Tosca: Operationalizing Commitments Over Information Protocols

Thomas C. King[1] and **Akın Günay**[1] and **Amit K. Chopra**[1] and **Munindar P. Singh**[2]

[1]Lancaster University, Lancaster, LA1 4WA, United Kingdom
[2]North Carolina State University, Raleigh, NC 27695-8206, USA
{t.c.king, a.gunay, amit.chopra}@lancaster.ac.uk, singh@ncsu.edu

Abstract

The notion of *commitment* is widely studied as a high-level abstraction for modeling multiagent interaction. An important challenge is supporting flexible decentralized enactments of commitment specifications. In this paper, we combine recent advances on specifying commitments and *information protocols*. Specifically, we contribute Tosca, a technique for automatically synthesizing information protocols from commitment specifications. Our main result is that the synthesized protocols support *commitment alignment*, which is the idea that agents must make compatible inferences about their commitments despite decentralization.

1 Introduction

Commitments represent a high-level abstraction for modeling multiagent interaction [Singh, 1999]. The main idea behind commitment protocols is to specify the *meanings* of messages in terms of commitments [Pitt *et al.*, 2001; Yolum and Singh, 2002]. For example, to capture a purchase, one may specify that a *Quote* message means creating a commitment from the seller to the buyer to deliver an item in exchange for payment. In addition to meanings, a commitment protocol typically also specifies operational constraints such as message ordering and occurrence. Thus, for example, one would specify that the *Quote* message cannot occur before the *Request For Quote* message from the buyer to the seller. Intuitively, the motivation behind operational constraints is to rule out causally invalid protocol enactments.

A fundamental challenge in this line of work has been supporting *decentralized* enactments of commitment protocols, that is, in *shared nothing* settings where agents communicate *asynchronously*. Specifically, the only way for one agent to convey information to another is to send it a message. Supporting decentralized enactments in such settings is nontrivial because agents may observe messages in incompatible orders. Specifically, decentralization may lead to situations where agents deadlock (lack of *liveness*), observe inconsistent messages (lack of *safety*), or come to incompatible conclusions about commitments that hold between them (lack of *alignment* [Chopra and Singh, 2008; Chopra and Singh, 2009; Chopra and Singh, 2015b])—all three properties being crucial to interoperability.

Tosca addresses the challenge of decentralized enactments. It builds upon the conceptual observation that commitment specification and operational constraints are distinct concerns [Chopra and Singh, 2008; Baldoni *et al.*, 2013]. For simplicity and clarity, from here on, we reserve *protocol* to mean an operational protocol specifying messages and the operational constraints on their ordering and occurrence. Specifically, the question Tosca answers is: how can we operationalize commitment specifications over protocols such that liveness and safety are preserved, and alignment is guaranteed? Tosca's contribution is a method for automatically synthesizing the appropriate protocol.

Tosca's conceptual contribution is bringing three technical strands on interaction in multiagent systems together. One, BSPL [Singh, 2011], a declarative language for specifying protocols. BSPL protocols are known as *information protocols* because ordering and occurrence constraints fall out from more fundamental causality and integrity constraints on information in messages. A BSPL protocol can be checked for liveness and safety [Singh, 2012]. Two, Cupid [Chopra and Singh, 2015a], a declarative language for specifying commitments. The semantics of Cupid is in terms of commitment-oriented queries on a relational database. Thus we may imagine an agent that runs (for whatever purpose) commitment-oriented queries on its local database. Three, research on alignment [Chopra and Singh, 2015b] (C&S, for brevity), which is about mechanisms for ensuring that the parties to a commitment (the debtor and creditor) always progress toward states where they make mutually compatible local inferences about the commitment. Specifically, whenever the creditor infers a commitment as active from the messages it has observed, the debtor must as well infer it as active from its own observations.

Architecturally, Tosca brings the three strands together in the following manner. Each agent's local database or *state* comprises the messages it would have sent or received following a BSPL protocol. Cupid enables inferring the states of the commitments an agent is party to from this database. However, because each agent carries out this inference on its own local state, it may turn out that agents are not aligned with respect to a commitment. Tosca gives a method for ensuring progress toward alignment. Specifically, given a BSPL protocol and a set of commitments defined over the messages in the protocol, it gives a method for synthesizing a BSPL protocol whose enactment guarantees progress toward alignment. Furthermore, if the input protocol is live and safe, the synthesized protocol is live and safe as well.

Tosca goes beyond C&S in two ways. One, it addresses alignment for a more expressive language that includes deadlines, nested commitments, and a richer commitment lifecycle. Two, whereas C&S give algorithms for alignment, thereby constraining the implementation of agents, Tosca gives a purely interactive solution in terms of a protocol whose enactment would guarantee alignment.

2 Background

We now overview BSPL and Cupid, where for clarity we use *message* (as in BSPL) and *commitment* (as in Cupid) to mean instances, and *specification* to mean the respective specifications.

2.1 BSPL

BSPL is used to declaratively specify protocols without explicit control flow. By contrast, languages such as AUML [Huget and Odell, 2004] and RASA [Miller and Mcginnis, 2007] rely on explicitly specifying message ordering. Instead, BSPL protocols impose information causality constraints on each message *m*: what information *m*'s emission creates and what information the sending role must know before sending *m*. Thus, an implicit message ordering is imposed based solely on a protocol's explicit and declarative information causality specification.

Listing 1 demonstrates BSPL via the *Ordering* protocol. From here on, we describe such a protocol as an "input protocol" for Tosca, because it provides general message schemas for taking communicative actions (instantiating messages) and it is distinguished from a synthesized protocol for aligning a commitment.

The protocol *Ordering* has the roles *M* (merchant), *C* (customer), and *S* (shipper); and the parameters *oID* (order identifier), *item*, *price*, *pID* (pay identifier), *rID* (request identifier), and *sID* (ship identifier).

A *complete* enactment of *Ordering* comprises a tuple of bindings for all of its parameters. All parameters are adorned ⌜out⌝ for the protocol as a whole, meaning that their values are bound by enacting the protocol. Parameter *oID* is annotated as a key for the other parameters. This means each *oID* binding corresponds to a distinct tuple of bindings for non key parameters and thus identifies *Ordering*'s enactment. For example, it is not possible for the merchant to send two quotes with key binding *oID = 1* and different non key parameter bindings.

Ordering declares four message schemas (their placement is irrelevant). By convention, any key parameter of the protocol is a key parameter for any message in which it appears. The message schema *quote* on Line 4 is from the merchant to the customer. It has three parameters, whose values are bound by sending a quote due to being adorned ⌜out⌝, namely *oID*, *item*, and *price*.

The message schema *pay* on Line 5 is from the customer to the merchant. It comprises one parameter adorned ⌜in⌝, namely *oID*, which means that its value binding must be known via message emission or reception before a pay message is sent from the customer to the merchant. For example, the customer cannot send a *pay* message with ⌜in⌝ parameter binding *oID = 1* before receiving a *quote* message with the same binding. Hence, the customer can only send a *pay* message after receiving a *quote* from the merchant with the same key value, based on the information (parameter) causality constraints.

Likewise, the message schema *requestShip* on Line 6 is from the merchant to the shipper and it has the ⌜in⌝ parameter *oID*. Finally, *ship* on Line 7 has the *oID* parameter adorned ⌜in⌝. Since the shipper can only know about *oID*'s binding by receiving a *requestShip* message from the merchant, *ship* can only be sent after being requested.

Listing 1: A BSPL protocol for placing and fulfilling orders.

```
1 Ordering{
2  roles M, C, S // Merchant, Customer, Shipper
3  parameters out oID key, out item, out price,
       out pID, out rID, out sID
4  M ↦ C: quote[out oID, out item, out price]
5  C ↦ M: pay[in oID, out pID]
6  M ↦ S: requestShip[in oID, out rID]
7  S ↦ C: ship[in oID, out sID] }
```

2.2 Cupid

Cupid is a language for specifying commitments over an event database schema and inferring commitment states based on an event database state. In this paper, we only consider defining commitments over protocol message schemas, where commitments are inferred over messages.

We demonstrate Cupid's basic ideas with an example commitment in Listing 2 defined on top of the message schemas of Listing 1.

Listing 2: A specification in Cupid's surface syntax.

```
commitment Purchase M to C
 create quote
 detach pay[,quote + 10]
 discharge ship[,pay + 5]
```

A *Purchase* commitment from M (merchant) to C (customer) is *created* when a quote is made. The created commitment is uniquely identified by *quote*'s key (*oID*). *Purchase* is *detached* if a payment correlated to a quote occurs within ten time points of the quote (*pay* and *quote* both have the same key, *oID*). If the payment does not occur by the deadline, then the commitment is *expired* (failure to meet detach). The commitment is *discharged* if the (correlated) shipment occurs within five time points of the payment; if the shipment does not happen by the deadline, the commitment is *violated* (failure to meet discharge). Cupid treats such lifecycle events as first-class events, meaning that one commitment's lifecycle event may depend upon another's.

2.3 Separation of Concerns

Architecturally, Tosca uses BSPL to specify the *operational* layer interaction focusing on informational causality, as a separate concern from the interaction *requirements* specified in Cupid. For example, we could change Listing 1's *requestShip* message schema to be causally dependent on *pay*'s identifier (*pID*) before emission:

```
M ↦ S: requestShip[in oID, in pID, out
    rID]
```

The modified information causality does not alter the fact that the *Purchase* commitment continues to be discharged by a *ship* message within five time points of the *pay* message.

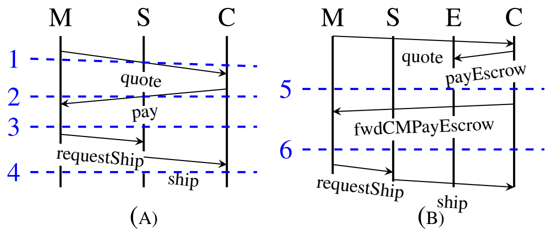

Figure 1: Protocol enactment for **M**erchant, **S**hipper, **E**scrow, and **C**ustomer roles.

The modification does alter when these messages *can* be sent. Conversely, changing the *Purchase* commitment in Listing 2's discharge from *ship* to another message would not affect if and when *ship* can be sent. Tosca's separation of concerns supports modularity: swapping out a protocol or modifying its information causality does not affect commitment level requirements, only when information (descriptively) *can* be created and thus when commitments can be met; changing a commitment does not affect if and when information can be created, only the (prescriptive) messaging *requirements* between parties.

3 Technical Motivation

We now demonstrate commitment operationalization protocols, which support realizing a commitment's lifecycle and progression towards alignment via messaging. Specifically, given a commitment defined over an input protocol that potentially causes misalignment, we synthesize a commitment alignment protocol as output. A commitment alignment protocol includes the necessary message schemas for forwarding messages to the creditor and debtor in order to guarantee commitment alignment. Together, an input protocol and multiple commitment alignment protocols are composed to form a *commitment operationalization protocol*.

3.1 Commitments Guaranteed Alignment

The *Purchase* commitment in Listing 2 is already alignable for the create, detach, discharge, and expired lifecycle events by the input protocol, *Ordering*, in Listing 1. In Figure 1 (A), at time point 1 after the merchant sends the customer a *quote* but before the customer receives it, the debtor (merchant) infers that the *Purchase* commitment is created. Hence the commitment is already aligned (the debtor knows that they are committed) regardless of what the creditor (customer) knows.

When the customer emits *pay* before time point 2 they infer that the *Purchase* commitment is *detached*. Hence, *Purchase* becomes misaligned, since the creditor (customer) has a stronger expectation of the debtor (merchant) to discharge the commitment, which the debtor does not know. The misalignment is rectified at time point 3, after the merchant receives the *pay* message.

Subsequently, after the merchant requests the shipper to ship, the shipper sends a *ship* message to the customer. At time point 4, after receiving the *ship* message, the creditor (customer) infers that the commitment is discharged and hence does not have a stronger expectation of the debtor (merchant) than what the debtor knows about (alignment).

Alignment for create, detach, discharge, and expired lifecycle events is guaranteed, either because misalignment does not occur (the creditor does not infer stronger expectations of the debtor) or misalignment is rectified via message reception. However, if ship is received after five time points of payment, then the creditor (customer) infers violation whereas the debtor (merchant) cannot (permanent misalignment). Such misalignment requires message forwarding, (e.g., notifying the debtor of ship), which we will cover in the next section.

3.2 Commitments Requiring Forwarding

Suppose an escrow service is used instead of direct payment from the customer to the merchant. The *EscrowOrdering* input protocol in Listing 3 and the *EscrowPurchase* commitment in Listing 4 capture this situation.

Listing 3: An input protocol providing messaging for placing and carrying out orders using an escrow service.

```
EscrowOrdering{
 roles E, M, C, S // Escrow, Merchant,
     Customer, Shipper
 parameters out oID key, out item, out price,
     out pID, out rID, out sID, out tID

 M ↦ C: quote[out oID, out item, out price]
 C ↦ E: payEscrow[in oID, out pID]
 M ↦ S: requestShip[in oID, out rID]
 S ↦ C: ship[in oID, out sID]
 E ↦ M: payTransfer[in oID, in pID, out
     tID] }
```

Listing 4: A commitment to capture escrow payment.

```
commitment EscrowPurchase M to C
 create quote
 detach payEscrow[, quote + 10]
 discharge ship[, payEscrow + 5]
```

In this scenario, we need to introduce a message that forwards another message's occurrence to an otherwise ignorant party. Listing 5 shows a protocol that introduces message forwarding in order to align the *EscrowPurchase* commitment (Listing 4) for the input protocol, *EscrowOrdering* in Listing 3. Specifically, by incorporating the message schema, *fwdCMPayEscrowID*, for *forwarding payEscrow* from the customer to the merchant. Each forwarding message schema has a distinct name mapped to the message being forwarded.

To exemplify, in Figure 1 (B) the customer sends *payEscrow* to the escrow. At time point 5 we have misalignment, because the customer (creditor) infers the *EscrowPurchase*'s detach and an expectation for the merchant to ship the goods. Yet the debtor (merchant) cannot know the customer's expectation without notification. Misalignment is resolved at time point 6 once the customer forwards *payEscrow* to the merchant via *fwdCMPayEscrow*. Both roles know that the debtor (merchant) is expected to discharge the commitment (alignment).

Listing 5: A protocol for aligning the *EscrowPurchase* commitment in Listing 4.

```
EscrowPurchaseAl{
 roles C, M
 parameters in oID key, in pID, out
     fwdCMPayEscrowID

 C ↦ M: fwdCMPayEscrow[in oID, in pID,
                    out fwdCMPayEscrowID] }
```

3.3 Nested Commitments

We now consider the case where one commitment's lifecycle event depends upon another's (nesting). The *EscrowTransfer* commitment given in Listing 6 is defined over the message schemas from the input protocol *EscrowOrdering* in Listing 3. The escrow service is committed to the merchant to transfer the customer's payment, once *EscrowPurchase* is discharged.

Listing 6: Escrow service's commitment to the merchant.

```
commitment EscrowTransfer E to M
 create payEscrow
 detach discharged(EscrowPurchase)
 discharge payTransfer[,
     discharged(EscrowPurchase) + 5]
```

EscrowTransfer is operationalized with the protocol in Listing 7. Focusing on the nested lifecycle event, the idea is that if *EscrowTransfer*'s creditor (merchant) infers its detach, then so should the debtor (escrow). *EscrowTransfer*'s detach is *EscrowPurchase*'s discharge. Hence we ensure that whenever a message contributes to *EscrowPurchase*'s discharge it can be forwarded to *EscrowTransfer*'s debtor (escrow).

Listing 7: A protocol for aligning the *EscrowTransfer* commitment in Listing 6.

```
EscrowTransferAl{
  roles C, E, M //Customer, Escrow, Merchant
  parameters in oID key, in item, in price,
      in pID, in sID, out fwdMEQuoteID, out
      fwdCMPayEscrowID, out fwdSEShipID,
      out fwdMEShipID

  M ↦ E: fwdMEQuote[in oID, in item, in
      price, out fwdMEQuoteID]
  C ↦ M: fwdCMPayEscrow[in oID, in pID, out
      fwdCMPayEscrowID]
  S ↦ E: fwdSEShip[in oID, in sID, out
      fwdSEShipID] }
  M ↦ E: fwdMEShip[in oID, in sID, out
      fwdMEShipID] }
```

In Figure 2 at time point 7, the merchant infers *EscrowPurchase*'s discharge and consequently *EscrowTransfer*'s detach. Specifically, due to knowing about the messages contributing to *EscrowPurchase*'s discharge: *quote* (by sending it), *payEscrow* (via the forwarding message *fwdCMPayEscrow*), and *ship* (via the forwarding message *fwdSMShip*). Yet the debtor (escrow) neither infers *EscrowPurchase*'s discharge nor consequently *EscrowTransfer*'s detach. Hence, the creditor (merchant) expects the debtor (escrow) to discharge *EscrowTransfer*, which the debtor is unaware of (misalignment).

Forwarding message schemas to the escrow support rectifying the misalignment. *EscrowPurchase*'s discharge is due to: *quote*, which can be forwarded to the escrow; *payEscrow*, which the escrow receives (hence no forwarding is required) and *ship*, which can be forwarded to the escrow. In Figure 1 at time point 8, after sending the forwarding messages, escrow knows about *EscrowPurchase*'s discharge and thus *EscrowTransfer*'s detach (alignment).

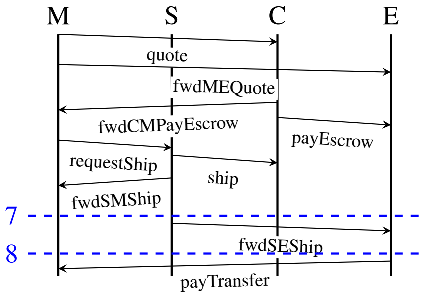

Figure 2: Protocol enactment for **M**erchant, **S**hipper, **E**scrow, and **C**ustomer roles where each message shares key values.

3.4 Compositionality

Commitment alignment protocols can be synthesized independently and then composed. In Listing 8, our preceding commitment alignment protocols are subprotocols of an overall operationalization protocol. If two commitment alignment protocols bind the same ⌜out⌝ parameter, it is due to the same message being sent and hence the binding is the same. For example *fwdCMPayEscrowID* is bound by *EscrowPurchaseAl* when a forward message *fwdCMPayEscrow* is sent if and only if *fwdCMPayEscrowID* is bound with the same value by *EscrowTransferAl* when the same *fwdCMPayEscrow* message is sent. Hence, independently constructed alignment protocols are composed together without contradictory parameter bindings during enactment.

Listing 8: An operationalization protocol composed from an input protocol and synthesized alignment protocols.

```
OperationalizationProtocol{
 roles M, C, E, S
 parameters out oID key, out item, out price,
     out pID, out sID, out rID, out tID,
     out fwdMEQuoteID, out fwdCMPayEscrowI,
     out fwdSEShipID, out fwdMEShipID

 EscrowOrdering(M,
     C, E, S, out oID, out item, out price,
     out pID, out sID, out rID, out tID)
 EscrowPurchaseAl(E, M, C, S,
     in oID, in pID, out fwdCMPayEscrowID)
 EscrowTransferAl(C, E, M, S, in oID,
     in item, in price, in pID, in sID, out
     fwdMEQuoteID, out fwdCMPayEscrowID,
     out fwdSEShipID, out fwdMEShipID) }
```

3.5 Summary

Tosca synthesizes the alignment protocol for a commitment and an input protocol. The alignment protocol comprises forwarding message schemas, supporting participants in aligning the commitment via messaging. Multiple commitment alignment protocols are composed together, without parameter interference, into an operationalization protocol for triggering commitment lifecycles and supporting alignment via messaging.

Table 1: Cupid's grammar. Expr is create, detach, and discharge conditions.

Event	$\longrightarrow$	Base | LifeEvent
LifeEvent	$\longrightarrow$	created($\mathcal{R}$, $\mathcal{R}$, Expr, Expr, Expr) | detached($\mathcal{R}$, $\mathcal{R}$, Expr, Expr, Expr) | discharged($\mathcal{R}$, $\mathcal{R}$, Expr, Expr, Expr) | expired($\mathcal{R}$, $\mathcal{R}$, Expr, Expr, Expr) | violated($\mathcal{R}$, $\mathcal{R}$, Expr, Expr, Expr)
Expr	$\longrightarrow$	Event[Time, Time] | Expr $\sqcap$Expr|Expr $\sqcup$Expr| Expr $\ominus$ Expr
Time	$\longrightarrow$	Event + $\mathcal{T}$ | $\mathcal{T}$
ComSpec	$\longrightarrow$	c($\mathcal{R}$, $\mathcal{R}$, Expr, Expr, Expr)

4 Synthesizing Protocols

4.1 Protocols

We adopt BSPL's formal syntax from [Singh, 2012]. We use the following lists treated as sets: public roles $\vec{x}$, private roles $\vec{y}$, public parameters $\vec{p}$, $\ulcorner\mathsf{key}\urcorner$ parameters $\vec{k} \subseteq \vec{p}$, $\ulcorner\mathsf{in}\urcorner$ parameters $\vec{p}_I \subseteq \vec{p}$, $\ulcorner\mathsf{out}\urcorner$ parameters $\vec{p}_O \subseteq \vec{p}$, private parameters $\vec{q}$, and parameter bindings $\vec{v}$ and $\vec{w}$. The set of all parameters is $\vec{p} = \vec{p}_I \cup \vec{p}_O$. The $\ulcorner\mathsf{in}\urcorner$ and $\ulcorner\mathsf{out}\urcorner$ parameters are mutually disjoint: $\vec{p}_I \cap \vec{p}_O = \emptyset$. A protocol's references (i.e., subprotocols, including message schemas) are denoted by the set F.

Definition 1. A *protocol* is a tuple $P = \langle n, \vec{x}, \vec{y}, \vec{p}, \vec{k}, \vec{q}, F \rangle$ where n is a name. $\vec{x}$,$\vec{y}$,$\vec{p}$,$\vec{q}$ are as above, F is a finite set of f subprotocol references $F = \{F_1, \dots, F_f\}$, such that P includes each referenced sub protocol F_i's roles, and key and non key parameters $(\forall i : 1 \le i \le f \Rightarrow F_i = \langle n_i, \vec{x}_i, \vec{p}_i, \vec{k}_i \rangle)$ where $\vec{x}_i \subseteq \vec{x} \cup \vec{y}$, $\vec{p}_i \subseteq \vec{p} \cup \vec{q}$, $\vec{k}_i = \vec{p}_i \cap \vec{k}$). An atomic protocol with two roles and no references is a message schema denoted as $\lceil s \mapsto r : m\,\vec{p}(\vec{k}) \rceil$.

Later, protocol enactment is defined over a Universe of Discourse (UoD) comprising roles and message schemas (for convenience we modify the original BSPL definition from a UoD comprising message *references*).

Definition 2. [Singh, 2012, Def. 12] The *UoD* of protocol P, $\mathrm{UoD}(P) = \langle \mathcal{R}, \mathcal{M} \rangle$ consists of P's roles and message schemas including the message schemas of its referenced protocols recursively.

4.2 Commitments

A *commitment specification* is a finite string according to the corresponding representation in Table 1 over a message schema name set Base.

A commitment specification, $c(x, y, Cre, Det, Dis)$ from a debtor x to a creditor y is defined over BSPL protocol message schema references (Base events) used by an input protocol. Role names and time instants are sets $\mathcal{R}$ and $\mathcal{T}$, respectively. Operators $\sqcap$, $\sqcup$, and $\ominus$ are respectively conjunction, disjunction, and exception. In $E[l,r]$, $[l,r]$ is the time interval that $E[l,r]$ occurs within. We omit l and r when they are respectively 0 and ∞.

4.3 Commitment Operationalization

Each commitment we wish to operationalize is rewritten into a commitment alignment protocol for an input protocol. A forwarding message schema has a unique name and $\ulcorner\mathsf{out}\urcorner$ parameter, to avoid conflicts with other message schemas.

Let $\mathcal{N}$ be the set of unique message forwarding schema names disjoint from the message schema name set Base. Unique forwarding message schema names are obtained taking as input a message schema name from Base and a role, and then outputting a forwarding message schema name, using an assumed injective function $V : \mathrm{Base} \times \mathcal{R} \to \mathcal{N}$. For example, $fwdSEShip = V(ship, E)$ is a unique name for forwarding *ship* from the shipper (S) to the escrow (E). The inverse injective function V^{-1} determines which message is being forwarded. An assumed injective parameter name function $VID : \mathcal{N} \to \mathcal{ID}$ from forwarding message names to identifier parameter names ($\mathcal{ID}$) produces a uniquely named $\ulcorner\mathsf{out}\urcorner$ parameter for each forwarding message schema (e.g., $fwdSEShipID = VID(fwdSEShip)$).

A forwarding message schema is included in a commitment operationalization protocol when necessary to support commitment alignment. For example, if E is the commitment's create condition, b the debtor and a the creditor. Then, the event formula E must be aligned between roles a and b such that if a knows E then b can learn of E via message forwarding.

Each commitment C is decomposed via rewrites into instructions of the form $al(a, E, b)$ stating a requirement for E to be *al*igned from a to b. If the formula E is non atomic, then it is further decomposed to eventually atomic alignment instructions, $al(a, m, b)$ on messages m. Atomic message alignment instructions are rewritten into the necessary message forwarding schema in the alignment protocol that we are constructing. We present the base case rewrites for alignment instructions operating on message schemas, then non atomic event formulae and finally commitments.

The presented rewrite rules are for an input protocol $P = \langle n, \vec{x}, \vec{y}, \vec{p}, \vec{k}, \vec{q}, F \rangle$ and its Universe of Discourse $\langle \mathcal{R}, \mathcal{M} \rangle = \mathrm{UoD}(P)$, a commitment C and the commitment alignment protocol $P^C = \langle n^C, \vec{x}^C, \vec{y}^C, \vec{p}^C, \vec{k}^C, \vec{q}^C, F^C \rangle$ being constructed.

Rule R1 handles aligning an atomic message. It is conditional on: (1) An atomic message alignment instruction $al(a,m,b)$. (2) The commitment alignment protocol P^C being constructed. (3) A message schema in the input protocol P where a role s that is distinct from b instantiates the message m via emission to a role r distinct from b.

The rewrite result is: (4) A new message schema labeled m^{forw} acting to forward message schema m's instances, from the role s to the role b. (5) The commitment alignment protocol referencing m^{forw}. (6) The commitment alignment protocol including the forwarding message schema's $\ulcorner\mathsf{key}\urcorner$ parameters, (7) $\ulcorner\mathsf{in}\urcorner$ and $\ulcorner\mathsf{out}\urcorner$ parameters, and roles.

$$\frac{\begin{array}{cr} al(a,m,b), & (1) \\ P^C = \langle n^C, \vec{x}^C, \vec{y}^C, \vec{p}^C, \vec{k}^C, \vec{q}^C, F^C \rangle, & (2) \\ \lceil s \mapsto r : m\,\vec{p}(\vec{k}) \rceil \in \mathcal{M},\ s \neq b, r \neq b & (3) \end{array}}{\begin{array}{cr} \lceil s \mapsto b : m^{forw}\,\vec{p}^{forw}(\vec{k}^{forw}) \rceil & (4) \\ F^C = F^C \cup \{m^{forw}\}, & (5) \\ \vec{k}^C = \vec{k}^C \cup \vec{k}, & (6) \\ p_I^C = p_I^C \cup p_I^{forw},\ p_O^C = p_O^C \cup p_O^{forw}, \vec{x}^C = \vec{x} \cup \vec{x}^C & (7) \end{array}} \quad \text{(R1)}$$

Where:

- The uniquely named forwarding message schema forwards m to b: $m^{forw} = V(m,b)$.

- The forwarding message schema's parameters comprise: keys corresponding to m's ($k^{forw} = k$); a unique $\ulcorner\mathsf{out}\urcorner$ parameter to ensure that protocol enactment requires forwarding ($p_O^{forw} = \{VID(m^{forw})\}$); and $\ulcorner\mathsf{in}\urcorner$ parameters matching m's parameters ($p_I^{forw} = p$), meaning that m must be instantiated prior to it being forwarded.

Instructions to align messages from a to b occurring within a time window are reduced to atomic message alignment instructions. The message that occurs as well as any start or deadline messages are necessarily also aligned from a to b according to the rewrite Rule R4 (omitting cases for time windows without either a start time, a deadline, or both).

$$\frac{al(a,m[s\pm J,d\pm K],b)}{al(a,m,b)\ al(a,s,b)\ al(a,d,b)} \tag{R4}$$

To give an example, the *EscrowPurchase* commitment from the merchant to the customer in Listing 4 is detached when the customer pays the escrow within ten time points of a price quote. Hence we have an alignment instruction $al(C, payEscrow[, quote + 10], M)$ to ensure that when the creditor (customer) knows the detachment so does the debtor (merchant). The alignment instruction is reduced to $al(C,payEscrow,M)$ and $al(C,quote,M)$.

In the input protocol in Listing 3 quote is from the merchant to the customer, guaranteeing alignment, and so $al(C,quote,M)$ is not rewritten. However, $\mathsf{payEscrow}$ is not received or sent by the merchant and hence must be forwarded by the customer. The instruction $al(C,payEscrow,M)$ is rewritten to a message schema as in Listing 9.

Listing 9: A partial alignment protocol for the *EscrowPurchase* commitment in Listing 4

```
EscrowPurchaseAl{
roles C, M
parameters in oID key, in pID, out
    fwdCMPayEscrowID
C ↦ M: fwdCMPayEscrow[in oID, in pID, out
    fwdCMPayEscrowID] }
```

Both sides of a conjunct are aligned according to Rule R5. Likewise both sides of a disjunct are aligned via Rule R6, since we do not know which runtime messages will occur *a priori*.

Exceptions are handled by Rule R7. In order to guarantee that when a role a knows $L\ominus R$ then so does b, we must ensure that if a knows L then so can b via messaging. Yet, if b knows R then it will never know $L\ominus R$ to be true, even if a knows L, believes $L\ominus R$ is true and so forwards L to b. Thus we align L from a to b and R from b to a.

$$\frac{al(a,L\sqcap R,b)}{al(a,L,b)\ al(a,R,b)}\ \text{(R5)} \qquad \frac{al(a,L\sqcup R,b)}{al(a,L,b)\ al(a,R,b)}\ \text{(R6)}$$

$$\frac{al(a,L\ominus R,b)}{al(a,L,b)\ al(b,R,a)} \tag{R7}$$

For example, a shipment commitment from the shipper to the merchant is discharged if: the item is shipped within five time points of the shipment being requested, except if the shipment is reported as damaged within five time points of being received. Thus we have an alignment instruction from the shipper to the merchant: $al(S, ship[, requestShip + 5] \ominus reportDamage[, ship + 5], M)$. Alignment holds when: if the shipper knows $ship[, requestShip + 5]$ then so does the merchant and if the merchant knows the exception $reportDamage[,ship+5]$ then so does the shipper. Hence the alignment instruction is rewritten to $al(S,ship[requestShip+5],M)$ and $al(M,reportDamage,S)$.

Nested commitment lifecycle events occur when the corresponding lifecycle event for the referenced commitment $c(x,y,Cre,Det,Dis)$ occurs. Hence, we rewrite nested lifecycle events to the conditions under which they occur according to Cupid's semantics with Rules R8 to R12.

$$\frac{al(a,created(x,y,Cre,Det,Dis),b)}{al(a,Cre,b)} \tag{R8}$$

$$\frac{al(a,detached(x,y,Cre,Det,Dis),b)}{al(a,Cre\sqcap Det,b)} \tag{R9}$$

$$\frac{al(a,discharged(x,y,Cre,Det,Dis),b)}{al(a,(Cre\sqcap Dis)\sqcup(Det\sqcap Dis),b)} \tag{R10}$$

$$\frac{al(a,expired(x,y,Cre,Det,Dis),b)}{al(a,Cre\ominus Det,b)} \tag{R11}$$

$$\frac{al(a,violated(x,y,Cre,Det,Dis),b)}{al(a,(Cre\sqcap Det)\ominus Disch,b)} \tag{R12}$$

The final rewrite is for commitments. A commitment is aligned when: if the creditor knows it is created, detached, or violated, then the debtor respectively knows it is created, detached, or violated; and if the debtor knows it is discharged or expired, then the creditor knows it is respectively discharged or expired. Owing to this asymmetry, we rewrite a commitment with Rule R13 to alignment instructions for each lifecycle event.

$$\frac{c(x,y,Cre,Det,Dis)}{\begin{array}{c} al(c,created(x,y,Cre,Det,Dis),d) \\ al(c,detached(x,y,Cre,Det,Dis),d) \\ al(c,violated(x,y,Cre,Det,Dis),d) \\ al(d,discharged(x,y,Cre,Det,Dis),c) \\ al(d,expired(x,y,Cre,Det,Dis),c) \end{array}} \tag{R13}$$

We now define a commitment alignment protocol.

Definition 3. A protocol P^C is a *commitment alignment protocol* for a commitment C and an input protocol P iff all possible applications of R1 to R13 are made to C, an empty version of P^C and P's UoD $\langle\mathcal{R},\mathcal{M}\rangle = \text{UoD}(P)$.

Each rule is a monotonic reduction on finite formulae. Hence:

Lemma 1. There exists a commitment operationalization protocol P^C for each commitment C and input protocol P.

An operationalization protocol is composed from the input protocol and commitment alignment protocols (omitting empty and redundant subprotocols).

Definition 4. Let $P = \langle n,\vec{x},\vec{y},\vec{p},\vec{k},\vec{q},F\rangle$ be an input protocol and $\mathbb{C}$ be a set of commitments defined over the message schema names and roles in P's Universe of Discourse $\langle\mathcal{R},\mathcal{M}\rangle =$

UoD(P). Let each commitment $C \in \mathbb{C}$ have an alignment protocol $P^C = \langle n^C, \vec{x}^C, \vec{y}^C, \vec{p}^C, \vec{k}^C, \vec{q}^C, F^C \rangle$ for P that includes at least two roles ($|\vec{x}^c| \geq 2$). $P^{\mathbb{C}} = \langle n^{\mathbb{C}}, \vec{x}^{\mathbb{C}}, \vec{y}^{\mathbb{C}}, \vec{p}^{\mathbb{C}}, \vec{k}^{\mathbb{C}}, \vec{q}^{\mathbb{C}}, F^{\mathbb{C}} \rangle$ is an operationalization protocol for $\mathbb{C}$ and P iff:

$P^{\mathbb{C}}$ references the input protocol and all commitment operationalization protocols: $F^{\mathbb{C}} = \{n\} \cup \bigcup_{C \in \mathbb{C}} \{n^C\}$.

$P^{\mathbb{C}}$'s roles and key parameters match the input protocol's: $\vec{x}^{\mathbb{C}} = \vec{x}$ and $\vec{k}^{\mathbb{C}} = \vec{k}$.

P's ⌜out⌝ parameters comprise the input protocol's and each commitment alignment protocol's ⌜out⌝ parameters: $\vec{p}_O^{\mathbb{C}} = \{\vec{p}_O\} \cup \bigcup_{C \in \mathbb{C}} \{\vec{p}_O^C\}$.

4.4 Semantics

In BSPL, each message instance $m[s, r, \vec{p}, \vec{v}]$ denotes a sending role s, a recipient r, a parameter vector $\vec{p}$ with a corresponding parameter binding value vector $\vec{v}$. A role's history denotes its sent and received messages in sequence. A history vector comprises each role's history where every received message must have been sent.

Definition 5. [Singh, 2012, Def. 5] A history of a role ρ, H_ρ, is given by a sequence of zero or more message instances $m_1 \circ m_2 \circ \ldots$. Each m_i is of the form $m[s, r, \vec{p}, \vec{v}]$ where $\rho = s$ or $\rho = r$, and $\circ$ means sequencing.

Definition 6. [Singh, 2012, Def. 7] We define a *history vector* for a UoD $\mathcal{R}, \mathcal{M}$, as $[H^1, \ldots, H^{|\mathcal{R}|}]$, such that $\forall s, r : 1 \leq s, r \leq |\mathcal{R}|$: H^s is a history s.t. $\forall m[s, r, \vec{p}, \vec{v}] \in H^r : m \in \mathcal{M}$ and $m[s, r, \vec{p}, \vec{v}] \in H^s$.

A history vector is viable if and only if sent and received messages bind values to parameters specified in each corresponding message schema, respecting values already determined by keys via known messages (for brevity, we omit Singh's [2012, Def. 8] definition). The set of all viable history vectors for a UoD (e.g., a protocol's roles and message schemas) is its Universe of Enactments.

Definition 7. Given a UoD $\langle \mathcal{R}, \mathcal{M} \rangle$, the *Universe of Enactments* (UoE) for that UoD, $\mathcal{U}_{\mathcal{R},\mathcal{M}}$, is the set of viable history vectors [2012, Definition 8], each of which has exactly $|\mathcal{R}|$ dimensions and each of whose messages instantiates a schema in $\mathcal{M}$.

In Cupid [Chopra and Singh, 2015a], an agent's *model* maps from event schemas to event instances, representing the agent's local view of event occurrences. Here, we are dealing with messages and hence a model is simply a role's history albeit substituting each forwarding message with the message it forwards and timestamping each message with the time it became known (via emission, reception, or notification).

Definition 8. Let P be an input protocol and let Base be the message schema names for the message schemas in P's UoD. Let $P^{\mathbb{C}}$ be an operationalization protocol for P. A *model* for a role a's history H and $P^{\mathbb{C}}$ is a history M, where each message is in the model $m_i^M[s_i^M, r_i^M, \vec{p}^M, \vec{v}^M] \in M$ iff there is a corresponding original message $m_i^H[s_i^H, r_i^H, \vec{p}^H, \vec{v}^H] \in H$ meeting (a) **or** (b), **and** (c):

(a) The corresponding original message in H is a non forwarding message $m_i^H \in$ Base and the names match: $m_i^M = m_i^H$.

(b) The corresponding original message m_i^H in H is a forwarding message and the message m_i^M in the model takes the name of the message being forwarded: $m_i^M = V^{-1}(m_i^H, r_i^H)$.

(c) The message m_i^M contains all of the original message m_i^H in H's non forwarding ID parameters and parameter bindings with an additional timestamp parameter and parameter binding: if $\exists \vec{p}_j^H \in \vec{p}^H = VID^{-1}(m_i^H)$ then $\vec{p}_i^M = (\vec{p}_i^H \backslash \vec{p}_j^H) \cup \{\text{time}\}$ and $\vec{v}_i^M = (\vec{v}_i^H \backslash \vec{v}_j^H) \cup \{\text{t}\}$, otherwise $\vec{p}_i^M = \vec{p}_j^H \cup \{\text{time}\}$ and $\vec{v}_i^M = \vec{v}_j^H \cup \{\text{t}\}$, where t is a timestamp.

We adopt Cupid's commitment semantics. For brevity, we only define the set of instances for event formula E (e.g., a lifecycle event) entailed by a model M: $[\![E]\!]_M$. If E is a non atomic event formula or a lifecycle event, then the result is a database operation on the messages that cause E to occur. For example, the set of all ship instances is denoted as $[\![ship]\!]$. Moreover, the set of tuples modeled by the formulae for shipment occurring within ten time points, $ship[0, 10]$, is returned by selecting all shipment events that occur between zero and ten time points ($[\![ship[0,10]]\!] = \sigma_{0 \leq t < 10}([\![ship]\!])$). A database operation is inductively defined in Cupid for queries corresponding to each event formula type.

Definition 9. Let M be a model and E an event formula. $[\![E]\!]_M$ is the set of E's instances (a set of tuples combining stored events) returned from the query for E on M according to [Chopra and Singh, 2015a, $D_1 - D_{20}$].

5 Properties

An operationalization protocol retains an input protocol's message ordering.

Lemma 2. Let P be an input protocol, and $P^{\mathbb{C}}$ be a commitment operationalization protocol. If there is a history vector H in $UoD(P)$'s UoE then there exists a history vector $H^{\mathbb{C}}$ in $UoD(P^{\mathbb{C}})$'s UoE where for each history $H^{i\mathbb{C}}$ in $H^{\mathbb{C}}$ the messages instantiating schemas in $\mathcal{M}$ are included in the same order of the corresponding history H^i in H.

Proof sketch. We include messages from the input protocol's history to the operationalization protocol's corresponding history, if they instantiate a schema in the input protocol. We can always include received messages ([2012, Def. 6]), emitted messages can be included since they do not violate bindings and ⌜in⌝ parameters are necessarily known by binding parameters in the operationalization protocol's message schemas. □

Singh [2012] formalizes liveness and safety for BSPL, and gives verification techniques. A protocol is *safe* iff each history vector in the UoE preserves uniqueness for each binding. A protocol is *live* iff any enactment can progress to completion such that all parameters are bound.

Theorem 1. Let $P^{\mathbb{C}}$ be a commitment operationalization protocol for an input protocol P. If P is safe then $P^{\mathbb{C}}$ is safe. If P is live then $P^{\mathbb{C}}$ is live.

Proof sketch. Safety: Assume $P^{\mathbb{C}}$ is unsafe. Since $P^{\mathbb{C}}$ is an operationalization protocol, by Rule R1 (7) it does not introduce $\ulcorner\texttt{out}\urcorner$ parameters used by another message schema, hence by applying Lemma 2 P must be unsafe as well. Liveness: As for safety, relying on $P^{\mathbb{C}}$ not introducing $\ulcorner\texttt{out}\urcorner$ parameters used by P. □

C&S define alignment for active commitments. Definition 10 generalizes it to all states in Cupid's commitment lifecycle. Specifically, if a creditor infers created, detached or violated of a commitment (thereby strengthening the expectation) from its history, then the debtor must as well (i.e., know what is expected of them). Conversely, if a debtor infers discharge or expired (hence weakening the expectation), the creditor must as well.

Definition 10. Let M^x and M^y be models. The history vector H is aligned with respect to $c(x,y,C,D,U)$ iff:

$$i \in [\![created(x,y,C,D,U)]\!]_{M^y} \Rightarrow i \in [\![created(x,y,C,D,U)]\!]_{M^x}$$
$$i \in [\![detached(x,y,C,D,U)]\!]_{M^y} \Rightarrow i \in [\![detached(x,y,C,D,U)]\!]_{M^x}$$
$$i \in [\![violated(x,y,C,D,U)]\!]_{M^y} \Rightarrow i \in [\![violated(x,y,C,D,U)]\!]_{M^x}$$
$$i \in [\![discharged(x,y,C,D,U))]\!]_{M^x} \Rightarrow i \in [\![discharged(x,y,C,D,U)]\!]_{M^y}$$
$$i \in [\![expired(x,y,C,D,U)]\!]_{M^x} \Rightarrow i \in [\![expired(x,y,C,D,U)]\!]_{M^y}$$

The idea that messages should happen in some time interval relies on a global clock. However, there is the potential for misalignment due to message delays and local clock skews, rather than which messages are emitted and received. Such problems are avoided by making the time intervals an order of magnitude larger than the maximum clock skew and message delays [Cranefield, 2005].

We assume that for a commitment $c(x,y,Cre,Det,Dis)$ being operationalized with respect to a history vector H if when x or y knows a message m and the counter-party receives m they will do so at a time to make the same inferences over Cre, Det, and Dis. Under this assumption, an operationalization protocol is sufficient to support alignment.

Theorem 2 states that a commitment operationalization protocol always makes it possible to rectify alignment via messaging.

Theorem 2. Let $P^{\mathbb{C}}$ be a protocol that operationalizes a set of commitments $\mathbb{C}$ such that $c(x,y,Cre,Det,Dis) \in \mathbb{C}$. If history vector $H \in \mathcal{U}_{\mathcal{R},\mathcal{M}}$ is in $P^{\mathbb{C}}$'s UoE then there exists a longer H'' in $P^{\mathbb{C}}$'s UoE that is aligned with respect to $c(x,y,Cre,Det,Dis)$.

Proof sketch. If H is misaligned. Definition 10 and Cupid's semantics for lifecycle events [Chopra and Singh, 2015a, D_{15} to D_{19}] imply role s's model entails E. Base case: $E = m$. By R1 extend H to a history vector H'' in $P^{\mathbb{C}}$'s UoE (Definition 7) by inserting a notification m_i from m's sender to r in their respective histories. Inductive hypothesis: assume there exists a history H' in $P^{\mathbb{C}}$'s UoE extending H s.t. if $E = F \sqcap G$ or $E = F \sqcup G$ then r knows F and G, or for $E = F \ominus G$ r knows F and s knows G. Inductive step: Extend H' to H'' with an m_i via rules R1 to R12. By the time assumption and Cupid's semantic definitions [Chopra and Singh, 2015a, Def. 16 to Def. 19] s and r are aligned. □

6 Conclusions

Tosca addresses challenges of decentralized commitment enactment. Given a set of commitments defined over a protocol, it enables automatically synthesizing a new protocol that supports alignment, a form of commitment-level interoperability. Furthermore, the synthesized protocol preserves liveness and safety, both of which are also necessary for interoperation. The new protocol may be thought of as a *fleshing out* of the input protocol. Tosca brings together several advances—in the specification of protocols, commitments, and ideas about interoperability—toward supporting decentralization.

Tosca establishes a separation of concerns between commitments and protocols. Protocols are specified in BSPL whereas commitments are specified in Cupid—languages developed independently of each other. Notably, in Cupid, commitments are defined over a database schema, not a protocol. We use the fact that BSPL specifications are interpreted over databases in layering Cupid specifications over BSPL specifications.

Decentralization is a theme of growing interest, (e.g., for norm compliance [Baldoni *et al.*, 2015] and monitoring [Bulling *et al.*, 2013]). Tosca's architecture is distinct from shared memory (environment) approaches (e.g., [Omicini *et al.*, 2008]). Such approaches would benefit from Tosca in that they would also need a clear specification of interactions, both in terms of meanings and protocols, even if alignment itself would be trivial because of shared memory.

Other works treat commitments [Chesani *et al.*, 2013] and protocols [Yadav *et al.*, 2015] separately without studying their relationship. Günay et al. [Günay *et al.*, 2015] generate commitment specifications from requirements. We understand commitment specifications as requirements and synthesize operational protocol specifications.

Analogously Searle [1995, pp. 26–27] demarcates between constitutive rules, which make social actions possible by ascribing institutional (social) facts, and norms, which prescribe institutional facts. This separation is adopted for commitment protocols overlaying constitutive rules [Baldoni *et al.*, 2013]. Moreover, norms are both defined over institutional facts and interpreted at different levels of abstraction using constitutive rules within agents [Criado *et al.*, 2013] and legal institutions [King, 2016; King *et al.*, 2017]. We separate richer concerns: commitments, focusing on relational information, overlaying operational protocols, focusing on information causality.

Future directions should address limitations and further applications. Tosca maintains agent autonomy, making alignment possible but not regimented and hence limited by the extent to which autonomous agents communicate message notifications. We demonstrated Tosca on a business domain but it could just as well be applied to requirements in other domains that involve interaction. In particular, healthcare and government (e.g., national) contracts, studied in connection with Cupid are prime candidates for Tosca. Since Tosca provides support for messaging requirements via protocols, the extent to which it can be applied to agent development should be investigated (e.g., using communication abstractions supported in agent programming frameworks such as [Boissier *et al.*, 2013]).

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. King, Günay and Chopra were supported by the EPSRC grant EP/N027965/1 (Turtles). Singh thanks the US Department of Defense for partial support under the Science of Security Lablet.

References

[Baldoni *et al.*, 2013] Matteo Baldoni, Cristina Baroglio, Elisa Marengo, and Viviana Patti. Constitutive and Regulative Specifications of Commitment Protocols: a Decoupled Approach. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 4(2), 2013.

[Baldoni *et al.*, 2015] Matteo Baldoni, Cristina Baroglio, Amit K. Chopra, and Munindar P. Singh. Composing and Verifying Commitment-Based Multiagent Protocols. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 10–17. AAAI Press, 2015.

[Boissier *et al.*, 2013] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747–761, 2013.

[Bulling *et al.*, 2013] Nils Bulling, Mehdi Dastani, and Max Knobbout. Monitoring norm violations in multi-agent systems. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2013)*, pages 491–498. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[Chesani *et al.*, 2013] Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. Representing and monitoring social commitments using the event calculus. *Autonomous Agents and Multiagent Systems*, 27(1):85–130, 2013.

[Chopra and Singh, 2008] Amit K. Chopra and Munindar P. Singh. Constitutive interoperability. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, pages 797–804, 2008.

[Chopra and Singh, 2009] AK Chopra and MP Singh. Multiagent commitment alignment. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, pages 937–944, 2009.

[Chopra and Singh, 2015a] Amit K. Chopra and Munindar P. Singh. Cupid: Commitments in Relational Algebra. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2052–2059, 2015.

[Chopra and Singh, 2015b] Amit K. Chopra and Munindar P. Singh. Generalized Commitment Alignment. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, pages 453–461, 2015.

[Cranefield, 2005] Stephen Cranefield. A Rule Language for Modelling and Monitoring Social Expectations in Multi-Agent Systems. In *Coordination, Organization, Institutions and Norms in Multi-Agent Systems. Lecture Notes in Computer Science.*, volume 3913, pages 246–258. Springer, 2005.

[Criado *et al.*, 2013] N Criado, E. Argente, P. Noriega, and V. Botti. Reasoning about constitutive norms in BDI agents. *Logic Journal of the IGPL*, 22(1):66–93, 2013.

[Günay *et al.*, 2015] Akın Günay , Michael Winikoff, and Pınar Yolum. Dynamically generated commitment protocols in open systems. *Autonomous Agents and Multi-Agent Systems*, 29:192–229, 2015.

[Huget and Odell, 2004] Marc-Philippe Huget and James Odell. Representing agent interaction protocols with agent UML. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE 2000), Lecture Notes in Computer Science*, volume 3382, pages 16 – 30, 2004.

[King *et al.*, 2017] Thomas C. King, Marina De Vos, Virginia Dignum, Catholijn M. Jonker, Tingting Li, Julian Padget, and M. Birna van Riemsdijk. Automated multi-level governance compliance checking. *Autonomous Agents and Multi-Agent Systems*, 2017.

[King, 2016] Thomas C. King. *Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance*. PhD thesis, Delft University of Technology, 2016.

[Miller and Mcginnis, 2007] Tim Miller and Jarred Mcginnis. Amongst First-Class Protocols. In *Proceedings of the 8th International Workshop on Engineering Societies in the Agents World (ESAW 2007). Lecture Notes in Computer Science.*, volume 1957, pages 208 – 223, 2007.

[Omicini *et al.*, 2008] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.

[Pitt *et al.*, 2001] Jeremy Pitt, Lloyd Kamara, and Alexander Artikis. Interaction patterns and observable commitments in a multi-agent trading scenario. In *Proceedings of the 5th International Conference on Autonomous Agents*, pages 481–488, 2001.

[Searle, 1995] John R. Searle. *The Construction of Social Reality*. The Free Press, New York, 1995.

[Singh, 1999] MP Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.

[Singh, 2011] Munindar P. Singh. Information-driven interaction-oriented programming: BSPL, the blindingly simple protocol language. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, pages 491–498, 2011.

[Singh, 2012] Munindar Singh. Semantics and Verification of Information-Based Protocols. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, pages 1149–1156, 2012.

[Yadav *et al.*, 2015] Nitin Yadav, Michael Winikoff, and Lin Padgham. HAPN: Hierarchical Agent Protocol Notation. In *Proceedings of the International Workshop on Coordination, Organisation, Institutions and Norms in Multi-Agent Systems (COIN@IJCAI)*, 2015.

[Yolum and Singh, 2002] Pınar Yolum and Munindar P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In *Proceedings of the first International Joint Conference on Autonomous Agents and Multiagent Systems: part 2*, pages 527–534, Bologna, Italy, 2002.