be a contradiction as toBase would attempt to change the state to BASE while discharge is changing the state to SATISFIED. As the conditions of both C_0 and C_1 are being met, the desired outcome is the effect of the discharge operation alone.

Complex Commitment Conditions

Commitments in the above example involve only atomic conditions. In practice, preconditions and conditions can be more complex formulas, including those with nested commitments. The ability to represent and reason about such conditions is vital. For example, assume that in the *Order* protocol, the *Merchant* also sends a receipt of payment. Say the meaning of the quote message is that the *Merchant* commits to sending goods and providing a receipt if the payment is made for the quoted price.

Listing 7: Order with receipt

```
act(quote(m,c,item,price)) causes
create(CC(m,c,payment(price),
delivernreceipt(item,price))).

caused cond(delivernreceipt(item, price)) if
(act(goods(m,c,item)) & fl(receipt(m,c,price))) ++
(fl(goods(m,c,item)) & act(receipt(m,c,price))) ++
(act(goods(m,c,item)) & act(receipt(m,c,price))).
```

Listing 7 captures this commitment. The inner condition (delivernreceipt) acts as a placeholder. It is related to the appropriate logical expression via a separate axiom. As the receipt follows payment but can be concurrent with goods, either of the three possible combinations of actions and fluents can cause the condition. In general, a commitment

$\mathsf{cc}(x, y, \phi, \psi)$ where ϕ and ψ are arbitrary formulas, can be represented as $\mathsf{cc}(x, y, p_\phi, p_\psi)$ where p_ϕ and p_ψ are placeholders (albeit with all the parameters of ϕ and ψ, respectively), and separate axioms state that p_ϕ is caused by ϕ and p_ψ is caused by ψ.

Now, let us consider the case of nested commitment conditions. Say, the meaning of quote is that the *Merchant* commits to committing to delivering the goods if the payment is made. While this may sound redundant, it enables a degree of flexibility found commonly in practical situations. For example, the debtor of the inner commitment can be a third party, such as a shipper. Thus, the *Merchant* can promise the *Customer* that a third party would commit to delivering the goods if the payment is made. Also, if the condition itself is a commitment, it can be manipulated in various ways by the usual operations. Listing 8 shows the modifications.

Listing 8: Nested commitment in Order protocol

```
1  act(quote(m,c,item,price)) causes
2  create(CC(m,c,payment(price), commitToDeliver(item))).

4  caused cond(commitToDeliver(item)) if
5  create(CC(m,c,deliver(item))).
```

The creation of the inner commitment causes the placeholder condition. The debtor role in the inner commitment could as well be a party other than the merchant. The create itself would be caused separately. In general, a nested commitment $\mathsf{cc}(x, y, \mathsf{T}, \mathsf{cc}(z, w, \mathsf{T}, p))$ can be represented as $\mathsf{cc}(x, y, \mathsf{T}, cc_p)$ where cc_p is a place holder atomic condition having all the parameters of p and a rule that cc_p is caused by creation of $\mathsf{cc}(z, w, \mathsf{T}, p)$. The technique naturally extends to conditional commitments and arbitrary levels of nesting.

Discussion

Causal logic specifications can be translated into transition systems. However, as programming language abstractions, transition systems are unwieldy at best. Causal logic specifications can be refined with ease; they support elaboration tolerance (McCarthy 2003). Hence, the ability to derive transition systems from causal logic specifications is an added bonus as existing verification techniques can be applied on such transition systems.

Obligations have been shown to be useful in modeling contracts among entities interacting over the Web (Tan & Thoen 1998; Foster *et al.* 2006). Obligations can take complex formulas and can be nested. Minsky and Ungureanu (2000) propose law governed interactions with a stronger notion of obligations where the agents are guaranteed to keep their obligations as occurrences of violations are prevented. Grosof and Poon (2004) represent contracts and exceptions in a rule-based framework. However, unlike commitments, their contracts cannot be delegated or assigned.

Unlike directed obligations, which have been previously formalized in modal logic, commitments can be easily manipulated as the interaction progresses. Commitments enable flexible modeling and enactment, and accommodating organizational changes. Modeling complex commitments and handling concurrent commitment operations in interorganizational settings are the main contributions of this paper.

Chopra and Singh (2006) show how a protocol modeled in causal logic may be transformed to adapt to a context. Winikoff (2007) discusses distributed enactment of commitment machines for agent interactions. Fornara and Colombetti have studied the basic life cycle of commitments (2003). However, these papers do not discuss the challenges of concurrent operations and complex commitments.

In future, it would be interesting to explore the effects of organizational structures on the management of commitments, e.g., monitoring progress and falling back to the superiors in the case of a failure.

References

Chopra, A. K., and Singh, M. P. 2006. Contextualizing commitment protocols. In *AAMAS*, 1345–1352.

Desai, N., and Singh, M. P. 2007. A modular action description language for protocol composition. In *AAAI*.

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

Fornara, N., and Colombetti, M. 2003. Defining interaction protocols using a commitment-based agent communication language. In *AAMAS*, 520–527.

Foster, H.; Uchitel, S.; Magee, J.; and Kramer, J. 2006. Model-based analysis of obligations in web service choreography. In *ICIW*.

Fu, X.; Bultan, T.; and Su, J. 2004. Conversation protocols: A formalism for specification and verification of reactive electronic services. *Theoret. Comp. Sci.* 328(1–2):19–37.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153(1-2):49–104.

Grosof, B. N., and Poon, T. C. 2004. SweetDeal: Representing agent contracts with exceptions using semantic web rules, ontologies, and process descriptions. *Intl. J. Electronic Commerce* 8(4):61–98.

MAS Lab. 2007. Action description examples in $\mathcal{C}+$. http://research.csc.ncsu.edu/mas/code/causal/.

McCarthy, J. 2003. Elaboration tolerance. http:// www-formal.stanford.edu/ jmc/ elaboration.html.

Minsky, N. H., and Ungureanu, V. 2000. Law-governed interaction. *ACM Trans. Soft. Engg. Method.* 9(3):273–305.

OWL-S. 2004. OWL-S: Semantic markup for web services. http://www.w3.org/Submission/OWL-S/.

Singh, M. P. 1999. An ontology for commitments in multiagent systems *Artificial Intelligence and Law* 7:97–113.

Tan, Y.-H., and Thoen, W. 1998. A logical model of directed obligations and permissions to support electronic contracting. *Intl. J. Electronic Commerce* 3(2):87–104.

van der Aalst, W., and van Hee, K. 2002. *Workflow Management Models, Methods, and Systems*. MIT Press.

Winikoff, M. 2007. Implementing commitment-based interaction. In *AAMAS*.