Auton Agent Multi-Agent Sys (2007) 14:143–163
DOI 10.1007/s10458-006-7232-1

An algebra for commitment protocols*

Ashok U. Mallya · Munindar P. Singh

Published online: 5 April 2006

Abstract Protocols enable unambiguous, smooth interactions among agents. Commitments among agents are a powerful means of developing protocols. Commitments enable flexible execution of protocols and help agents reason about protocols and plan their actions accordingly, while at the same time providing a basis for compliance checking. Multiagent systems based on commitments can conveniently and effectively model business interactions because the autonomy and heterogeneity of agents mirrors real-world businesses. Such modeling, however, requires multiagent systems to host a rich variety of protocols that can capture the needs of different applications. We show how a commitment-based semantics provides a basis for refining and aggregating protocols. We propose an approach for designing commitment protocols wherein traditional software engineering notions such as refinement and aggregation are extended to apply to protocols. We present an algebra of protocols that can be used to compose protocols by refining and merging existing ones, and does this at a level of abstraction high enough to be useful for real-world applications.

Keywords Commitments · Interaction Protocols · Formal methods · Multiagent system modelling and design

1 Introduction

Multiagent systems composed of autonomous and heterogeneous agents provide a convenient and accurate model for describing and enacting many real-life processes and interactions.

* We thank Amit Chopra, Nirmit Desai, and anonymous referees for valuable comments. This research was supported partially by the NSF under grant DST-0139037, and partially by DARPA under contract F30603-00-C-0178.

A. U. Mallya · M. P. Singh (✉)
Department of Computer Science, North Carolina State University,
Raleigh, NC 27695-8206, USA
e-mail: ashok.mallya@gmail.com

M. P. Singh
e-mail: singh@ncsu.edu

While autonomy and heterogeneity are what make the multiagent paradigm attractive, heterogeneity gives rise to incompatibility and autonomy to unpredictability. Agents need to understand each other and behave in predictable ways for their interactions to be fruitful. To achieve consensus and facilitate interaction between agents, standards are required, as in most distributed systems. Web Services are an example of how standards enable heterogeneous systems to interact with each other. Recent efforts for Web Service *choreography*—which deals with the way services interact—and *orchestration*—which deals with the way services are composed using other services—address service interactions [16] similar in spirit to agent interaction protocols. Agent interaction, however, requires higher-level abstractions to deal with the rich variety of interactions found in multiagent systems.

1.1 Interaction protocols

A protocol is a description of the steps involved in an interaction. Protocols make interactions coherent and easy to implement. The use of protocols has successfully solved the problem of standardization in areas such as computer networks. Likewise, the heterogenous and distributed structure of multiagent systems necessitates clear protocols to govern any interaction. Network protocols explain the steps to be taken in great detail, sometimes even enumerating all possible events that can occur. For example, the Session Initiation Protocol (SIP), which is used to setup phone calls over the Internet, describes every message that needs to be sent for setting up and tearing down calls and also every possible resultant reply for the message [18]. By contrast, multiagent systems require protocols to be specified at a high level of abstraction, to accommodate the complexity of agent systems, and to not overwhelm protocol designers with unnecessary details.

While protocols are needed to force an agent to behave in a predictable manner, they should also allow flexibility of execution. A protocol that allows only one sequence of steps does not let its participants leverage their autonomy. A restrictive protocol, however, is not always bad. If a protocol allows only a single computation, checking whether the participants are compliant with the protocol is trivial. Any step that does not agree with the protocol signals a violation. As protocols become more flexible, however, compliance verification becomes harder, since many choices are offered to the participants at any step of the protocol. Consequently, protocol design is an exercise in finding the right balance between flexibility of execution and ease of compliance checking.

1.2 Motivation

The tradeoffs between execution and verification to be borne in mind make protocol design a nontrivial undertaking. It requires human expertise and knowledge of the application domain. To reduce unnecessary effort and to prevent reinventing the wheel, designers should be able to create new protocols by refining or combining existing protocols whose properties are well understood. A sound theory of refining and composing protocols would assist designers in ascertaining the properties of protocols. An algebra of protocols that includes operators for merging and refinement is needed as the basis for protocol composition.

Our central claim is that protocols can be characterized in terms of their content, not just their sequence of steps. We develop a protocol algebra which is at once a high-level abstraction of protocols and a useful tool for composing protocols and reasoning about them, as we demonstrate with an example.

1.3 Contribution

Our main contribution is in developing an algebra for composing protocols. Just as conceptual modeling in general involves abstractions such as refinement and aggregation, so must the conceptual modeling of protocols.

- We develop an algebra that provides the underpinnings of refinement and aggregation abstractions for protocols.
- We demonstrate how the use of commitments allows reasoning about protocols that leads to richer interaction patterns from existing ones.
- We outline how a hierarchy of protocols can be generated based on commitments. This hierarchy aids reasoning about which protocol is the most general for a given business process.
- Our algebra is a high-level abstraction that relates to real-world protocols, and hence is easy for protocol designers to understand.

1.4 Organization

The rest of this paper is organized as follows. Section 2 introduces the technical background, and some illustrative examples that are used throughout the paper. Section 3 develops our theory of semantics of protocol subsumptions, introduces the protocol algebra and demonstrates its utility in composing protocols. Section 4 summarizes the paper, identifies related work in the field, and charts out future directions.

2 Technical framework

We represent protocols as transition systems similar in spirit to finite state machines. These protocols generate computations or *runs*, which are sequences of states that a valid protocol execution can go through. We devise a hierarchical classification based on the runs generated by protocols. Runs are composed of *states* that the protocol computation (execution) goes through based on the *actions* that the participants in the given protocol perform. This classification forms the basis of our work. Next, we introduce commitments, discuss some scenarios from our running example, and then define the basic technical concepts needed for our semantics.

2.1 Commitments in protocols

Commitments among agents are an abstraction of contracts that exist in the real world [3, 19]. Commitments lend coherence to interactions because they help agents plan based on the actions of others, and they are, in principle, enforceable. Commitment-based protocols are more flexible than traditional formalisms like finite state machines and Petri nets [24, 26]. By specifying the states that need to be reached in terms of commitments, they can allow multiple paths to achieve a state, and consequently create a flexible protocol specification.

A commitment $\mathsf{C}(x, y, p)$ denotes that the agent x is responsible to the agent y for bringing about the condition p. Here x is the *debtor*, y the *creditor*, and p the *condition* of the commitment, expressed in a suitable formal language. Commitments can also be *conditional*, denoted by $\mathsf{CC}(x, y, p, q)$, meaning that x is committed to y to bring about p if q holds.

2.1.1 Commitment operations

Commitments are created, satisfied, and transformed in certain ways. Conventionally, six operations are defined on commitments. A detailed exposition of these operations is given in Singh [20] and is outlined here, for ease of exposition. Below, c stands for $\mathsf{C}(x, y, p)$

CREATE$(\boldsymbol{x}, \boldsymbol{c})$**.** The commitment c is created by its debtor x.
CANCEL$(\boldsymbol{x}, \boldsymbol{c})$**.** The debtor x of the commitment c cancels the commitment. A cancellation is typically compensated for by other commitments.
RELEASE$(\boldsymbol{y}, \boldsymbol{c})$**.** The creditor y releases the debtor x from the commitment c.
ASSIGN$(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{c})$**.** The creditor y transfers the commitment so that x is committed to z instead of y.
DELEGATE$(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{c})$**.** The debtor x transfers the commitment so that z is committed to y.
DISCHARGE$(\boldsymbol{x}, \boldsymbol{c})$**.** The debtor x fulfils the commitment c.

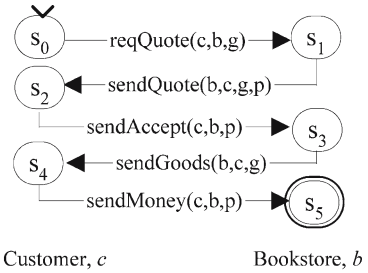
A commitment is said to be *active* if it has been created, but not yet operated upon by any of the other commitment operations. In other words, a commitment stays active after its creation until it is discharged, delegated, assigned, or canceled or its debtor is released.

2.2 Running example

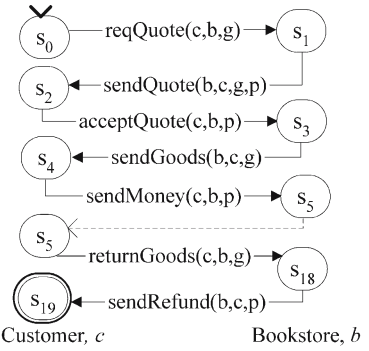
As a real-world example, we consider a variant of the NetBill protocol [21] used by a customer's agent to purchase a book from an online bookstore's agent. We identify four distinct, but related, scenarios that can arise during this purchase interaction. Each of these scenarios requires a different amount of effort from the participants in terms of protocol execution, planning, and coordination. Both agents would benefit from being able to compare scenarios to choose the one that best serves their interests.

1. The customer asks the bookstore for a price quote on a book, and upon receiving a quote from the bookstore, accepts the bookstore's offer. The bookstore sends the book, and the customer pays for it. Figure 1a shows this interaction. This interaction sequence belongs to the *purchase* protocol.
2. The bookstore is willing to refund the price of returned books. This scenario is similar to the previous scenario till the book is delivered to the customer, but is longer, since the customer then returns the book for a refund. Figure 1b. shows this interaction.
3. The customer delegates the payment to a third party, e.g., a bank. Such a situation is not very different from using a credit card to pay for goods, and is shown in Fig. 1c.
4. The customer wants insured shipping, and the bookstore's existing shipper does not insure goods. The bookstore negotiates with and contracts out the shipping to a shipper. Here, the shipper delivers the books to the customer, after which the shipper is paid by the bookstore. To complicate matters, the customer pays the bookstore via its bank like in the previous scenario. This scenario is shown in Fig. 2.

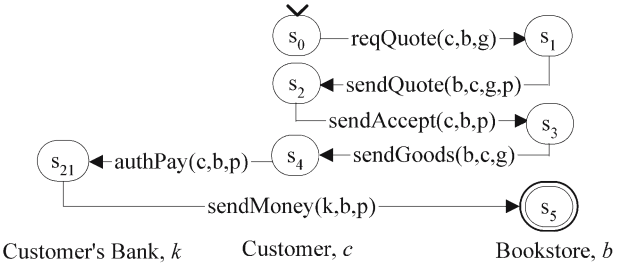
In Figures 1a–c, 2, 4a, b, ellipses represent states, named s_i. Solid arrows are labeled by the messages that are passed between the participating agents. These messages correspond to actions that the agents take. Note that each of these figures represent a possible scenario, i.e., a *run* of the protocol. Also, states of the runs are drawn in different columns (analogous to swimlanes in UML) to show the interacting agents clearly even though states are global and are maintained by all interacting agents. Dashed lines indicate that the states they link are the same, albeit shown more than once in the figure. An algorithm to guarantee consistent state maintainence across participants is described elsewhere [5].



(a) Purchase protocol scenario 1: normal purchase



(b) Purchase protocol scenario 2: goods returned for a refund



(c) Purchase protocol scenario 3: customer pays via bank

Fig. 1 Three scenarios of the purchase example

Table 1 explains the meanings of the states that the first scenario runs through. Table 2 shows the meanings of the messages passed, where c represents the customer, b, the bookstore, g, the book that the customer is interested in buying, and k, the customer's bank. The *delegate* message relates to corresponding commitment operation.

2.3 Propositions

Propositions capture facts about what conditions hold, what commitments have been made, and whether these commitments have been fulfilled. The propositions used in a protocol are assumed to be understood by agents involved in the protocol. In the *purchase* example, we use the propositions given in Table 3. In addition to these, active commitments are also represented as propositions, as we shall explain when discussing states.

2.4 Actions

Agents perform actions to bring about changes in the world. In our framework, actions are modeled as messages sent by an agent to other agents. Just like an action, a message sent by an agent can affect the state of a protocol in which the agent participates. Messages may be implemented in different ways. For example, filling a form with credit card information and submitting it over the web is a message that represents a transfer of funds. The set of actions is denoted by $\mathbb{A}$. The meanings of the actions used in our *purchase* example are given in Table 2.

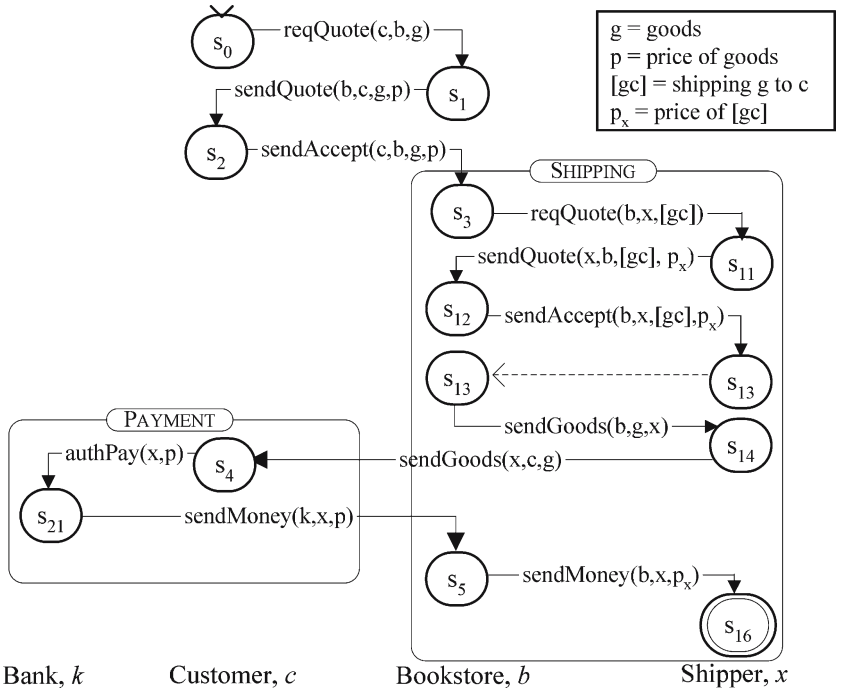


Fig. 2 Purchase protocol scenario 4: shipping via a separate shipper and payment via bank

Table 1 Meaning of states in the purchase protocol

State	Meaning
s_1	Customer has asked the bookstore the price of the goods. No commitments made.
s_2	Bookstore has quoted a price for the said goods. The bookstore is now willing to send the goods if the customer promises to pay for them
s_3	Customer has agreed to the bookstores price. The customer is willing to pay the price if the books are delivered.
s_4	Bookstore has delivered the book.
s_5	Customer has paid for the book.

Table 2 Meanings of actions (modeled as messages) in the purchase protocol

Message	Meaning
$reqQuote(c, b, g)$	c asks b what the price of g is
$sendQuote(b, c, g, p)$	b quotes price p to the c, for g
$sendAccept(c, b, g, p)$	c accepts the price p quoted by b for g. c is now committed to pay if the book is sent to it
$sendGoods(b, c, g)$	b sends g to c
$sendMoney(c, b, p)$	c sends the money p to b
$delegate(c, k, \mathsf{C})$	c delegates the commitment C to k
$returnGoods(c, b, g)$	c returns g to b
$sendRefund(b, c, p)$	b refunds the money p to c
$authPay(c, b, p)$	c authorizes its bank to pay the amount p to b; essentially c delegates $\mathsf{C}(c, b, p)$ to k

Table 3 Meanings of propositions used in the purchase protocol

Proposition	Meaning
$reqQuote(c, b, g)$	c has requested a quote for g from b
$quote(b, c, g, p)$	b quotes to c price p for g, i.e., b will deliver if c commits to pay upon delivery. This is represented by $\mathsf{CC}(b, c, goods(b, c, g), acceptQuote(c, b, g, p))$
$acceptQuote(c, b, g, p)$	c has accepted the price p that b quoted for g, i.e, c commits to pay if the goods are delivered. This is represented by $\mathsf{CC}(c, b, pay(c, b, p), goods(b, c, g))$
$goods(b, c, g)$	g has been delivered to c by b
$pay(c, b, p)$	The amount p has been paid to b by c
$return(c, b, g)$	g has been returned to b by c
$refund(b, c, p)$	The amount p has been refunded to c by b

In addition to the actions shown, $\mathbb{A}$ also contains actions corresponding to the commitment operations applied to each commitment. For example, $\mathbb{A}$ contains an action $delegate(c, k, \mathsf{C})$ corresponding to the operation $delegate(c, k, \mathsf{C})$, where C is a commitment made by c.

2.5 States

A protocol has many *states* that it goes through, during the course of its execution. A state is a snapshot of the world and is labeled by the set of propositions that are true in it. The propositions in the universe of discourse are termed the *frame*. A frame serves as a common ontology for the propositions used by a protocol. Frames provide the universe of discourse of a protocol. A state is an assignment of truth values to propositions. For example, state s_1 of the *purchase* example is labeled by the set $\{reqQuote(c, b, g)\}$ and state s_0 by $\{\mathsf{true}\}$. We denote the label of a state s by $[s]$. Table 4 shows the labels that are assigned to states in the *purchase* protocol. The set of states is denoted by $\mathbb{S}$. We include in this set a unique start state s_ϕ, which is labeled by the set $\{\mathsf{true}\}$. In the *purchase* example, $s_0 = s_\phi$.

2.6 Runs

A run is one possible execution sequence of a protocol. A protocol can allow many computations, or *runs*. A run is a sequence of states $\langle s_0 \ldots s_i \ldots \rangle$. We use $\in$ to indicate the occurrence of a state on in a run. For example, $s_i \in \langle s_0 \ldots s_i \ldots \rangle$. In this paper, we consider only finite runs. The empty run is allowed.

Table 4 State labels in the purchase protocol

State	Associated Label
s_0	$\{\mathsf{true}\}$
s_1	$\{reqQuote(c, b, g)\}$
s_2	$\{quote(b, c, g, p)\}$
s_3	$\{\mathsf{C}(b, c, goods(b,c,g)), \mathsf{CC}(c,b, pay(c, b, p), goods(b, c, g))\}$
s_4	$\{goods(b, c, g), \mathsf{C}(b, pay(c, b, p))\}$
s_5	$\{goods(b, c, g), pay(c, b, p)\}$
s_{21}	$\{goods(b, c, g), \mathsf{C}(k, b, pay(k, b, p))\}$
s_{17}	$\{goods(b, c, g), pay(c, b, p)\}$
s_{18}	$\{goods(b, c, g), return(c, b, g), \mathsf{C}(b, c, refund(b, c, p))\}$
s_{19}	$\{goods(b, c, g), return(c, b, g), refund(b, c, p)\}$

The operator $\prec_\tau$ orders states temporally with respect to a run τ, so that $s_i \prec_\tau s_j$ implies that s_i occurs before s_j in the run τ. The concatenation of a state s_n to a run $\tau = \langle s_0 s_1 \ldots s_k \rangle$, written as $\tau \circ s_n$, is given by appending the state to the run, i.e., by $\langle s_0 s_1 \ldots s_k s_n \rangle$. The sequence of states in a *subrun* of a run $\tau = \langle s_0 \ldots s_n \rangle$ is a possibly noncontiguous subsequence of $s_0 \ldots s_n$. We denote the first state of a run τ by $[\tau]^0$, the nth state by $[\tau]^n$, and the last state by $[\tau]^\top$.

2.7 Protocols

Computationally, a protocol corresponds to a set of computations that it allows. These can be captured as a set of runs where any of the runs that *subsume* the given runs may be realized. That is, each run in a protocol defines a sequence of steps that must be performed in the same order relative to each other. The subsumption of runs is defined below. A protocol is represented as a transition system as defined by a tuple $\langle \mathbb{A}, \mathbb{S}, S_0, \Delta, \mathbb{F}, \mathbb{R} \rangle$ where $\mathbb{A}$ is a set of actions, $\mathbb{S}$ is a set of states, S_0 is the set of initial states ($S_0 \subseteq \mathbb{S}$), Δ is a set of transitions ($\Delta \subseteq \mathbb{S} \times \mathbb{A} \times \mathbb{S}$), $\mathbb{F}$ is a set of final states, ($\mathbb{F} \subseteq \mathbb{S}$), and $\mathbb{R}$ is a set of roles (or participants).

Δ contains transitions of the form $\langle s_i, a, s_j \rangle$, where $s_i, s_j \in \mathbb{S}$ and $a \in \mathbb{A}$. Here s_i is the source of the transition and s_j its destination. Such a transition advances a computation from state s_i to state s_j when an action a is performed, i.e., when the message corresponding to a is sent (and received, assuming synchronous message passing, for convenience). In other words, a run can be generated from a protocol by the successive concatenation of transitions beginning from the initial state of the transition system. The concatenation of a transition to a run appends the destination of transition to the run if the source of the transition matches the last state of the run. Consequently, a run $\langle s_0 s_1 s_2 \ldots s_n \rangle$ can be generated by a protocol whose initial state is s_0, and whose transition set contains the elements $\langle s_0, \_, s_1 \rangle$, $\langle s_1, \_, s_2 \rangle$ and so on till $\langle s_{n-1}, \_, s_n \rangle$, where $s_0 \in S_0$ and $s_n \in \mathbb{F}$. The set of all such runs is denoted by $[P]$.

Protocols are specified by propositions and actions that cause states to change. The semantics of actions are given in terms of commitments such as those shown in Table 2. Given the actions and their semantics, the formalization of a protocol is straightforward. The transition function of a protocol can be specified explicitly as state-action-state triples or as a set of rules that are complied into such triples for runtime efficiency. Two example transition mechanisms for commitment-based protocols are commitment machines [26] and nonmonotonic commitment machines [4]. For example, Tables 2–4, along with a set of rules for determining the new state given the old state and the action taken would define the purchase protocol.

3 Reasoning about protocols

This section describes our theory of comparing and refining protocols. Section 3.1 defines how states are deemed similar to one another, Section 3.2 defines what it means for a run to subsume another or be similar to another, Section 3.3 defines subsumption and similarity of protocols, and Section 3.4 uses comparisons of commitment-operation based propositions to relate different protocols.

3.1 Similarity of states

States form the fundamental components of runs, and are identified (and labeled) by sets of propositions. Any comparison of states, therefore, must be based on comparing propo-

sitions. This section introduces three state-similarity functions ι, σ, and $\alpha_{A,P}$, all based on commitment propositions, and shows how these help relate different runs.

A *state-similarity function* f is a mapping from a state to a set of states, i.e., $f\colon \mathbb{S} \mapsto 2^{\mathbb{S}}$. From such a function, we can induce a binary relation $\approx_f \subseteq \mathbb{S} \times \mathbb{S}$, which is defined as $\approx_f = \{(s, f(s)) : s \in \mathbb{S}\}$. That is,

$$s_i \approx_f s_j \iff s_j \in f(s_i) \tag{1}$$

Definition 1 *A state similarity function f is well formed if $\approx_f$ is reflexive, symmetric, and transitive, i.e., $\approx_f$ is an equivalence relation.*

We denote by $\mathbb{F}$ the set of all well-formed state-similarity functions.

We constrain all runs to prevent *stuttering*, i.e., to not have consecutive states that are similar, i.e., for every run $\tau = \langle s_0 \ldots s_n \rangle$, s_i is not similar under f to s_{i+1}, for all $0 \leq i < n$.

3.1.1 Identity state-similarity

ι is the identity state-similarity function. That is, $s_i \approx_\iota s_j$ if and only if s_i and s_j are labeled by same set of propositions. $\iota(s_i) = \{s_j | [s_i] = [s_j]\}$. $\approx_\iota$ is an equivalence relation.

What other kinds of similarity functions can one develop? We have developed our semantics to support reasoning about protocols on commitments among autonomous agents. States are labeled by the propositions that hold in them. Whereas domain propositions can have different semantics in different applications, commitments are an abstraction that can be used across domains with uniform semantics. Since a commitment has a well-defined life cycle and is a *directed* obligation, comparison of commitments can be performed based on two criteria.

- A comparison based on which agent is committed to which other agent.
- A comparison based on which commitment operations were performed.

As examples of usage of both the above criteria, we introduce two state similarity functions, the creditor state-similarity function σ and the role-and-commitment state-similarity function α.

3.1.2 Creditor state-similarity

Under the creditor state-similarity function σ, a state s_i is similar to a state s_j if in the two states all the participants of the protocol have the same commitments being made towards them, regardless of which agent makes it. Since the creditor of a commitment is immaterial under σ and a *delegate*$(\cdot, \cdot, \cdot)$ action changes the creditor of a commitment, σ can be defined as $\sigma(s_i) = \{s_j | s_j$ can be reached by finite number of *delegate*$(\cdot, \cdot, \cdot)$ actions from $s_i\}$

As an example, consider states s_4 and s_{21} from the of the example scenarios.These states are similar under σ because, as described in Table 4, these states have propositions representing commitments that differ only in their creditors. $\approx_\sigma$ is an equivalence relation.

3.1.3 Role-and-commitment state-similarity

From the point of view of a participant of a protocol, two states can be said to be similar if they involve the same commitments. Based on this intuition, we describe role-and-commitment state-similarity function α. A state s_i is similar to a state s_j under $\alpha_{A,P}$, where A is a set

of roles and P is a set of propositions, if the commitments made by any role in A to any other role in A, and the propositions in P that hold at s_i, also hold at s_j. If, for example, A represents all the roles in a protocol and P represents all the proposition used by that protocol, then $\alpha_{A,P}$ can be used as a similarity function to detect cases where the protocol has been merged with other protocols. This is explained in greater detail in Section 3.4.2.

3.2 Subsumption and similarity of runs

Comparisons among protocols are based on a notion of *subsumption* of runs. $[\![f]\!\rangle$ denotes subsumption operator over runs. The operator $[\![f]\!\rangle$ is an order-preserving mapping from one run to another, and depends on the function f.

Definition 2 *A run τ_j subsumes a run τ_i under function f if and only if, for every state s_i that occurs in τ_i, there occurs a state s_j in τ_j that is similar under f, and s_j has the same temporal order relative to other states in τ_j as s_i does with states in τ_i.*

$$\begin{aligned} \tau_j [\![f]\!\rangle \tau_i \iff (\forall s_i, s_i' \in \tau_i : (\exists s_j, s_j' \in \tau_j \\ \text{and } s_i \approx_f s_j \text{ and } s_i' \approx_f s_j' \\ \text{and } (s_i \prec_{\tau_i} s_i' \Rightarrow s_j \prec_{\tau_j} s_j'))) \end{aligned} \tag{2}$$

That is, longer runs subsume shorter ones, provided they have similar states occurring in the same order. Before we describe properties of run subsumption, we define *run-similarity*.

Definition 3 *A run τ_i is similar to a run τ_j under a well-formed state-similarity function f if and only if the two runs are of equal length and every kth state of τ_i is similar under f to the kth state in τ_j. In notation, $[\tau_i]^k \approx_f [\tau_j]^k, 0 \leq k < |\tau_i|$ and $|\tau_i| = |\tau_j|$.*

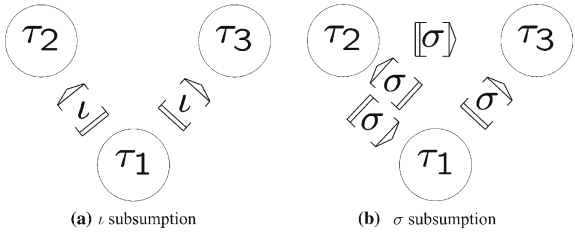
Lemma 1 *If a run τ_j subsumes a run τ_i under f, then, for all subruns β_i of τ_i, there exists a subrun β_j of τ_j such that $\beta_i \approx_f \beta_j$. That is, $\tau_j [\![f]\!\rangle \tau_i \Rightarrow (\forall \beta_i$ subrun of $\tau_i, \exists \beta_j$ subrun of $\tau_j : \beta_i \approx_f \beta_j)$.*

Proof This lemma holds trivially for subruns of lengths 0 or 1. We prove this lemma by induction on the length of subruns. As the base case, we note that by the definition of run subsumption, there must be a pair of states in both runs for which the above property holds, that is, $(\forall \beta_i$ subrun of $\tau_i : |\beta_i| = 2 \Rightarrow (\exists \beta_j$ subrun of τ_j and $\beta_i \approx_f \beta_j))$. Assume this property holds for subruns of a particular length $k > 2$. Next, consider $|\beta_i'| = k + 1$, where $[\beta_i']^k = s_i'$. The first k states of β_i' form a subrun of length k. Therefore, there exists a subrun β_j of τ_j such that $\beta_i \approx_f \beta_j$ and β_j is the earliest such subrun in τ_j. Consider the k pairwise temporal precedence relationships (in τ_i) between the first k states of β_i' and s_i'. By the definition of run subsumption, we know that since $\tau_j [\![f]\!\rangle \tau_i, \exists s_j' \in \tau_j$ such that $[\beta_j]^\top \prec_{\tau_j} s_j'$ and $s_j' \approx_f s_i'$. Let s_j' be the earliest such state in τ_j, so that $\beta_j' = \beta_j \circ s_j'$. Thus, the inductive hypothesis holds for subruns of length $k + 1$. By induction, the lemma holds. □

Theorem 1 *Under any well-formed state similarity function, run subsumption is reflexive, transitive, and antisymmetric up to state similarity.*

Proof The reflexivity of run subsumption follows from the reflexivity of state-similarity functions. The transitivity of run subsumption follow from the definition.

The antisymmetry of run subsumption means that if two runs subsume each other under some well-formed state similarity function f, then those runs are similar under f. By



(a) ι subsumption **(b)** σ subsumption

Fig. 3 Subsumption of purchase protocol runs under ι and σ

Lemma 1 we know that if $\tau_j [\![f ⟩\!] \tau_i$, then for each subrun of τ_i, there is a similar subrun of τ_j. Let the entire run τ_i be treated as a subrun of itself. Then, there is a similar subrun in τ_j, i.e., $|\tau_j| \geq |\tau_i|$. In the same way, from $\tau_i [\![f ⟩\!] \tau_j$, we infer that $|\tau_i| \geq |\tau_j|$. Hence, $|\tau_i| = |\tau_j|$. Since the runs are of equal length and all the subruns of one have similar subruns in the other, $\tau_i \approx_f \tau_j$ holds. □

Let us consider an example to better explain the above concepts. Let τ_1, τ_2, and τ_3 be the runs shown in Figures 1a–c, respectively. We then have $\tau_2 [\![\iota ⟩\!] \tau_1$. Also, $\tau_3 [\![\iota ⟩\!] \tau_1$. However, τ_1 subsumes neither τ_2 nor τ_3, because τ_3 has a state s_{21}, whose label does not match any state label in τ_1, and τ_2 has states s_{18} and s_{19}, whose labels do not match any state label in τ_1. Run τ_3 does not subsume τ_2 because of s_{21}, and τ_2 does not subsume τ_3 because of s_{18} and s_{19}.

Next, we consider run subsumption under the creditor state-similarity function σ. Here, $\tau_2 [\![\sigma ⟩\!] \tau_1$. Also, since $s_{21} \in \sigma(s_4)$, $\tau_1 [\![\sigma ⟩\!] \tau_2$. Run τ_3 subsumes both τ_1 and τ_2 for the same reason, but neither τ_1 nor τ_2 subsumes τ_3, since neither of them have states that are σ-similar to s_{18} and s_{19}.

Figure 3a shows the subsumption relation between the runs τ_1, τ_2, and τ_3 under the identity function ι and Fig. 3b, under σ.

3.3 Subsumption and similarity of protocols

3.3.1 Closure of protocol runs

In line with our intuitions that generic protocols allow many variations, we describe the semantics of a protocol as a set of runs that is closed under run subsumption. That is, if any run occurs in the set of runs of a protocol, all the runs that subsume it (under a well-formed state-similarity function) are also in that set. Given a well-formed state-similarity function f, a protocol P allows all runs that subsume any run in $[P]$. We define this new set of runs as the set of runs *allowed* by the protocol, and denote this set by $[\![P]\!]$.

Observation 1 From the definition of closure of the set of allowed runs of a protocol $[\![P]\!]$ under a well-formed state-similarity function f, we have that $[\![P]\!] = \{\tau | \forall \tau' \in [P] : \tau [\![f ⟩\!] \tau'\}$.

We refer to $[\![P]\!]$ also as the *closure* of the protocol P under a well-formed state-similarity function.

Operationally, the runs allowed by a protocol completely characterize that protocol by naturally illustrating the key tradeoff in protocol design, that of *flexibility versus compliance:* A protocol that allows many runs is better than one that allows a few runs, since the many-run protocol affords more choice and flexibility in protocol execution to the participants. However, checking for compliance can potentially be more demanding when protocols have

more runs that do not subsume a common run. The definition of the subsumption of protocols reflects these intuitions.

Definition 4 *A protocol P_j subsumes a protocol P_i under a state-similarity function f if and only if, every run in $[\![P_i]\!]$ subsumes, under f, a run in $[\![P_j]\!]$.*

$$P_j [\![f]\!\rangle P_i \iff \forall \tau_i \in [\![P_i]\!] \exists \tau_j \in [\![P_j]\!] : \tau_i [\![f]\!\rangle \tau_j \tag{3}$$

If P_j is a protocol that has short runs (and consequently all runs that subsume them) and P_i is a protocol that has long runs only, then P_j subsumes P_i. Since long runs subsume shorter ones, protocols with long runs only are subsumed by protocols with short runs as well.

Before we describe the properties of protocol subsumption, we define protocol similarity as follows:

Definition 5 *Two protocols are similar under a state similarity function f if and only if for every run in one protocol, there exists at least one similar (under f) run in the other protocol, and vice versa.*

$$\begin{aligned} P_j \approx_f P_i \Leftrightarrow \forall \tau_i \in [\![P_i]\!], \exists \tau_j \in [\![P_j]\!] : \tau_i \approx_f \tau_j \\ \text{and } \forall \tau_j \in [\![P_j]\!], \exists \tau_i \in [\![P_i]\!] : \tau_j \approx_f \tau_i \end{aligned} \tag{4}$$

Theorem 2 *For any well-formed state-similarity function f, the protocol subsumption relation $[\![f]\!\rangle$ is a partial order, i.e., reflexive, transitive, and antisymmetric (up to state similarity).*

Proof The reflexivity and transitivity of protocol subsumption follow from its definition. Because of the property of closure of the set of runs of a protocol under run subsumption and the definition of protocol subsumption, we know that if $P_i [\![f]\!\rangle P_j$, then $\forall \tau_j \in [\![P_j]\!] : \exists \tau_i \in [\![P_i]\!] : \tau_j \approx_f \tau_i$. Conversely, if $P_j [\![f]\!\rangle P_i$, then $\forall \tau_i \in [\![P_i]\!] : \exists \tau_j \in [\![P_j]\!] : \tau_i \approx_f \tau_j$. Together, these are equivalent to $P_j \approx_f P_i$. □

3.4 The protocol algebra

We now introduce our protocol algebra as consisting of two operators (*merge* and *choice*), their identity elements ($\mathbb{1}$ and $\mathbb{0}$, respectively), and an ordering relationship (protocol subsumption, as defined above).

Merge. The merge operator, denoted by $\otimes_f$, splices two protocols (under a state-similarity function f) to produce a new protocol. A merge of two protocols involves interleaving their runs.

Definition 6 *The merge of two protocols P and Q under a well-formed state-similarity function f creates a protocol whose closure consists exactly of runs that subsume some run from $[\![P]\!]$ and some run from $[\![Q]\!]$.*

$$[\![P \otimes_f Q]\!] = \{r | \exists r_p \in [\![P]\!], \exists r_q \in [\![Q]\!] : r [\![f]\!\rangle r_p \text{ and } r [\![f]\!\rangle r_q\} \tag{5}$$

Choice. The choice operator, denoted by $\oplus_f$, chooses runs from the closures of the protocols it operates upon and the closure of their merge under f.

Definition 7 *The choice of two protocols P and Q under a well-formed state-similarity function f creates a protocol whose closure consists exactly of runs that subsume either a run from $[\![P]\!]$ or a run from $[\![Q]\!]$.*

$$[\![P \oplus_f Q]\!] = \{r | \exists r_p \in [\![P]\!], \exists r_q \in [\![Q]\!] : r [\![f]\!\rangle r_p \text{ or } r [\![f]\!\rangle r_q\} \tag{6}$$

Constants. The properties of the merge operator lead us to define two protocols, $\mathbb{0}$ and $\mathbb{1}$. The $\mathbb{0}$ protocol is an "impossible" protocol, which does not have any runs. $[\![\mathbb{0}]\!] = \{\}$. The $\mathbb{1}$ protocol is a "trivial" protocol which contains the zero-length run in its semantics, and consequently contains all possible runs. $[\![\mathbb{1}]\!] = \{\tau | \forall f \in \mathbb{F}, \tau [\![f]\!] \tau_\phi\}$, where $\tau_\phi = \langle\rangle$. The $\mathbb{0}$ and the $\mathbb{1}$ protocols form the bottom and the top element, respectively, of a protocol hierarchy based on the merge function. All protocols are subsumed by the $\mathbb{1}$ protocol and all protocols subsume the $\mathbb{0}$ protocol.

3.4.1 Formal results

We briefly present some formal results, which simplify reasoning about protocols using our algebra. First, we present a formulation for the operators in terms of protocol subsumption. Since Definitions 6 and 7 state the properties of the operators only in terms of the runs of the protocols, this new formulation simplifies the understanding of the behavior of the merge and choice operators, and likens them to set intersection and union, respectively.

Theorem 3 *Given the run-subsumption relation $[\![f]\!]$ under a well-formed state-similarity function f, the merge $\otimes_f$ of two protocols P and Q under f is the greatest lower bound of P and Q under the protocol subsumption relation $[\![f]\!]$.*

Proof Let $R = P \otimes_f Q$. That is,

$$[\![R]\!] = \{r | \exists p \in [\![P]\!], \exists q \in [\![Q]\!] : r [\![f]\!] p \text{ and } r [\![f]\!] q\} \tag{7}$$

From Observation 1, it is easy to see that $P [\![f]\!] R$ and $Q [\![f]\!] R$. That is, R is a lower bound of P and Q.

Consider some protocol R' that is a lower bound of P and Q, i.e., $P [\![f]\!] R'$ and $Q [\![f]\!] R'$. By definition of protocol subsumption,

$$\forall r'_p \in [\![R']\!], \exists p \in [\![P]\!] : r'_p [\![f]\!] p \tag{8}$$

$$\forall r'_q \in [\![R']\!], \exists q \in [\![Q]\!] : r'_q [\![f]\!] q \tag{9}$$

From Eq. (7) we know that all runs that subsume a run each from $[\![P]\!]$ and $[\![Q]\!]$ are in $[\![R]\!]$. From Eqs. (8) and (9), we know that every run in $[\![R']\!]$ is just such a run. Hence, $\forall r' \in [\![R']\!] : r' \in [\![R]\!]$. From the closure property of protocol run-sets, we further infer that $\forall r' \in [\![R']\!], \exists r \in [\![R]\!] : r [\![f]\!] r'$, which means that $R [\![f]\!] R'$. Since any R' is subsumed by $P \otimes_f Q$ under f, this means that $\otimes_f$ gives the greatest lower bound of P and Q under f. □

Theorem 4 *Given the run-subsumption relation $[\![f]\!]$ under a well-formed state-similarity function f, the choice $\oplus_f$ of two protocols P and Q under f is the least upper bound of P and Q under the protocol subsumption relation $[\![f]\!]$.*

Proof Let $R = P \oplus_f Q$. That is,

$$[\![R]\!] = \{r | \exists p \in [\![P]\!] : r [\![f]\!] p \text{ or } \exists q \in [\![Q]\!] : r [\![f]\!] q\} \tag{10}$$

From Observation 1, it is easy to see that $R [\![f]\!] P$ and $R [\![f]\!] Q$. That is, R is an upper bound of P and Q.

Consider some protocol R' that is an upper bound of P and Q, i.e., $R' [\![f]\!] P$ and $R' [\![f]\!] Q$. By definition of protocol subsumption,

$$\forall p \in [\![P]\!], \exists r'_p \in [\![R']\!] : p [\![f]\!] r'_p \tag{11}$$

$$\forall q \in [\![Q]\!], \exists r'_q \in [\![R']\!] : q [\![f]\!\rangle r'_q \tag{12}$$

Let $r \in [\![R]\!]$. From Eq.(10), without loss of generality, we can assume that $\exists p \in [\![P]\!] : r [\![f]\!\rangle p$. From this, Eq. (11), and the transitivity of run subsumption, we infer that $\exists r'_p \in [\![R']\!] : r [\![f]\!\rangle r'_p$. By the definition of protocol subsumption under f, $R' [\![f]\!\rangle R$. Since any R' that subsumed P and Q under f subsumes $P \oplus_f Q$ under f, this means $\oplus_f$ gives the least upper bound of P and Q under f. □

Essentially, the merge is analogous to the intersection of the sets of runs of the protocols being merged, where run similarity is used to determine equality of elements of the sets. Formally,

$$[\![P \otimes_f Q]\!] = [\![P]\!] \cap_f [\![Q]\!]$$

where $\cap_f$ is the *run intersection* operator, defined over sets of runs A and B as

$$A \cap_f B = \{r | \exists p \in A : r \approx_f p \text{ and } \exists q \in B : r \approx_f q\}$$

We could alternatively define $\cap_f$ as the greatest lower bound of P and Q under f.

Lemma 2 *Given protocols P and Q and a well-formed state-similarity function f, $r \in [\![P \otimes_f Q]\!] \Rightarrow r \in [\![P]\!]$ and $r \in [\![Q]\!]$*

Proof Consider $r \in [\![P \otimes_f Q]\!]$. From Definition 6, we know that $r [\![f]\!\rangle r_p$ and $r [\![f]\!\rangle r_q$ for some $r_p \in [\![P]\!]$ and $r_q \in [\![Q]\!]$. From this and Observation 1 (the closure of sets of protocol runs), we infer that $r \in [\![P]\!]$ and $r \in [\![Q]\!]$. □

Lemma 3 *Given protocols P and Q and a well-formed state-similarity function f, $r \in [\![P]\!]$ and $r \in [\![Q]\!] \Rightarrow r \in [\![P \otimes_f Q]\!]$*

Proof Consider $r \in [\![P]\!]$. From Observation 1 and the reflexivity of run subsumption under any well-formed state-similarity function, we can infer $\exists r_p \in [\![P]\!]$ such that $r [\![f]\!\rangle r_p$. Similarly, from $r \in [\![Q]\!]$, we infer $\exists r_q \in [\![Q]\!]$ such that $r [\![f]\!\rangle r_q$. From Definition 6, we conclude that $r \in [\![P \otimes_f Q]\!]$. □

Choice is analogous to the union of the sets of the runs, using run similarity to compare elements of the sets. Formally,

$$[\![P \oplus_f Q]\!] = [\![P]\!] \cup_f [\![Q]\!]$$

where $\cup_f$ is the *run union* operator, defined over sets of runs A and B as

$$A \cup_f B = \{r | \exists p \in A : r \approx_f p \text{ or } \exists q \in B : r \approx_f q\}$$

We could alternatively define $\cup_f$ as the least upper bound of P and Q under f.

Lemma 4 *Given protocols P and Q and a well-formed state-similarity function f, $r \in [\![P \oplus_f Q]\!] \Rightarrow r \in [\![P]\!]$ or $r \in [\![Q]\!]$*

Proof Consider $r \in [\![P \oplus_f Q]\!]$. From Definition 7, we know that $r [\![f]\!\rangle r_p$ or $r [\![f]\!\rangle r_q$ for some $r_p \in [\![P]\!]$ and $r_q \in [\![Q]\!]$. From this and Observation 1 (the closure of sets of protocol runs), we infer that $r \in [\![P]\!]$ or $r \in [\![Q]\!]$. □

Lemma 5 *Given protocols P and Q and a well-formed state-similarity function f, $r \in [\![P]\!]$ or $r \in [\![Q]\!] \Rightarrow r \in [\![P \oplus_f Q]\!]$*

Proof If $r \in [\![P]\!]$, then, from Observation 1 and the reflexivity of run subsumption under any well-formed state-similarity function, we can infer $\exists r_p \in [\![P]\!]$ such that $r [\![f]\!\rangle r_p$. Similarly, if $\exists r_q \in [\![Q]\!]$, then $r [\![f]\!\rangle r_q$. From Definition 7, we conclude that $r \in [\![P \oplus_f Q]\!]$. □

The following results are then obvious

1. The merge of a protocol with itself yields the same protocol (idempotence).

$$(P \otimes_f P) = P.$$

2. The merge operator is commutative and associative.

$$P \otimes_f Q = Q \otimes_f P$$
$$P \otimes_f (Q \otimes_f R) = (P \otimes_f Q) \otimes_f R$$

3. Merge distributes over choice.

$$P \otimes_f (Q \oplus_f M) = (P \otimes_f Q) \oplus_f (P \otimes_f M)$$

Proof From Lemmas 2, 3, 4, 5, and the distributivity of "and" over "or",

$$\begin{aligned} r \in [\![P \otimes_f (Q \oplus_f M)]\!] &\Rightarrow r \in [\![P]\!] \text{ and } r \in [\![Q \oplus_f M]\!] \\ &\quad \text{by Lemma 2} \\ &\Rightarrow r \in [\![P]\!] \text{ and } (r \in [\![Q]\!] \text{ or } r \in [\![M]\!]) \\ &\quad \text{by Lemma 4} \\ &\Rightarrow (r \in [\![P]\!] \text{ or } r \in [\![Q]\!]) \\ &\quad \text{and } (r \in [\![P]\!] \text{ or } r \in [\![M]\!]) \\ &\quad \text{by distributivity of "and" over "or"} \\ r \in [\![P \otimes_f (Q \oplus_f M)]\!] &\Rightarrow r \in [\![(P \otimes_f Q) \oplus_f (P \otimes_f M)]\!] \qquad (13) \\ &\quad \text{by Lemmas 3 and 5} \end{aligned}$$

□

Similarly, we can prove

$$r \in [\![(P \otimes_f Q) \oplus_f (P \otimes_f M)]\!] \Rightarrow r \in [\![P \otimes_f (Q \oplus_f M)]\!] \qquad (14)$$

From Eqs. (13), (14), and the definition of protocol-similarity (Definition 5), we conclude that merge distributes over choice.

4. The merge of any protocol with $\mathbb{1}$ gives that protocol and the merge of any protocol with $\mathbb{0}$ gives $\mathbb{0}$. In this way, the $\mathbb{1}$ protocol is the identity element and the $\mathbb{0}$ is the nil element for merge.
$P \otimes_f \mathbb{1} = P$
$P \otimes_f \mathbb{0} = \mathbb{0}$

Choice also supports idempotence, commutativity, and associativity. The choice of a protocol with $\mathbb{1}$ yields $\mathbb{1}$ and the choice of a protocol with $\mathbb{0}$ yields that protocol itself.

1. Idempotence.
$(P \oplus_f P) = P$

2. Commutativity.
   $(P \oplus_f Q) = (Q \oplus_f P)$
3. Associativity.
   $P \oplus_f (Q \oplus_f R) = (P \oplus_f Q) \oplus_f R$
4. Distributivity over merge.
   $P \oplus_f (Q \otimes_f R) = (P \oplus_f Q) \otimes_f (P \oplus_f R)$
5. Choice with $\mathbb{1}$ and $\mathbb{0}$
   $P \oplus_f \mathbb{1} = \mathbb{1}$
   $P \oplus_f \mathbb{0} = P$

3.4.2 Applying the algebra

Now we discuss how the algebra can be applied to create new protocols. The choice operator $\oplus_f$ allows us to choose between runs belonging to different protocols. This operator can be used, for example, when multiple ways of payment exist, such as payment by credit card, or payment by personal check. The result of the choice operator is a protocol whose set of runs is larger and thus offers more choices than the individual protocols to which the choice was applied.

The merge operator is more interesting. As an example of its application, consider the run shown in Fig. 1c. This run belongs to the merge of the simple purchase shown in Fig. 1a. and payment, shown in Fig. 4a. The merge is performed under the creditor state-similarity function σ. As a more complicated example, consider a run of the refined purchase example as shown in Fig. 2. This run belongs to the refined purchase protocol, which is the result of a merge of the simple purchase, the shipping, and the payment protocols. The state-similarity function used here is $\alpha_{A,P}$. Under $\alpha_{A,P}$, A is a set of agents, P, a set of propositions, and two states are similar if all commitments between agents agents in A and all propositions in P that exist in one state also exist in the other. Under $\alpha_{A,P}$, where A denotes the set containing the participants of *Shipping*, i.e., $\{b, x, c\}$, and P denotes the set of all propositions that are used in *Shipping*, we see that the *Shipping* run shown in Fig. 4b. is subsumed by the refined *Purchase* run shown in Fig. 2. Specifically, the states s_3, s_{11}, s_{12}, s_{13}, s_{14}, s_5, and s_{16} of refined *Purchase* are $\alpha_{A,P}$-similar to the states s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, and s_{16} of *Shipping* respectively. Similarly, the states s_4 and s_{21} of the refined *Purchase* are similar to states s_{20} and s_{21} of *Payment* under $\alpha_{A,P}$, where A denotes $\{c, k\}$ and P denotes the set of all propositions used in *Payment*. Consequently, the refined *Purchase* run subsumes *Payment*. Note that Fig. 2 shows only one run of the refined purchase protocol. Given the semantics of the merge operator, the refined purchase protocol allows more runs, since all valid interleavings of runs of the merged protocols are allowed. One such run could be where the shipping protocol is started before the first step of the purchase protocol. In practice, data dependencies and temporal state ordering are specified to filter the set of runs generated by interleaving, as done, for example, in the OWL-P framework [5], which describes a *composition profile* for composing protocols. The composition profile describes roles that have to be played by the same agent, data dependencies and ordering constraints, if any, among the messages in the protocol.

4 Discussion

Our research program seeks to develop rich abstractions methodologies that will ease the development of large-scale open systems. This paper is part of our ongoing research in that

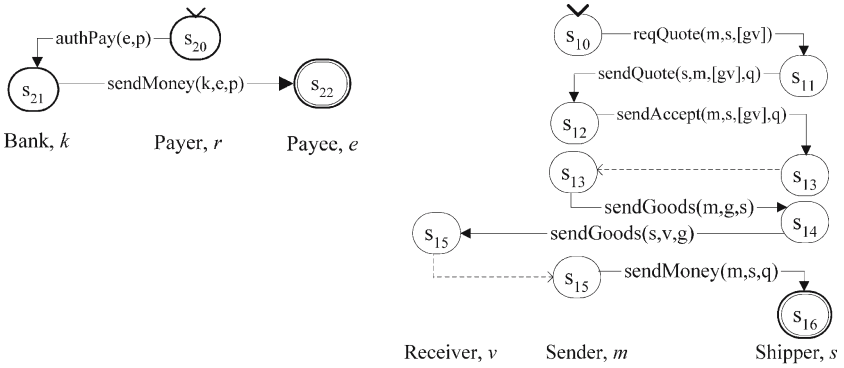


Fig. 4 A payment and a shipping protocol. (a) Payment-via-bank protocol and (b) Shipping protocol

direction. Singh et al. present an overview of the motivations of this program and of the applications envisioned for it [5].

The framework presented in this paper can serve as the foundation for developing design-time tools, and possibly for automated, runtime composition. Complete automation of protocol composition requires a complete specification of the behavior of a protocol. This is rarely the case when dealing with complex agent interaction protocols that can find applications in business processes modeling or Web Service composition. Most realistic settings require considerable context-sensitive information, which may be encoded as policies local to the agent. Such contexts may be based on motivations such as trust and economics, which can change unpredictably. The autonomy of agents in a multi agent system allows agents to behave differently under different contexts. It is this dynamic behavior that makes the agent paradigm attractive for application to open systems, and cannot be statically specified for all but the simplest of agents. Our framework helps develop tools that aid protocol designers in tailoring an existing protocol to meet their requirements by automatically verifying properties of the designed protocol.

Our understanding of protocols as specifications of the minimum states that a computation should contain is analogous to the minimal process execution semantics as defined in the MIT Process Handbook [15]. There is an underlying assumption that concise specifications are better than elaborate ones, since flexibility of a protocol is desirable in business applications where opportunities can be profitably exploited. In some cases, however, a maximal execution semantics might be applicable, e.g., in a protocol for which compliance checking is costly or difficult or where unexpected actions are undesirable.

4.1 Literature

Our work relates to and draws both from well established and emerging fields. Business processes have received much attention lately because of the economic benefits of cross-enterprise business. Coordination and interaction protocols have been studied by the agent research community so that agent conversations and interaction can be computationally realized. We list here selected literature from both areas.

4.1.1 Workflows and processes

Business processes have been traditionally automated as workflows. Recently, the web service model has been applied to process automation.

Workflows have been studied extensively as Petri-net based models of business processes [22, 23]. The Workflow Management Coalition (WfMC) is a standards body that has created a reference model for workflows [6]. This model has two basic parts, a modeling and an enactment part. The model prescribes a workflow engine as the system that executes the workflow. These models specify a rigid sequence of steps. Workflows require human intervention to handle most exceptions. Because of their inflexibility, workflows have had only limited success.

The MIT Process Handbook [15] is a project that aims to create a hierarchy of commonly used business processes. Based on this hierarchy, Grosof and Poon [11] develop a system to represent and execute business rules.

Of late, web services have been touted as the solution to the business interoperability problem. The need for process composition and interoperability has led to the development of standards for orchestration and choreography of web services [16]. Orchestration refers to intra-service planning and choreography to an overall view of inter-service coordination. Here, we shall mention only two important standards, WSCI and BPEL4WS. The Web Services Choreography Interface (WSCI) is an XML-based language that describes a service interface by the flow of messages sent and received by the service. The standard, however, looks at protocols one level lower than our view, since each WSCI specification corresponds to a role in our scheme. The Business Process Execution Language for Web Services (BPEL4WS) is currently the most widely used web services standard for describing business processes [1]. However, BPEL4WS is no more than a procedural script encoded in XML.

Fu et al. [10] develop methods to verify if a given web service will adhere to a given conversation protocol. Their work develops formal results about verification of protocol compliance for protocols based on finite state machines. Benatallah [12] develop a protocol algebra for petri nets and show its applicability to workflows and web services. However, this approach suffers from the same pitfalls as workflows modeled using Petri nets.

4.1.2 Interaction protocols

Yolum and Singh [26] give one of the first accounts of the use of commitments in modeling agent interaction protocols and the flexibility that it affords the participating agents. Colombetti [8] describe how commitments relate to FIPA-ACL messages and demonstrate with an example. Both approaches highlight the benefits of a commitment-based approach to interaction protocol design.

Johnson et al. [13] develop a scheme for identifying when two commitment-based protocols are equivalent. Their scheme, however, is simplistic, classifying protocols based solely on their syntactic structure. Our work provides stronger results about the relationships between protocols from an application point of view and relates better to the Web Services approach.

Bussmann et al. [2] present a design methodology to aid in the selection of a protocol from a library of existing protocols to apply to agent-based control applications. They identify criteria like the number of agents, the number of roles, and the number and kind of commitments and use these to select a protocol from an existing pool of interaction protocols. This approach is quantitative, and lacks a formal semantics to base the methodology on.

Pitt and Mamdani [17] describe a semantics for agent interaction protocols using the Belief-Desire-Intention (BDI) theory. Using this semantics, they outline the design of a system of agent plans that are instantiated by agents to carry on conversations with other BDI agents. In our work, an agents beliefs, desires, and intentions are private to that agent. We work with social commitments which are observable by all agents and whose breach is easier to verify.

In more recent work, Vitteau and Huget [25] describe an approach for designing agent interaction protocols using modular *micro-protocols*. This scheme is similar to our protocol design proposal in spirit. However, Vitteau and Huget do not provide a formal basis for putting protocols together.

4.1.3 Agent communication

Flores et al. [7] describe how agent conversations using commitments can be modeled by giving meanings to commitment operations incrementally via message exchanges. Their work is a good example of an operationalization of commitment protocols using lower-level messages relating to communicative acts. As they correctly point out, our algebra can be applied at a higher level than their theory.

Fornara et al. [9] develop a method of specifying changes in ontological relationships among communicative acts of agents so that the changes in meaning of the acts can be easily captured when applied to different institutions (such as an auction). Whereas our work describes how to combine protocols to apply in different scenarios, Fornara et al. demonstrate how a change in the meaning of utterances can be handled in a principled manner in different contexts. A key contribution of their work is the specification of the organizational structure (the institution) as an integral part of a commitment-based conversation protocol.

Kagal and Finin [14] propose Rei, a language and a methodology for the specification of policies and policy-based institutions. Under this methodology, policies are used to express deontic notions such as permissions, prohibitions, and obligations. A policy engine determines at runtime utterances or actions that an agent can say or perform. Kagal and Finin also provide a mapping of the concepts they develop into the FIPA ACL and KQML, thus promoting wider usage. Policy-based institutions are realizations of *spheres of commitment*, in which a social structure exists because of commitments among agents and these agents have rights, privileges, and duties to create and operate upon commitments based on their position in the society. Kagal and Finins work could thus provide a bridge for the operationalization of our theory.

4.2 Conclusions and directions

This paper described a semantic approach to commitment protocols that yields a simple algebra for protocols. This algebra provides a basis for conceptual reasoning about protocols in terms of refinement and aggregation, which is essential if we are to engineer protocols that way other software systems are engineered. To our knowledge, this work is unique in formulating the problem of problem design at a conceptual level. Partly, it derives it uniqueness from a careful consideration of the commitments that underlie protocols in multiagent settings.

This work opens up some interesting challenges. One, it would help consider how the algebra will work with more subtle kinds of state-similarity functions. Two, the abstractions supported by our algebra must be woven into a methodology for designing protocols. Three,

such methodologies should be supported by tools that give appropriate reasoning assistance to designers. We are pursuing these directions in our ongoing research.

References

1. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., & Weerawarana, S. (2003). Business Process Execution Language for Web Services (BPEL 1.1).
2. Bussmann, S., Jennings, N. R., & Wooldridge, M. (2003). Re-use of interaction protocols for agent-based applications. In F. Giunchiglia, J. Odell, & G. Weiß (Eds.), *Third international workshop on agent-oriented software engineering (AOSE 2002)*, Vol. 2585 of *Lecture notes in computer science* (pp. 73–87). Springer.
3. Castelfranchi, C. (1993). Commitments from individual intentions to groups and organizations. In *Proceedings of the AAAI-93 workshop on AI and theories of groups and organizations: Conceptual and empirical research*.
4. Chopra, A., & Singh, M. P. (2003). Nonmonotonic commitment machines. In M.-P. Huget, & F. Dignum (Eds.). *Proceedings of the AAMAS workshop on agent communication languages and conversation policies*.
5. Desai, N., Mallya, A. U., Chopra, A. K., & Singh, M. P. (2005). Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering*, *31*(12), 1015–1027.
6. Fischer, L. (Ed.). (2004). *The workflow handbook 2004*. Lighthouse Point, FL: Future Strategies Inc. Compendium of standards and articles by the Workflow Management Coalition (WfMC).
7. Flores, R. A., Pasquier, P., & Chaib-draa, B. (2006). Conversational semantics with social commitments. *Journal of Autonomous Agents and Multiagent Systems*, this volume.
8. Fornara, N., & Colombetti, M. (2003). Defining interaction protocols using a commitment-based agent communication language. In *Proceedings of the second international joint conference on autonomous agents and multiagent systems (AAMAS)* (pp. 520–527). ACM Press.
9. Fornara, N., Viganó, F., & Colombetti, M. (2006). Agent communication and institutional reality. *Journal of Autonomous Agents and Multiagent Systems*, this volume.
10. Fu, X., Bultan, T., & Su, J. (2004). Realizability of conversation protocols with message contents. In *Proceedings of the 2004 international conference on web services* (pp. 96–105). IEEE Computer Press.
11. Grosof, B. N., & Poon, T. C. (2003). SweetDeal representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proceedings of the twelth international conference on the world wide web* (pp. 340–349).
12. Hamadi, R., & Benatallah, B. (2003). In Zhou, X., & Schewe, K.-D. (eds.), *Proceedings of the fourteenth Australian database conference. ADC*, Vol. 17 of *conferences in research and practice in information technology* (pp. 191–200). Australian Computer Society.
13. Johnson, M. W., McBurney, P., & Parsons, S. (2003). When are two protocols the same? In M.-P. Huget, (Ed.), *Communication in multiagent systems: Agent communication languages and conversation policies*, Vol. 2650 of *Lecture notes in artificial intelligence* (pp. 253–268). Berlin: Springer-Verlag.
14. Kagal, L., & Finin, T. (2006). Modelling communicative behavior using permissions and obligations. *Journal of Autonomous Agents and Multiagent Systems*, this volume.
15. Malone, T. W., Crowston, K., & Herman, G. A. (Eds.). (2003), *Organizing business knowledge: The MIT process handbook*. Cambridge, MA: MIT Press.
16. Peltz, C. (2003). Web service orchestration and choreography. *IEEE Computer, 36*(10), 46–52.
17. Pitt, J., & Mamdani, A. (2000). Communication protocols in multi-agent systems. In F. Dignum, & M. Greaves (Eds.). *Issues in agent communication*, (pp. 160–177). Vol. 1916 of *Lecture notes in artificial intelligence* Springer.
18. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., & Schooler, E. (2002). SIP: Session Initiation Protocol. IETF Proposed Standard.
19. Singh, M. P. (1991). Social and psychological commitments in multiagent systems. In *AAAI fall symposium on knowledge and action at social and organizational levels* (pp. 104–106).
20. Singh, M. P. (1999). An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law, 7*, 97–113.
21. Sirbu, M. A. (1997). Credits and debits on the internet. *IEEE Spectrum, 34*(2), 23–29.
22. van der Aalst, W., Desel, J., & Oberwis, A. (Eds.). (2000). *Business process management: Models, techniques, and empirical studies*. Vol. 1806 of *Lecture notes in computer science*. Berlin: Springer-Verlag.
23. van der Aalst, W., & van Hee, K. (Eds.). (2002). *Workflow management models, methods, and systems*. Cambridge, MA: MIT Press.

24. Verdicchio, M., & Colombetti, M. (2002). Commitments for agent-based supply chain management. *ACM SIGecom Exchanges, 3*(1), 13–23.
25. Vitteau, B., & Huget, M.-P. (2004). Modularity in interaction protocols. In: F. Dignum (Ed.). *Advances in agent communication*, (pp. 291–309). Vol. 2922 of *Lecture notes in artificial intelligence* Springer.
26. Yolum, P., & Singh, M. P. (2002). 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 (AAMAS)* (pp. 527–534). ACM Press.