A Modular Action Description Language for Protocol Composition*

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

Abstract

Protocols are modular abstractions that capture patterns of interaction among agents. The compelling vision behind protocols is to enable creating customized interactions by refining and composing existing protocols. Realizing this vision presupposes (1) maintaining repositories of protocols and (2) refining and composing selected protocols. To this end, this paper synthesizes recent advances on protocols and on the knowledge representation of actions. This paper presents MAD-P, a modular action description language tailored for protocols. MAD-P enables building an aggregation hierarchy of protocols via composition. This paper demonstrates the value of such compositions via a simplified, but realistic, business scenario.

Introduction

Business protocols are widely used to enable superior engineering of business processes (Desai *et al.* 2005; Fu, Bultan, & Su 2004; Krazit 2002; Winikoff, Liu, & Harland 2005). Protocols characterize the interactions among business partners. For example, RosettaNet (1998) defines over 100 protocols for various aspects of e-business, e.g., purchase order processing. Traditionally, protocols are either not described formally or are described merely in terms of message order without regard to the meanings of the messages. It is now recognized that traditional approaches lead to rigid enactments and an equally rigid notion of compliance. Recent approaches describe the meanings of the messages in terms of the commitments of the participants (Desai *et al.* 2005; Winikoff, Liu, & Harland 2005). Such approaches give a formal basis involving reasoning about commitments, but do not address the fundamental knowledge engineering challenge of reuse and composition at the level of protocols.

Specifically, because protocols address different business goals, they often need to be composed to be put to good use. For example, a process for purchasing goods may involve protocols for ordering, shipping, and paying for goods. A commitment-based semantics enables such composition by clearly specifying the states of the interaction as they evolve through the messages exchanged by the participants. However, a commitment-based semantics by itself does not address how to specify the protocols so they can be reused, or how to specify how the protocols are to be used.

We would like to treat protocols as reusable components, potentially composed into additional protocols, and applied in a variety of business processes. By maintaining one or more repositories of commonly used, generic, and modular protocols, we can facilitate the reuse of a variety of well-defined, well-understood, validated, modular protocols. For example, a payment protocol can be used in a process for purchasing goods as well as in a process for registering for classes at a university. Further, the repository would expand as newly composed protocols are inserted into it.

How should protocols be formally represented so as to be included in a repository? How can existing protocols be composed in a manner that does not require hacking the protocols at a low level? In the spirit of Lifschitz and Ren's (2006) Modular Action Description (MAD) language, this paper addresses the above challenges by developing a language MAD-P that is geared toward protocols. Like MAD, MAD-P is layered over the causal logic $\mathcal{C}+$ (Giunchiglia *et al.* 2004).

Chopra and Singh (2006) show how to express protocols in $\mathcal{C}+$. MAD-P enhances Chopra and Singh's approach for representing individual protocols in $\mathcal{C}+$. More importantly, MAD-P includes axiom schemas by which desired protocol compositions can be specified. These schemas formalize some of the intuitive, but semantically informal constructs described by Desai *et al.* (2005). This paper shows how MAD-P schemas can be rewritten as $\mathcal{C}+$ axioms. Thus, protocols composed using MAD-P are also $\mathcal{C}+$ protocols.

Like MAD, MAD-P involves importing predefined modules (here protocols). However, the meaning and purpose of importing in the two approaches is quite different. Whereas MAD imports modules to be specialized and extended, MAD-P imports protocols to be aggregated. Thus, the importing protocol is defined as an aggregation of the imported protocols, and can be further imported by other protocols.

Below, we first present an ontology for protocols and a protocol specification in $\mathcal{C}+$ followed by the syntax and semantics of MAD-P. We illustrate the technique of protocol composition via MAD-P using a purchase process example.

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

Protocols in $\mathcal{C}+$

A *protocol* specifies a set of rules from a global viewpoint that govern the interaction among roles. These rules typically specify a *choreography*, i.e., constraints on the ordering of the messages to be exchanged in a protocol. The meaning of a message is given primarily by its effects on the agents' commitments. The general concepts relating to protocols are specified in $\mathcal{C}+$ as an ontology (Listing 1), to be included with specifications of individual protocols.

Messages are modeled as exogenous actions (line 9) reflecting the autonomy of the agents. Inertial fluents (line 8) record the history of all message occurrences (line 24). A static fluent initial ensures that the start state of a protocol is void of any fluents or commitments (lines 14, 20–22).

A commitment CC(ρ_1, ρ_2, p, q) denotes a directed obligation from role ρ_1 to ρ_2 to bring about q if p holds. Here, ρ_1 is called the debtor, ρ_2 the creditor, p the precondition and q the condition of the commitment. If the precondition p is T, the commitment is termed a *base* and otherwise a *conditional* commitment. Commitments are modeled as inertial fluents (line 11) and their preconditions and conditions are objects wrapped within action constants cond (line 10). Condition actions are disabled by default (line 26). Condition T denotes a condition that always holds (line 17).

Commitments can be created (create), discharged (discharge) , delegated, assigned, canceled, released. Additionally, when a conditional commitment's precondition is met, it yields a base commitment: this is treated as an action toBase. For simplicity, Listing 1 only describes create, discharge, and toBase (lines 12–13). Desai *et al.* present a more thorough treatment of commitments (2007).

Causing the conditions and preconditions of a commitment simultaneously causes appropriate operations: discharge and toBase, respectively, provided the commitment is active or being created simultaneously (lines 28–32). If a commitment is discharged it is deemed satisfied and ceases to hold (line 34). If toBase is caused, the original commitment ceases to exist, and a base level commitment is created provided the original commitment is not being discharged simultaneously (lines 36–39). A commitment is asserted if create is caused and that commitment is not being simultaneously discharged or converted to base, and the commitment does not already exist (lines 41–42). All commitment operations are disabled by default (lines 44–46). These laws collectively ensure correct behavior of commitment operations in the face of concurrent actions.

Listing 1: Protocols ontology in $\mathcal{C}+$ (ontology)

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

3  :- variables
4    msg :: Message;  p,q :: Condition;
5    cc :: Commitment;  db,cr :: Role.

7  :- constants
8    fl(Message) :: inertialFluent;
9    act(Message) :: exogenousAction;
10   cond(Condition) :: action;
11   comm(Commitment) :: inertialFluent;
12   create(Commitment), discharge(Commitment),
13     toBase(Commitment) :: action;
14   initial :: sdFluent.

16 :- objects
17   T :: Condition;
18   CC(Role,Role,Condition,Condition) :: Commitment.

20 caused initial if initial.
21 caused -initial if comm(cc).
22 caused -initial if fl(msg).

24 act(msg) causes fl(msg).

26 -cond(p) causes -cond(p).

28 caused discharge(CC(db,cr,p,q)) if cond(q) &
29 (comm(CC(db,cr,p,q)) ++ create(CC(db,cr,p,q))).

31 caused toBase(CC(db,cr,p,q)) if cond(p) &
32 (comm(CC(db,cr,p,q)) ++create(CC(db,cr,p,q))) & p<>T.

34 discharge(cc) causes -comm(cc).

36 toBase(cc) & -discharge(cc) causes -comm(cc).

38 toBase(CC(db,cr,p,q)) & -discharge(CC(db,cr,p,q))
39 causes comm(CC(db,cr,T,q)).

41 caused comm(cc) if true after create(cc) &
42 -(discharge(cc) ++ toBase(cc)) & -comm(cc).

44 -create(cc) causes -create(cc).
45 -toBase(cc) causes -toBase(cc).
46 -discharge(cc) causes -discharge(cc).
```

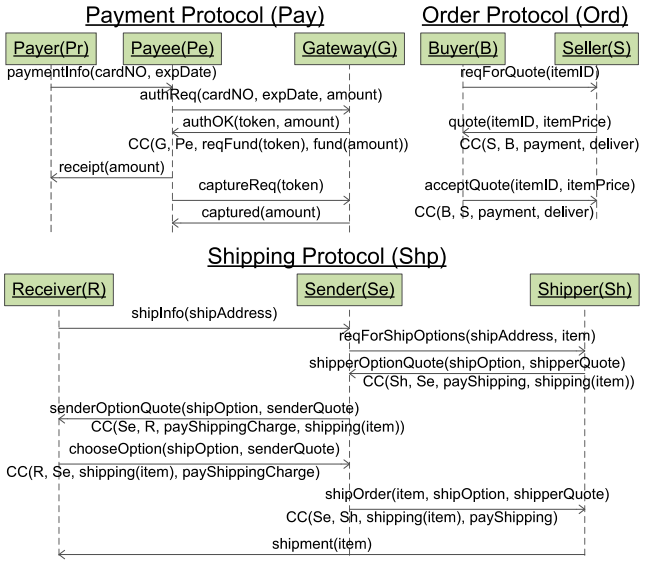

Figure 1: Scenarios from protocols *Ord*, *Shp*, and *Pay*

Now, let us consider a purchase process that is widely used in the business process modeling literature (e.g., (Desai *et al.* 2005)). A buyer and a seller interact to decide

the items to be purchased and the price. The seller then arranges for shipment of these items via a shipper. The customer pays via a payment gateway. Figure 1 shows how we disaggregate this process in terms of modular protocols. For simplicity, Figure 1 depicts only one scenario per protocol (although nontrivial protocols include multiple scenarios). Consider the *Ord* interaction by which the *Buyer* and *Seller* roles achieve a deal. The *Buyer* requests a quote for an item. The *Seller* responds by quoting a price for the item. The meaning of quoting a price is given by the creation of a commitment from the *Seller* to the *Buyer* that when the price is paid for, the goods would be delivered. The *Buyer* can then either accept or reject the quote. The rejection scenario is omitted here but is posted online (MAS Lab 2007). The meaning of acceptance is that the *Buyer* commits to paying for the delivered goods. Listing 2 describes *Ord*.

Listing 2: *Ord* protocol in $\mathcal{C}+$

```
1  :- protocol Ord.

3  :- include 'protocol-ontology'.

5  :- sorts
6    Slot >> ItemID;  Slot >> ItemPrice;
7    Role >> Buyer;  Role >> Seller.

9  :- objects
10   reqForQuote(Buyer,Seller,ItemID) :: Message;
11   quote(Seller,Buyer,ItemID,ItemPrice) :: Message;
12   accept(Buyer,Seller,ItemID,ItemPrice) :: Message;
13   deliver :: Condition;  payment :: Condition;
14   b :: Buyer;  s :: Seller;
15   myItem :: ItemID;  myPrice :: ItemPrice.

17 :- variables
18   itemID :: ItemID;  itemPrice :: ItemPrice.

20 nonexecutable act(reqForQuote(b,s,itemID)) if
21 fl(reqForQuote(b,s,itemID)).

23 nonexecutable act(quote(s,b,itemID,itemPrice)) if
24 -fl(reqForQuote(b,s,itemID)) ++
25 fl(quote(s,b,itemID,itemPrice)).

27 nonexecutable act(accept(b,s,itemID,itemPrice)) if
28 -fl(quote(s,b,itemID,itemPrice)) ++
29 fl(accept(b,s,itemID,itemPrice)).

31 act(quote(s,b,itemID,itemPrice)) causes
32 create(CC(s,b,payment,deliver)).

34 act(accept(b,s,itemID,itemPrice)) causes create(
35 CC(b,s,deliver,payment)).

37 :- query
38 label :: 5;  maxstep :: 4;  0: initial;
39 maxstep: fl(accept(b,s,itemID,itemPrice)).
```

The expression $m(\rho_s, \rho_r, v_1, \ldots, v_n)$ denotes a message of type m sent from ρ_s to ρ_r, and with contents v_i. Lines 20–29 of Listing 2 restrict sending the messages such that they respect the desired choreography, and are not repeated. Lines 31–35 model the contractual meanings of the quote and accept messages, respectively. As the conditions of the commitments are not caused within the protocol, their content slots are unknown and configurable via composition (next section). The query (lines 37–39) yields all paths (if any) from the initial state to a state where accept has already happened. Having messages, commitments, and conditions as objects enables wrapping of them within other fluents and actions.

At enactment, the roles would be adopted by agents who ground the protocols with their policies. For example, the protocol would specify that a quote be presented when a request for quotes is received. The specific price used in the quote message, however, is determined by the policy of the agent adopting the *Seller* role. Thus, in a protocol specification, a rule may have free variables in its head. Enactment is not discussed further in this paper.

Protocol Composition in MAD-P

Any realistic business process, e.g., purchase, would compose multiple protocols, e.g., order, shipping, and payment. Usually, each protocol is designed and maintained independently without assuming the existence of other protocols. As a result, protocols may use identical names for different concepts or distinct names for identical concepts. Therefore, MAD-P associates with each protocol a unique namespace to which all names of the protocol belong. MAD-P supports axiom schemas for stating which parameters of different protocols are identified, for example, that itemID in *Ord* and item in *Shp* refer to the same piece of information.

Whereas exogenous actions can happen freely and need not be caused explicitly, endogenous actions must be caused explicitly. Modeling messaging actions as exogenous has a significant impact on composition. Let us consider the protocols *P* and *Q* having exogenous actions whose "standard" models are shown in Figure 2 (these models are $\mathcal{C}+$ transition systems with null and self-loop transitions removed as in (Chopra & Singh 2006)).

If *S* is the union of *P* and *Q*, it should allow all possible interleavings of actions from *P* and *Q*, as shown in Figure 2 (far right). This is due to the fact that the only constraints specified by *P* and *Q* are that a must precede b and c must precede d. If no additional constraints are specified, *S* is the composite protocol constructed from *P* and *Q*.

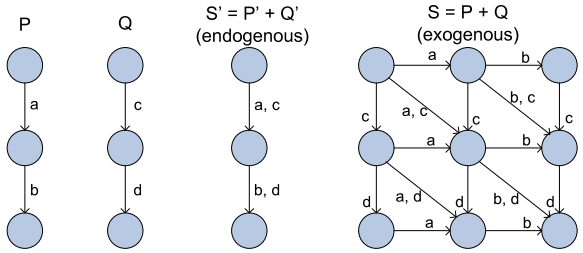

Figure 2: Exogenous vs. endogenous protocol actions

Let *P'* and *Q'* be protocols that have the same models as *P* and *Q*, respectively, but which involve endogenous instead of exogenous actions. Figure 2 shows that *S'*, the union of

P' and *Q'*, allows only one interleaving of the actions. This is due to the fact that for endogenous actions, their cause must be explicitly specified. If the specified cause is present, then the action must occur immediately. Thus, endogenous actions reduce the flexibility of agents by preventing some legitimate choices.

To meaningfully compose protocols, simply unioning their descriptions would generally not be enough. The desired interdependencies among the component protocols would need to be specified. For example, a slot of a protocol may get its value from a slot in another protocol, or the messages of different protocols may need to respect a temporal ordering. A protocol designer specifies *composition axioms* to capture such dependencies.

Let us illustrate protocol composition by composing a protocol *Pur* from the protocols *Shp*, *Ord*, and *Pay*. Listing 2 describes *Ord*. The specifications for *Shp*, *Pay*, and the composite protocol *Pur* are posted (MAS Lab 2007). Listing 3 describes *Pur* in terms of *Ord*, *Shp*, and *Pay*.

Listing 3: *Pur* protocol in MAD-P

```
1  :- protocol Pur.

3  :- import Ord; Shp; Pay.

5  :- role-identification
6  Pur.Customer is Ord.Customer, Shp.Receiver, Pay.Payer;
7  Pur.Merchant is Ord.Merchant, Shp.Sender, Pay.Payee;
8  Pur.Shipper is Shp.Shipper;
9  Pur.Gateway is Pay.Gateway.

11 :- data-flow
12 Shp.reqForshipOptions.Item uses Ord.accept.ItemID if ⊤;
13 Pay.authOK.Amount uses Ord.accept.ItemPrice if ⊤.

15 :- commitment-condition
16 Shp.shipment(item) means Ord.deliver(item);
17 Shp.authOK(·, amount) means Ord.payment(amount).

19 :- event-order
20 Pay.authOK before Shp.shipOrder.
```

Syntax and Semantics of MAD-P

MAD-P supports five composition axiom schemas, whose formal syntax is given by Equations (1), (2), (4), (5), (7), and (9) below along with applicable restrictions. The role identification axioms are mandatory in each composition; the other axioms are not. The semantics of MAD-P is given by rewriting each of the axioms in $\mathcal{C}+$. The $\mathcal{C}+$ axioms of Equations (3), (6), (8), and (10) are added to the theories as a result of rewriting the corresponding composition axiom schemas. *Pur* of Listing 3, rewritten in $\mathcal{C}+$, is posted (MAS Lab 2007). The procedure of rewriting is performed in sequential steps. Each step rewrites a set of composition axioms of a kind (e.g., role identification axioms) as $\mathcal{C}+$ axioms and as a result adds rules and other language elements to the theory of the composite protocol being constructed.

In the following, a protocol specification $\mathbb{P}$ is a tuple $\langle \rho_p, V_p, M_p, R_p, Cond_p, Comm_p \rangle$, where ρ_p is the set of participant roles in $\mathbb{P}$, V_p is the set of message slots in $\mathbb{P}$, M_p is the set of messages in $\mathbb{P}$, R_p is the set of causal rules of $\mathbb{P}$, $Cond_p$ is the set of commitment conditions, and $Comm_p$ is the set of commitments in $\mathbb{P}$. The formula ψ is an arbitrary logical expression over the constants of protocol ontology. Constants with trailing 'A' are actions whereas those with trailing 'F' are fluents.

The following rewriting process considers different axiom schemas in a specific order, which ensures that the rewriting of axioms does not create inconsistent names and axioms. The basic idea is

- Put the composed protocols into a "scratch" protocol by rewriting their roles based on the role identification axioms.
- Create additional axioms in the scratch protocol to capture the other composition axioms.
- Modify some of the other components in the scratch protocol.
- Return the newly minted scratch protocol as the result of the composition.

More than two protocols may be composed similarly.

Role Identification Axioms A *role identification* axiom defines a new role (named on the left) by identifying it with the roles named on the right (Lines 6–9). Thus, it specifies a constraint that an agent adopting the role on the left must adopt the roles on the right. Informally, the effect of these axioms is the substitution of the roles on the right by the role on the left in the commitments and messages of the imported protocols.

Formally, say two protocols $\mathbb{P}$=$\langle \rho_p, V_p, M_p, R_p, Cond_p, Comm_p \rangle$ and $\mathbb{Q}$=$\langle \rho_q, V_q, M_q, R_q, Cond_q, Comm_q \rangle$ having $|\rho_p|$ and $|\rho_q|$ roles, respectively, are composed to obtain $\mathbb{S}_{ri}$=$\langle \rho_{ri}, V_{ri}, M_{ri}, R_{ri}, Comm_{ri}, Cond_{ri} \rangle$, via n ($n \geq 2$) role definition axioms of the form:

$$\rho^i \text{ is } \rho_p^j,\ \rho_q^k \tag{1}$$

where, $0 \leq i < n$, $0 \leq j < |\rho_p|$, $0 \leq k < |\rho_q|$. Each ρ^i, ρ_p^j and ρ_q^k appears in exactly one such axiom.

Then, in the resulting composite protocol $\mathbb{S}_{ri}$, the set of roles ρ_{ri}=ρ, therefore $|\rho_{ri}|$=n. Also, V_{ri}=$V_p \cup V_q$ and R_{ri}=$R_p \cup R_q$. Where $(a/b)E$ substitutes the name b inside entity E by the name a, M_{ri}=$(\rho^i/\rho_p^j)M_p \cup (\rho^i/\rho_q^k)M_q$ and $Comm_{ri}$=$(\rho^i/\rho_p^j)Comm_p \cup (\rho^i/\rho_q^k)Comm_q$.

Data Flow Axioms Lines 12–13 specify the data flow from the slots on the right to the slots on the left provided the if condition holds. In general, a data flow axiom specifies that the value of a slot on the left must equal the value of the slot on the right during enactments of the composite protocol where the specified condition holds. Indirectly, data flow axioms impose a temporal order on the messages—a message that uses a slot cannot precede the message that provides the slot. Data flow composition is performed after the role definitions. Formally, data flow axioms are of the form:

$$m_q.v_q \text{ uses } m_p.v_p \text{ if } \psi \tag{2}$$

where $m_p \in M_p$, $m_q \in M_q$, and ψ can include constants from both protocols. Then, for the resulting composite protocol $\mathbb{S}_{df}$, v_q is renamed to v_p and a new rule r_{df}

$$\mathsf{nonexecutable}\ m_q A \ \mathsf{if}\ (-m_p A\ \&\ -m_p F)\ \texttt{++}\ -\psi \quad (3)$$

is added to R_{df}, meaning that m_q cannot happen unless m_p has already happened or is happening concurrently and ψ holds. Conditionalization via ψ enables alternative data flows for a slot depending on the protocol state. Thus, when ψ holds, v_q of m_q would be bound to v_p of m_p. Hence, R_{df}=$R_{ri} \cup \{r_{df}\}$ and V_{df}=$(v_p/v_q)V_{ri}$. The remaining entities are carried forward from the role definitions, i.e., E_{df}=E_{ri}, where E is ρ, M, $Comm$, or $Cond$.

Commitment Condition Axioms Lines 16–17 are *commitment condition* axioms defining the causation of commitment conditions across protocols. In the individual protocols, these conditions may not be caused locally and hence their precise meaning and parameters may not be known. For example, deliver is not caused in *Ord*. Commitment condition composition has two flavors: abstract and concrete. In the abstract flavor, the cause for the condition is itself a condition and the precise meaning and parameters of both conditions are unknown. It is known, however, that their meaning, whatever it may be, is identical. In the concrete flavor, the cause is an arbitrary formula that provides the meaning and parameters of the condition. Condition composition is performed after data flow composition, and abstract condition composition is performed before concrete condition composition. Lines 16–17 are concrete commitment condition axioms. An abstract condition axiom is of the form (where $cond_p \in Cond_p$ and $cond_q \in Cond_q$):

$$cond_p\ \mathsf{means}\ cond_q \quad (4)$$

As this axiom identifies the meaning of $cond_q$ with $cond_p$, $cond_q$ is simply renamed as $cond_p$ to effect this constraint. Such axioms are generally accompanied by the concrete flavor where the cause for $cond_p$ would be provided. Thus, for the resulting composite protocol $\mathbb{S}_{abs}$, $Cond_{abs} = (cond_p/cond_q)Cond_{rd}$. The remaining entities are carried forward from $\mathbb{S}_{df}$, i.e., E_{abs}=E_{df}, where E is ρ, V, M, R, or $Comm$. A concrete condition axiom is of the form:

$$\psi_p\ \mathsf{means}\ cond_q(v_1, v_2, \ldots, v_n) \quad (5)$$

where ψ_p is a formula of the atoms in $\mathbb{P}$, $cond_q \in Cond_q$, and each v_i is bound by the atoms in ψ_p, expressing that if ψ_p holds then the commitment condition $cond_q$ is met. Then, for the resulting composite protocol $\mathbb{S}_{conc}$, a rule r_{conc}

$$\mathsf{caused}\ cond_q(\ldots)\ \mathsf{if}\ \psi_p \quad (6)$$

is added. Thus, R_{conc}=$R_{abs} \cup \{r_{conc}\}$. The declaration of $cond_q$ is changed to have the slots v_i as parameters. The remaining entities are carried forward from $\mathbb{S}_{abs}$, i.e., E_{conc}=E_{abs}, where E is ρ, V, M, $Cond$, or $Comm$.

The above axiom schemas can be thought of as providing knowledge about various "counts as" relationships in the sense of Searle (1995). In essence, a commitment condition axiom states that a certain condition counts as another condition in the intended context of usage. For example, shipping the goods may count as satisfying a commitment of delivering the goods, even though these are not necessarily identical (some shipments can potentially be lost). Such operating procedures are common in business settings.

Commitment Operation Axioms These specify causation of commitment operations across protocols. An action or a state in a protocol may cause an operation on a commitment in another protocol. The purchase example does not exhibit this type of axiom, which is of the form:

$$\psi_p\ \mathsf{causes}\ op_i(comm_q) \quad (7)$$

where ψ_p is a formula of the atoms in $\mathbb{P}$, the commitment $comm_q \in Comm_q$. The operation op_i can be delegate, assign, release, or cancel. Such an axiom specifies that if ψ_p holds, then the operation op_i is performed on $comm_q$. Then, for the resulting composite protocol $\mathbb{S}_{op}$, a rule r_{op}

$$\mathsf{caused}\ op_i(comm_q)\ \mathsf{if}\ \psi_p \quad (8)$$

is added. Thus, R_{op} = $R_{conc} \cup \{r_{op}\}$. The remaining entities are carried forward from $\mathbb{S}_{conc}$, i.e., E_{op}=E_{conc}, where E is ρ, V, M, $Cond$, or $Comm$.

Event Order Axioms Line 20 is an *event order* axiom. It constrains the temporal ordering among the events by specifying that a payment authorization must be received before a shipping request is placed. Event order composition is performed after commitment operations. Event order axioms explicitly impose a temporal ordering regardless of other dependencies. Formally, an event order axiom is of the form:

$$m_p\ \mathsf{before}\ m_q \quad (9)$$

where $m_p \in M_p$ and $m_q \in M_q$. To effect the constraint, a new rule r_{eo}

$$\mathsf{nonexecutable}\ m_q A\ \mathsf{if}\ -m_p F \quad (10)$$

is added. Hence, for the resulting protocol $\mathbb{S}_{eo}$, R_{eo}=$R_{op} \cup \{r_{eo}\}$. The remaining entities are carried forward from $\mathbb{S}_{op}$, i.e., E_{eo}=E_{op}, where E is ρ, V, M, $Comm$, and $Cond$.

Listing 4 shows some of the causal axioms added to *Pur* as result of rewriting the axioms of Listing 3.

Listing 4: *Pur* rewritten in $\mathcal{C}+$

```
1  ...
2  nonexecutable act(reqForShipOptions(itemID)) if
3  -act(accept(itemID, itemPrice)) &
4  -fl(accept(itemID, itemPrice)).

6  nonexecutable act(authOK(token, itemPrice)) if
7  -act(accept(itemID, itemPrice)) &
8  -fl(accept(itemID, itemPrice)).

10 caused cond(deliver(itemID)) if act(shipment(itemID)).

12 caused cond(payment(itemPrice)) if
13 act(authOK(token, itemPrice)).

15 nonexecutable act(shipOrder(...)) if
16 -fl(authOK(token, itemPrice)).
```

In MAD-P, unlike in MAD, new axioms for the composite protocol need not be specified. Instead, the composite protocol is derived from the imported protocols. The composition axioms described in this paper express typical requirements in composing protocols. They act as configuration parameters reflecting a policy for composition. For special purposes, further axioms capturing additional "patterns of composition" might be readily constructed by combining features of the above. Composite protocols such as *Pur* can be treated and enacted like other protocols. Also, as the composite protocols can be rewritten as $\mathcal{C}+$ theories, they can be imported and composed further.

Given a correct specification of a protocol composition, the $\mathcal{C}+$ theory generated by the above would also be correct. The proof is by inspection that the intuitions underlying the composition axioms are met by the proposed translation. Placing composed protocols in $\mathcal{C}+$ opens up the possibility of verifying the specifications with respect to various desired properties. A useful future task is extending the causal calculator CCALC to handle MAD-P descriptions and rewrite them as $\mathcal{C}+$ descriptions.

Discussion

This paper synthesizes the intuitions previously developed in the studies of agent protocols and theories of action. To the area of agents, this paper contributes a rigorous approach for modeling and reusing protocols. To the area of reasoning about action, it contributes an application domain of considerable business value (Krazit 2002) as well as the general and important themes of reasoning about commitments and about communications.

A driving theme in our research program is to build a formally robust repository of protocols, whose components can be readily composed with others and to which new protocols can be contributed. This paper lays the groundwork for such a repository. It would be interesting to develop the notions of protocol refinement in the setting of such a repository.

Most existing approaches for composing protocols are based on low-level representations such as finite automata (Fu, Bultan, & Su 2004) or Petri Nets (Mazouzi, Seghrouchni, & Haddad 2002). Such formalisms do not support flexible execution and, because they are low level, do not support modeling in terms of high-level user requirements. AI-based approaches have a distinct advantage of flexibility in modeling and enactment. However, current such approaches are not sufficiently rigorously developed. Notable such approaches include (Desai *et al.* 2005; Winikoff, Liu, & Harland 2005), which lack a formal treatment of actions and constraints as developed here.

Vitteau and Huget (2004), driven by the motivation for reuse of interaction protocols, propose the notion of modular microprotocols to be composed with a Communication Protocol Description Language. Although formal, their composition is procedural and does not allow arbitrary interleaving of protocols like in MAD-P.

Previous work has studied procedural composition of dialogues. Reed (1998) discusses a typology of dialogues. He supports composing dialogues via nesting or functional embedding. McBurney and Parsons (2002) extend Reed's work and support composing dialogues via operators for sequential, parallel, and iterative execution. By contrast, the present paper composes protocols in a declarative manner wherein constraints are stated as composition axioms that capture the essential dependencies among protocols, and all executions are allowed that satisfy those axioms.

Whereas this paper emphasizes engineering business processes via reuse and composition of modular protocols, the work on dialogues emphasizes classifying dialogues broadly into a small number of categories, e.g., information seeking, persuasive, and so on. Its emphasis is on deliberation by agents. Only two-party interactions are studied and there is no support for interposing new roles, which is common in business settings, e.g., when a bank is introduced to achieve payment. It would be interesting to explore how dialogue typologies can help agents classify messages and enact different protocols.

References

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

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.

Desai, N.; Chopra, A. K.; and Singh, M. P. 2007. Representing and reasoning about commitments in business processes. In *AAAI*. To appear.

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.

Krazit, T. 2002. Intel conducts $5b in transactions via RosettaNet. http://tinyurl.com/344ah6.

Lifschitz, V., and Ren, W. 2006. A modular action description language. In *AAAI*, 853–859.

MAS Lab. 2007. MAD-P purchase examples in $\mathcal{C}+$. http://research.csc.ncsu.edu/mas/code/causal/.

Mazouzi, H.; Seghrouchni, A. E. F.; and Haddad, S. 2002. Open protocol design for complex interactions in multi-agent systems. In *AAMAS*, 517–526.

McBurney, P., and Parsons, S. 2002. Games that agents play: A formal framework for dialogues between autonomous agents. *J. Logic, Lang., & Info.* 11(3):315–334.

Reed, C. 1998. Dialogue frames in agent communications. In *ICMAS*, 246–253.

RosettaNet. 1998. Home page. www.rosettanet.org.

Searle, J. 1995. *Construction of Social Reality*. Free Press.

Vitteau, B., and Huget, M.-P. 2004. Modularity in interaction protocols. *Advances in Agent Communication*, *LNCS* 2922, 291–309.

Winikoff, M.; Liu, W.; and Harland, J. 2005. Enhancing commitment machines. Proc. *DALT Workshop*, *LNAI* 3476.