A Semantic Approach for Designing E-Business Protocols

Ashok U. Mallya
Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27695–7535
Email: aumallya@ncsu.edu

Munindar P. Singh
Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27695–7535
Email: mpsingh@ncsu.edu

Abstract— **Business processes involve interactions among autonomous partners. We propose that these interactions be specified modularly as protocols. Protocols can be published, enabling implementors to independently develop components that respect published protocols and yet serve diverse interests. A variety of business protocols would be needed to capture subtle business needs. We propose that the same kinds of conceptual abstractions be developed for protocols as for information models. Specifically, we consider (1) *refinement*: a subprotocol may satisfy the requirements of a superprotocol, but support additional properties; and (2) *aggregation*: a protocol may combine existing protocols. In support of the above, this paper develops a semantics of protocols and an operational characterization of them. This supports judgments about the potential subclass-superclass relations between protocols, which are a result of protocol refinement. It also enables protocol aggregation by splicing a protocol into another protocol.**

I. INTRODUCTION

Modern e-business processes span multiple autonomous entities or business partners. Such processes therefore are based on a rich variety of interactions among software components that are independently designed and configured and which represent independent (sometimes mutually competitive) business interests. Web services provide a basis for realizing such processes by enabling businesses to interoperate in a standardized manner. This has led to interest in technologies such as coordination and process flows, distributed transactions of various flavors, and conversations. While these approaches have some benefits, they mostly take a centralized perspective, akin to workflow technologies, viewing a process as a series of tasks to be performed. This proves too tedious for reliable modeling and too rigid for enactment, which is the reason workflow technologies have been considered a failure in many practical settings. The present paper relates to all of these efforts, but concentrates on the semantical aspects of the interactions among business partners.

We propose a novel framework for thinking about processes. Simply put, *a process instantiates one or more business protocols* among designated parties. We define a protocol as a specification of a logically related set of interactions. A protocol specifies only the key desired aspects of the interactive behavior; it leaves the details of a local implementation entirely up to those who implement the protocol.

Realistic business settings will need an endless variety of business processes. While some of these processes will be widely deployed, several will be customized to special application domains, industries, and circumstances. While the hard-coded systems of todays process management require a serious integration and configuration effort to accomodate change, we imagine that by employing well-specified, published protocols to compose processes, the various stake-holders can considerably simplify their integration and configuration efforts [1]. Given a set of protocols, they would only need to acquire implementations for the roles of those protocols. RosettaNet is already a step in this direction [2]. It defines over 100 protocols (callep PIPs in their terminology). RosettaNet's protocols are limited to two-party implementations and are mostly two-step protocols.

We want general protocols to support flexibility, and specialized protocols to support efficiency, security, or risk management. For example, we can imagine a generic payment protocol as well as specializations of it such as payment by cash, credit card, checks, wire transfer, and so on. Each of these would differ in the steps that each participant takes. Moreover, the protocols only specify the interactions, not the local policies of the participating entities, such as that they don't take cash after sunset. Protocols enable such policies to be inserted but are not directly concerned with the policies. As long as we recognize that these are payment protocols, our top-level design goal, namely, to enable some form of payment would be satisfied.

The most fundamental computer science approach for dealing with complexity is to enable reuse. Two of the most basic ideas for doing so are to build a specialization-generalization hierarchy and to aggregate components. The objective of this paper is develop notions akin to traditional subsumption and aggregation that are applicable to protocols. We develop two main classes of abstractions: *refinement* (like the subclass-superclass hierarchy) and *aggregation* (like the part-whole hierarchy). We develop a formal semantics to support the hierarchy and propose and algebra to facilitate reasoning about protocols.

Contribution: Traditional workflow technologies are quite rigid in that they allow very little variation from the specified sequence of steps. Hence, composition of new work-

flows, or creating variants of existing ones involve considerable effort. Our contribution is in developing a basis for easily comparing protocols and an algebra for aggregating them to create business processes. The algebra provides the underpinnings of refinement and aggregation abstractions for protocols. The algebra is a high-level abstraction that relates to real-world interaction protocols, and hence is easy for protocol designers to understand. We also demonstrate how the use of commitments allows reasoning about protocols that leads to richer interaction patterns from existing ones. Further, we outline how a hierarchy of protocols can be generated based on commitments.

II. TECHNICAL MOTIVATION

As a running example, we consider a *purchase* interaction in which a customer wants to buy a book from an online bookstore. The bookstore obtains the customers' order for a book if the customer accepts the price quoted by the bookstore for that book. The book is then shipped to the customer, and the bookstore is then paid for the book. The actual execution of the process, however could involve many different scenarios, two of which we shall describe shortly.

Commitments in E-Business Protocols: To talk about how e-business protocols can be aggregated or refined, we must represent not just the behaviors of the participants but also how the contractual relationships among the participants evolve over the course of an interaction. Doing so enables us to determine if the interactions are indeed compliant with the stated protocols. The contractual relationships of interest are naturally represented through *commitments*, which have recently gained importance in the field of multiagent systems [3]. Commitments capture the obligations of one party to another. For example, the customer's agreement to pay the price for the book after it is delivered is a commitment that the customer has towards the bookstore. Commitments lend coherence to the interactions because they enable agents to plan based on the actions of others. In principle, violations of commitments can be detected and, with the right social relationships, commitments can be enforced. Enforceability of contracts is necessary when the participants are autonomous and heterogeneous [4].

Why Formal Semantics?: As explained above, the objective of this paper is develop notions akin to traditional subsumption and aggregation that are applicable to protocols. Doing so presupposes that we have a crisp semantics and can reason formally about protocols. Accordingly, our task is to develop a semantics that facilitates flexible actions.

III. 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 computation (execution) goes through. State changes are caused by *actions* that the participants perform. We devise a hierarchical classification based on the runs generated by protocols. This classification forms the basis of our work.
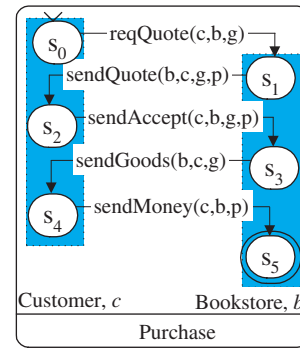

Fig. 1.   Purchase example, scenario 1

Commitments: A commitment $C(x, y, p)$ denotes that the agent x is responsible to the agent y for bringing about the condition p. Here x is called the *debtor*, y the *creditor*, and p the *condition* of the commitment, expressed in a suitable formal language. Commitments are created, satisfied, and transformed in certain ways, using commitment operations. The conventional commitment operations are described in more details elsewhere [7].

Example Scenarios: We identify two distinct, but related, scenarios that can arise during the above book 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. These scenarios are shown in Figures 1 and 3. *Customer* refers to the customer's agent and *Bookstore* refers to the bookstore's agent. Ellipses represent states, labeled 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.

1) Normally, the customer would ask the bookstore for a price quote on the book it wishes to buy, and upon receiving a quote from the bookstore, would accept the bookstore's offer. The bookstore would then send the book, after which the customer would send the payment. This is modeled after the NetBill protocol [8]. Figure 1 shows this interaction.

2) The bookstore might have to contract out the actual shipping to a shipper. This might happen, for example, if the customer wants insured shipping, and the bookstore's existing shipper does not insure goods. Here, the bookstore interacts with the shipper, and gets the books delivered to the customer. The shipper is then paid by the bookstore, but only after the book has been delivered to the customer. The customer pays the bookstore via its bank. This scenario is shown in Figure 3 and discussed in detail in Section V.

Propositions: Propositions capture facts about what conditions hold, what commitments have been made, and whether these commitments have been fulfilled. The set of propositions is represented by $\mathbb{P}$

States: A state captures the condition of the world by assigning truth values to the propositions pertaining to a protocol. A state is labeled by the set of propositions which hold at that state. The set of states is denoted by $\mathbb{S}$.

Actions: Agents perform actions to bring about changes in the world. The communicative actions of the agents correspond to messages sent by the agents to others. An action of an agent affects the state of a protocol in which it participates.

The set of actions is denoted by $\mathbb{A}$. All participants of a protocol are assumed to know the meaning of actions used in that protocol.

Runs: A run is a sequence of states $\langle s_0 \ldots s_i \ldots \rangle$. A run describes the states that a single execution of a protocol goes through.

Protocols: A protocol is 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 initial state; $s_0 \in \mathbb{S}$, 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) in the protocol.

Δ contains transitions of the form $\langle s_i, a, s_j \rangle$, where $s_i, s_j \in \mathbb{S}$ and $a \in \mathbb{A}$. Such a transition advances a run that is in state s_i to state s_j based on an action a. 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 $\langle \_, \_, s_n \rangle$, where $s_n \in \mathbb{F}$. This set of runs is the minimal set of runs defined by the protocol, and is denoted by $[M]$. We denote by $[\![M]\!]$, the set of all runs that a protocol M can generate.

IV. REASONING ABOUT PROTOCOLS

This section describes our theory of comparison of protocols and protocol refinements. It defines the concepts of similarity of states, and subsumption of runs and protocols.

State-Similarity: A *state-similarity function* f is a mapping from a state to a set of states. For example, if $f(s)$ gives the set of states which have the same commitments being made towards the same participants, regardless of which participant makes it, then all such states are similar.

Run Subsumption: A run τ_1 subsumes a run τ_2 under a state-similarity function f if every state that occurs in τ_2 also occurs in τ_1, and in the relative order of all such states are the same in both runs.

A. Subsumption of protocols

When enacting processes using protocols, a protocol that generates only short runs is preferable over a protocol that generates longer runs since short runs speed up the protocol execution. At the same time, 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 its execution to the participants. We now develop some results about subsumption of protocols and demonstrate them with examples.

A protocol M_j *subsumes* a protocol M_i under the function f if and only if, for every run τ_i that M_i can generate, M_j can generate a run τ_j that is subsumed by τ_i under f.
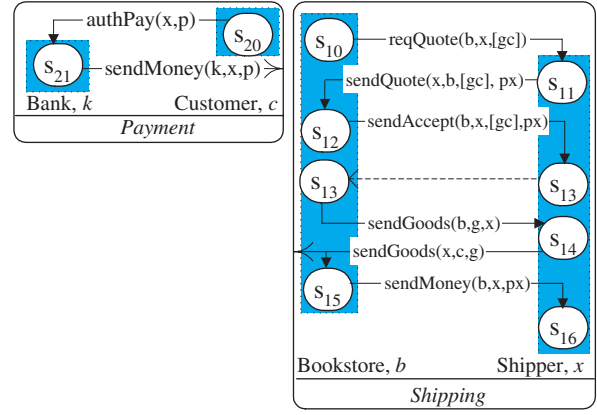

Fig. 2.   A shipping and a payment protocol

V. DESIGNING PROTOCOLS VIA AGGREGATION

Protocols can be developed by aggregating smaller protocols to implement stages. For example, the Purchase protocol, at a high level of abstraction, is implemented by a protocol that has three stages: an initial *negotiation* stage, followed by a *shipment* stage, and finally a *payment* stage. In the purchase scenario shown in Figure 1, states s_0, s_1, s_2, and s_3 belong to the negotiation stage, states s_3 and s_4 belong to the shipment stage, and states s_4 and s_5 belong to the payment stage. For simplicity, let us adhere to the negotiation-shipment- payment order even though more flexible protocols might allow the payment to precede the shipping. This three-stage protocol can be refined, for example, by substituting, or *splicing into* the purchase protocol, any of the shipping protocols available. One can ship via regular mail or use return-receipt mail. The splicing works because the purchase protocol is specified as an interface, and the shipping protocol adheres to the interface. For example, the shipping protocol in Figure 2 can be spliced into *Purchase* as shown in Figure 3. Similarly, the payment stage, which is also shown in Figure 2 can be substituted for by the *Payment* protocol. Further, the simple Purchase protocol subsumes the resultant protocol.

One important observation to make is that a protocol that is spliced into another might itself be spliced by the second protocol. In our example above, the Shipping protocol splices the Purchase protocol. However, the shipper is paid by the bookstore only after the customer has paid the bookstore. Therefore, the Shipment protocol has essentially be spliced in between its stages s_{14} and s_{15}.

Enabling Splicing: In realistic settings, e-business protocols will be refined by splicing to enable the participants' existing processes to interoperate seamlessly. As a guide, the following are to be borne in mind when designing a protocol.

- A protocol can be spliced into by another if the contractual relationships between the participants in each protocol are preserved.
- In most cases a refined protocol differs from the original only in terms of the creditor or the debtor of commitments that are made in the protocol. State similarity functions
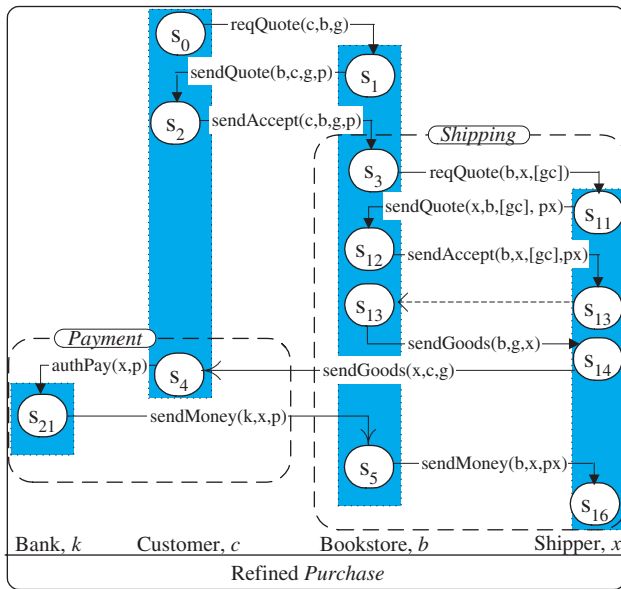
Fig. 3. Refinement of purchase by splicing in shipping and payment

therefore need to compare states respecting the commitments made by or to the participants.

- The interface definition of a protocol should specify states between which another protocol could potentially be spliced in. For example, the purchase protocol interface would expose states s_3 and s_4 as the states between which the goods should be delivered.

Commitment-based protocol design helps reason about legal and illegal splicing easily since commitments have a clear operational semantics across domains.

VI. DISCUSSION

We introduced a technical approach for modeling protocols that provides a natural basis for principled methodologies for designing custom protocols for e-business. The main contributions lie in the formalization of protocol specialization and aggregation. This can further be employed to perform subsumption reasoning and to carry out more interesting operations on protocols, such as splicing.

A. Literature

Yolum and Singh [6] and Fornara and Colombetti [9] highlight the benefits of a commitment-based approach to interaction protocol design. Johnson *et al.* [10] 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 from an application point of view and relates better to the Web Services approach.

The MIT Process Handbook [11] is a project that aims to create a hierarchy of commonly used business processes. Based on this hierarchy, Grosof and Poon [12] develop a system to represent and execute business rules.

The Web Services related standards for process composition and interoperability [13], such as the Web Services Choreography Interface (WSCI) are lower level abstractions than ours since they specify flows in terms of message sequences.

B. Directions

We introduced a methodology above, but design tools for applying this methodology would considerably enhance its power. Further, other methodologies based on the above semantics may conceivably be invented. Moreover, the above offers an abstract characterization of protocols. It would help to relate the semantics to more concrete forms of reasoning.

It would help to develop a taxonomy and rules of thumb for dealing with the choice of a protocol refinement that a participant can use to maximize its benefit. A designer may use such rules of thumb to specify a desired composite protocol and a participant may use such rules of thumb to seek out or negotiate for particular refinement based on its needs. A natural challenge is to develop an taxonomy geared toward protocols analogous to the taxonomy of business processes described in the MIT Process Handbook [11].

REFERENCES

[1] M. N. Huhns, L. M. Stephens, and N. Ivezic, "Automating supply-chain management," in *Proceedings of AAMAS-2002*. ACM Press, July 2002, pp. 1017–1024.
[2] "Rosettanet," www.rosettanet.org.
[3] C. Castelfranchi, "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*, 1993.
[4] M. P. Singh, "Agent communication languages: Rethinking the principles," *IEEE Computer*, vol. 31, no. 12, pp. 40–47, Dec. 1998.
[5] M. Verdicchio and M. Colombetti, "Commitments for agent-based supply chain management," *ACM SIGecom Exchanges*, vol. 3, no. 1, pp. 13–23, 2002.
[6] P. Yolum and M. P. Singh, "Flexible protocol specification and execution: Applying event calculus planning using commitments," in *Proceedings of AAMAS-2002*. ACM Press, July 2002, pp. 527–534.
[7] M. P. Singh, "An ontology for commitments in multiagent systems: Toward a unification of normative concepts," *AI and Law*, vol. 7, pp. 97–113, 1999.
[8] M. A. Sirbu, "Credits and debits on the Internet," *IEEE Spectrum*, vol. 34, no. 2, pp. 23–29, Feb. 1997.
[9] N. Fornara and M. Colombetti, "Defining interaction protocols using a commitment-based agent communication language," in *Proceedings of AAMAS-2003*. ACM Press, July 2003, pp. 520–527.
[10] M. W. Johnson, P. McBurney, and S. Parsons, "When are two protocols the same?" in *Communication in Multiagent Systems: Agent Communication Languages and Conversation Policies*, ser. LNAI, M.-P. Huget, Ed. Berlin: Springer-Verlag, 2003, vol. 2650, pp. 253–268.
[11] T. W. Malone, K. Crowston, and G. A. Herman, Eds., *Organizing Business Knowledge: The MIT Process Handbook*. Cambridge, MA: MIT Press, 2003.
[12] B. N. Grosof and T. C. Poon, "SweetDeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions," in *Proceedings of WWW-2003*, 2003.
[13] C. Peltz, "Web service orchestration and choreography," *IEEE Computer*, vol. 36, no. 10, pp. 46–52, Oct. 2003.