Protocols for Processes: Programming in the Large for Open Systems

Munindar P. Singh Amit K. Chopra Nirmit Desai Ashok U. Mallya
{singh,akchopra,nvdesai,aumallya}@ncsu.edu

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7535, USA

ABSTRACT

The modeling and enactment of business processes is being recognized as key to modern information management. The expansion of Web services has increased the attention given to processes, because processes are how services are composed and put to good use. However, current approaches are inadequate for flexibly modeling and enacting processes. These approaches take a logically centralized view of processes, treating a process as an implementation of a composed service. They provide low-level scripting languages to specify how a service may be implemented, rather than what interactions are expected from it. Consequently, existing approaches fail to adequately accommodate the essential properties of the business partners in a process (the partners would be realized via services)—their *autonomy* (freedom of action), *heterogeneity* (freedom of design), and *dynamism* (freedom of configuration).

Flexibly represented *protocols* can provide a more natural basis for specifying processes. Protocols specify *what* rather than *how*; thus they naturally maximize the autonomy, heterogeneity, and dynamism of the interacting parties. We are developing an approach for modeling and enacting business processes based on protocols. This paper describes some elements of (1) a conceptual model of processes that will incorporate abstractions based on protocols, roles, and commitments; (2) the semantics or mathematical foundations underlying the conceptual model and mapping global views of processes to the local actions of the parties involved; (3) methodologies involving rule-based reasoning to specify processes in terms of compositions of protocols.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; D.2.10 [**Software Engineering**]: Design; D.2.13 [**Software Engineering**]: Reusable Software; I.2.11 [**Artificial Intelligence**]: Distributed Artificial IntelligenceMultiagent Systems

General Terms

Standardization, Languages, Design

OOPSLA'04, Oct. 24-28, 2004, Vancouver, British Columbia, Canada.

Keywords

Open Systems, Interaction Protocols, Business Processes

1. INTRODUCTION

We think of business process modeling and enactment as a form of programming in the large [DeRemer and Kron, 1976]. Programming in the large is distinguished from the more common programming in the small in several ways. Programming in the large emphasizes putting together large software components, built by several people over a long period of time, and having their local state, whereas programming in the small is about developing the individual components. Programming in the large is a much more challenging engineering problem.

Business processes, which span multiple autonomous business partners, are an example of programming in the large. Cross enterprise processes, in particular, involve a rich variety of interactions among software components that are independently designed and configured, and which represent independent (and sometimes mutually competitive) business interests. Because Web services simplify interoperation, they have led to a resurgence of interest in technologies for process modeling and enactment.

Current approaches for the modeling and enactment of business processes are woefully inadequate, and reflect the similar inadequacy of programming in the large in open environments. Whereas the initial attempts at programming in the large were based on simple module interconnections, more serious attempts, such as megaprogramming, employed ontologies to address the heterogeneity of the information processed by various components [Wiederhold et al., 1992].

However, no good abstractions have been developed to model the interactive processing of information by multiple components. Consequently, processes are still specified today as scripts– providing, in essence, the same level of abstraction as developed in the job control languages (JCLs) of the mainframes of the 1950s. The above claim may sound unduly harsh, but while we readily acknowledge progress in technologies for process management, in terms of abstractions for modeling processes the progress is incremental at best. Whether you consider any of the modeling tools in use today, or leading proposed standards such as the Business Process Execution Language for Web Services (BPEL) [2003] or the Web Ontology Language for Services (OWL-S) [DAML-S, 2002], the process abstractions they offer are constructs such as sequence, iterate, fork, and join. In other words, the abstractions are little more than what you might find in a JCL. Hence our claim above.

But, still, why are such abstractions not adequate for processes? Imperative languages are abstractions that are best suited to pro-

gramming in the small. They assume the invoked components behave as expected. However, the main problems that arise in processes are problems of programming in the large. Because of environmental effects, exceptions can arise. Because the participants are autonomous, they can act to exploit opportunities, and their behavior may appear to be unexpected to their partners.

The effect of the above is that current approaches for process modeling and enactment have some key, well-known limitations in practice. They are either too rigid (thus frustrating users and causing systemic inefficiencies), or are extremely expensive in time and effort to construct and manage, and usually both. Thus some of the benefits of openness are lost.

1.1 Vision

Our diagnosis of the above challenges is that they reflect a fundamental problem for programming in the large. The traditional (imperative) scripting constructs specify flows. Clearly any process execution must correspond to a flow. However, this does not mean that we must specify the set of flows explicitly. Instead, we propose that processes be captured in terms of protocols, where each protocol is a flexible encoding of a meaningful set of interactions.

We define *(business) protocols* as publishable specifications of business interactions. We propose to model a business process as a composition of protocols. For example, we can have protocols for negotiation, for payment, for selecting a shipping company, and so on. A protocol is an interface, meaning that it specifies only the key desired aspects of the interactive behavior, not how the interacting parties are implemented.

Each protocol constrains the business partners involved in it. Protocols are modular, i.e., functionally decentralized. For example, a payment protocol between a customer and a merchant would be specified independently of the merchant's inventory fulfillment protocol for ordering goods from its suppliers. Thus, capturing processes via protocols enables us to more easily represent and enact interactions among autonomous business partners.

To model a process, we first identify the protocols using which the different participants interact. For example, a merchant and a customer may interact with each other using a negotiation protocol; the merchant, customer, and payment agency may interact via an escrow protocol; and, the merchant, customer, and shipper may interact through some specialized logistics protocol. When each participant acts according to its local reasoning but respecting the stated protocols, a multiparty business process is enacted but without a global flow necessarily ever having been explicitly encoded.

1.2 Benefits

Our protocol-based approach offers the following natural advantages. One, for process *design*, protocols are naturally reusable whereas complete processes are not. More importantly, protocols lend themselves to modeling abstractions such as specialization and aggregation. Two, for process *enactment*, when protocols are flexible, they enable each party to exercise some discretion in applying its local policies or preferences while obeying a protocol. For example, a merchant may accept only cash for discounted goods and a customer may prefer to pay for goods after using them for a month. This flexibility also enables us to capture and handle business exceptions and opportunities in a natural manner at the level of protocols. Three, for process *monitoring*, protocols provide a clean basis for determining that the interacting parties are complying with the given protocols.

1.3 Trends

Just as network protocols enabled the expansion of the lower layers of the Web architectures, business protocols will enable the development of processes involving autonomous, heterogeneous business partners. For this reason, we expect to see an increasing set of business protocols to be published and custom protocols to be designed. Several business protocols have been defined. Some general-purpose ones are NetBill [Sirbu, 1997], Secure Electronic Transactions (SET) [2003], Internet Open Trading Protocol (IOTP) [2003], and Escrow [2003].

RosettaNet is a leading industry effort, involving about 400 electronics and telecommunications companies. The RosettaNet [1998] Partner Interface Processes (PIPs), of which 107 are currently listed, are business protocols in spirit. These modularly describe several important business interaction scenarios. RosettaNet is in active production use with several billion dollars worth of commerce being conducted over it, e.g., [Krazit, 2002]. Another major industry effort is ebXML [2002]. ebXML is similar to RosettaNet but more general in style. Some of RosettaNet's components are gradually shifting over to using ebXML, e.g., for messaging formats. ebXML's Business Process Specification Schema (BPSS) describes partner roles and the documents they would exchange. RosettaNet's PIPs map to instances of BPSS. ebXML's Collaboration Protocol Agreement (CPA) describes an agreement (including conversations) between collaborating parties that is derived from their individual profiles.

We find the above trends, along with the well-known expansion of service-oriented computing, extremely encouraging. These trends clearly suggest that industry has understood the problem of developing cross-enterprise information systems, and is showing us researchers the way (in terms of what is important). However, current integration efforts are tedious and expensive, because they require extensive hard-coding. RosettaNet's limitations include that the PIPs specify interactions rather rigidly and do not offer a formal semantics. Further, the PIPs seem to specify some internal operations of each partner. Lastly, the interactions are short (in fact, just involving two parties and typically no more than a request and a response pair) and exceptions, where accounted for, are encoded as separate PIPs. ebXML has the same limitations.

Consequently, although current specifications of the business protocols are lacking in some respects, an increasing set of such protocols is an indication of the significance of our approach. The main reason behind the above limitations is that while business processes apply among autonomous, heterogeneous partners, the programming abstractions are still based on closed systems.

1.4 Key Traditional Approaches

Section 3.2 discusses the related literature in more detail, but it would help to outline the key approaches here. Conventionally, a business process is modeled in a conceptually centralized manner as a global flow. Emerging standards and tools support the specification and enactment of business processes specified as flows, but they cannot escape the fundamental limitations of such representations. One, it is difficult to create and maintain flows: they become complex in the face of exceptions and cannot easily be verified. Two, because a flow represents a central view, it inevitably limits the autonomy and heterogeneity of the business partners involved, leading to a suboptimal treatment of exceptions and opportunities. Three, flow representations are about how a process or composed service is implemented; what we need are interfaces that assure us that independent implementations will interoperate.

It helps to distinguish *orchestration* (how a process is implemented by composing services) from *choreography* (how services interact) [Peltz, 2003]. Both BPEL [2003] and OWL-S [DAML-S, 2002] emphasize orchestration by encoding flows. Service compo-

sition has drawn an increasing amount of attention lately. However, work in this area has concentrated on orchestrating services so as to accomplish a desired composition. For example, McIlraith *et al.* [2001] and Cardoso and Sheth [2003] show how composed services may be put together. This body of work addresses how a composed service may be implemented, not how an autonomous service would interact with other services.

By contrast, the Web Services Choreography Interface (WSCI) [2002] describes "conversations" or constraints on how a service is willing to interact with others, e.g., by requiring login before purchase. Unfortunately, existing choreography approaches rigidly specify a series of message exchanges but without an account of how they could be modularized and combined and how they could be related computationally to the orchestration approaches.

Like choreography, a protocol describes how services interact, but unlike traditional choreography, a protocol would consider the perspective of the interaction rather than of a particular participant. Further, protocols are conceptually composable to yield processes. Lastly, protocols would map into the flows of the participants, where the flows would interact to yield the desired process, thereby fulfilling the purpose of orchestration as well.

1.5 Why is this an Onward! Paper

Programming in the large as a topic has been around for almost three decades. Business process management has seen a phenomenal resurgence in interest as it has expanded into cross-enterprise settings to serve the needs of e-business. The challenges the open environments pose for process modeling and enactment call for new approaches for software development. We provide a promising such approach. This paper outlines several technical aspects of this approach, indicating that it could be a promising new paradigm for scientific contributions. However, the results might need further refinement before they are ready for conventional forums.

1.6 Organization

The rest of this paper is organized as follows. Section 2 describes our motivations in greater detail, presents our technical framework, and introduces our approach. Section 3 describes our contributions in relation to the most relevant literature.

2. APPLYING PROTOCOLS

The above idea, of basing processes on protocols, leads to a series of technical challenges. The objective of our ongoing research is to address these challenges in the manner described below.

- How can we compose protocols into processes? How can we reconcile a global perspective on a process with the local perspectives of the participants? For example, a supply chain is a process whose overall workflow is just one view; equally important is understanding the participant's perspectives, which govern their individual actions.

  Approach. Develop a formal model for protocols that supports refinement and aggregation, and to further develop it into a formal model of processes. For example, the generic payment protocol can be refined into a rich variety of protocols, each with different tradeoffs of expense, speed, and convenience (for one or more of the parties involved). Examples include handing over cash, paying with a credit card, paying with a debit card, paying with a personal check, paying via a third party such as Paypal, and so on. 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.

- How can we support the configuration of processes in a manner that respects the local policies of the participants? How can we manage implementations of protocols? For example, a seller may at its discretion waive checking a buyer's credit history, even though the process allows it or, conversely, insist on checking a buyer's credit history even if it is not explicitly required. That is, it should be possible to modify local policies and to negotiate additional requirements.

  Approach. Employ agents, defined in a broad sense, to represent the interests of autonomous partners and to carry out extended interactions with each other. The agents would not only respect the protocols, but also apply the local policies and preferences of the participants.

- How can we enact such processes in a distributed manner using current techniques (which are not geared toward protocols)? In particular, how can process enactment support recovery for a protocol while guaranteeing some transactional properties and how can we determine compliance of the participants' behavior?

  Approach. Compile protocols into skeletal flows that can be completed based on local policies and executed using conventional business flow engines and rule engines. The flows would produce and consume messages to yield the desired choreography. They would determine if any of the protocol requirements were being violated. Compliance verification is harder when flexible actions are allowed.

We do not directly study how individual protocols are invented. Our other work deals with the specification and verification of flexible protocols based on a nonmonotonic logic formalism [Chopra and Singh, 2003; Yolum and Singh, 2003]. That work addresses a limitation of conventional techniques for developing protocols, by improving their flexibility. For our present purposes, we can assume that protocols are available and specified in a monotonic formalism, although possibly derived from nonmonotonic representations. We do represent and reason about protocols.

2.1 Conceptual Model and Definitions

Figure 1 shows our conceptual model for a protocol-based treatment of business processes. The abstract entities (sharp rectangles) are publishable interfaces, which must be implemented to yield configurable entities (rounded rectangles) that can be fielded in a running system.

An *agent* is an implementational entity, something that can represent a real-world business partner with its own local business rules and configurations. For our purposes, agents are persistent computations capable of forming commitments with other agents. Agents naturally represent parties such as the partners involved in a business scenario, who might collaborate but retain their autonomy. There is no assumption of any of the aspects of agents traditionally associated with artificial intelligence.

A business *protocol* is a specification of a logically self-contained interaction, i.e., an interface that can be captured in a registry. A protocol involves two or more roles and is specified via some protocol logic. The protocol logic is strong enough to encode the contractual obligations of the various parties and is formalized through the notion of commitments, which we introduce in Section 2.2.1. A role's behavior for participating in a protocol can be extracted from the protocol logic into a protocol-specific skeleton or *P-Skel*. When a role plays more than one roles, its different P-Skels must be combined and reconciled into a composite skeleton or *C-Skel*. Protocols can also be aggregated from other protocols, but for simplicity we don't emphasize such relationships here.

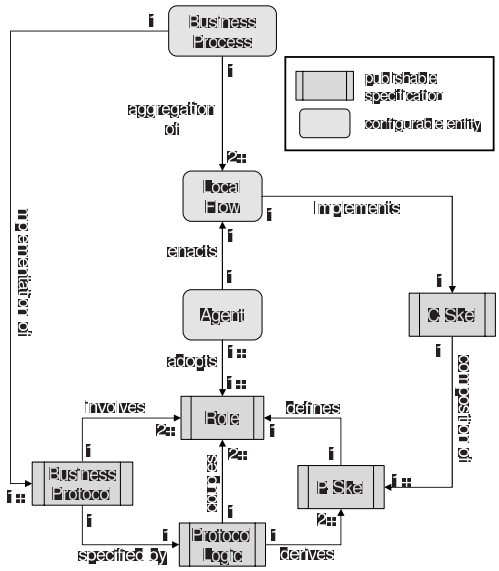

Figure 1: Conceptual model

An agent may participate in multiple business protocols by *adopting* a role in each of them, e.g., a bookstore may adopt the role of a seller while interacting with customers and the role of a buyer while interacting with publishers. A *local flow* is an executable realization of an agent's view of its tasks for its various protocols and incorporating its local business logic. Such a flow would invoke the right services in order to realize the C-Skels that it implements.

A *business process* aggregates the local flows of the agents participating in it. For example, a supply chain business process would be the aggregation of the local flows of the consumer, the retailer, the supplier, and the manufacturer, each flow incorporating the policies of its respective party. The contractually required parts would have been encoded in the protocols that are employed; the other parts may only be in the policies of the participants and would generally not be visible externally.

2.2 Representations

We briefly introduce our key representations, which are geared to supporting reasoning about protocols and processes as motivated above. To talk about how a protocol specializes or aggregates one or more protocols presupposes that we can characterize the computations allowed by a protocol and the evolving states of those computations so that we can consider whether a particular refinement or detour is legitimate. For business protocols, therefore, this means 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.

2.2.1 Commitments

The contractual relationships of interest are naturally represented through *commitments*. A legal notion of contracts was identified by Hohfeld [1919]. Commitments capture the directed obligations of one party to another and cover Hohfeld's notions [Singh, 1999]. In simple terms, a commitment is a directed obligation from an agent (the *debtor*) to another agent (the *creditor*) regarding a particular *condition* that, in effect, the debtor promises to bring about. 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. Using commitments enables us to model not just a participant's actions, but also how the actions advance the ongoing business interaction, which enables us to more readily detect and accommodate business exceptions and opportunities.

Commitments lend coherence to the interactions because they enable agents to plan based on the actions of others. Commitments can be manipulated through a small set of operations, including create, discharge, cancel, release, assign, and delegate [Singh, 1999], which we lack the space to discuss here.

In principle, violations of commitments can be detected and, with the right architecture, commitments can be enforced—by penalizing agents who do not comply with their commitments. Enforceability of contracts is necessary in practical settings where the participants are autonomous and heterogeneous [Singh, 1998].

2.2.2 Branching Model of Time

We use a branching model of time to represent the evolving state of a system. Each point in the model corresponds to a snapshot of the possible state of the system. Like in traditional applications of temporal logic, the state of the system is identified by the atomic propositions that are true therein. However, we go beyond traditional approaches in considering the commitments among the different participants of the system, and the operations on commitments that they continually perform as they enact various protocols.

Except for the "interpretation" (in the mathematical logic sense of the term), our model of time is a standard one [Emerson, 1990]. The language we use is the well-known Computation Tree Logic (CTL). This is helpful in enabling us to benefit from traditional tools for temporal logic.

We describe our model using the following example scenarios that can arise during a book purchase interaction. Each of these scenarios requires a different amount of effort from the participants in terms of protocol execution, planning, and coordination. In these scenarios, *customer* or *c* represents the customer's agent, *bookstore* or *b* the bookstore's agent, *bank* or *k* the bank's agent, and *shipper* or *x* the shipper's agent. Circles represent states, labeled by s_i, and arrows are labeled by the messages passed between the 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 he wishes to buy, and upon receiving a quote from the bookstore, would accept the bookstore's offer. The bookstore would then ship the book, after which the customer would send the payment. This is modeled after the NetBill protocol [Sirbu, 1997].

2. The bookstore might be willing to give a refund if the customer returns the book for some reason. This scenario is similar to the above scenario until the stage where the book is delivered to the customer. However, this scenario has more steps, since the customer returns the book, and terminates only when the bookstore sends the refund to the customer.

3. The customer might delegate 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. The customer, after accepting the bookstore's price quote and later receiving the book, sends a message to both the bookstore and the bank (although only one arrow is shown in the figure, between states s_4 and s_{21}) indicating that the bank will honor the customer's commitment to pay.

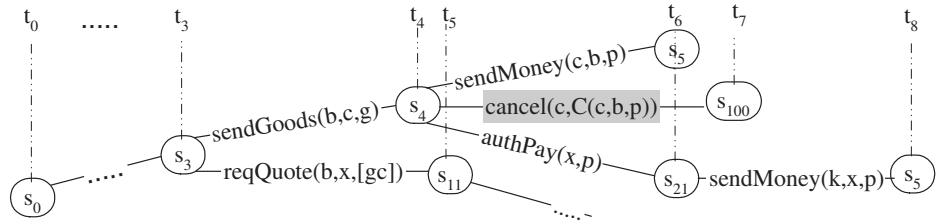

Figure 2: Branching model corresponding to the book purchase example

Figure 2 shows how the evolution of our scenarios can be represented in branching time. States s_1 and s_2 are not shown for the sake of brevity. The t_i's represent instants of time increasing from left to right. Each path from the root s_0 of the tree to a leaf node denotes one *computation* of the system. The computation $s_0, \ldots, s_3, s_4, s_5$ is a valid one, corresponding to scenario 1, while the computation $s_0, \ldots, s_3, s_4, s_{100}$ is invalid, because the customer cancels his commitment to pay after receiving the book.

2.3 Protocol Semantics and Design

Figure 3 shows two simple protocols, also based on the book purchase example. The protocols show states and transitions.

2.3.1 Modeling protocols

Information modeling involves the application of some key abstractions such as classification, aggregation, and relationships among components. However, traditional process models don't readily support such abstractions. Because they are based on data and control flow, they correspond to flow abstractions (e.g., sequence, branch, and so on). These abstractions take the perspective of one party whereas business processes are inherently distributed among autonomous entities. In essence, whether a given process is a specialization or a generalization of another process is not easy to decide, especially because the contributions of a given partner may be radically different in the two processes.

By contrast, because protocols are modular, they enable abstractions that are on par with the classification, aggregation, and relationships for information modeling. Protocols can be defined based on other protocols and they can be combined to yield a desired process. Instead of comparing entire processes, we can compare their protocols. For example, a process may use a more general payment protocol and a less general shipping protocol than another process.

2.3.2 Protocols and computations

In essence, each protocol allows a set of computations. More general protocols allow more computations than the protocols that refine them. In simple terms, if we begin with a protocol and eliminate even one of the computations in it, we would have produced a refinement of it. However, in practice, the refinement of a protocol involves more than just selecting some of the computations it allows. Typically, the refinement involves that additional constraints and wrinkles be added to the computations. For example, a simple payment protocol might require that the payer transfer funds to the payee. A particular refinement of this might be through payment with a check. To pay with a check, the payer would send a check to the payee who would deposit the check to his bank, which would present it to the payer's bank, which would (presumably after debiting the appropriate amount from the payer's account) send the funds to the payee's bank, which would make those funds available to the payee. Thus payment by check is a specialization of payment, but it involves a lot more steps. We know it is a refinement of the payment protocol because, just as the payment protocol requires, it ends up transferring funds to the payee. That is, the commitments at critical states in the two protocols line up correctly.

In general, the refinement of protocols can involve a mix of shrinkage along the dimension of the number of computations and expansion along the dimension of the number of steps in an allowed computation. Protocols that allow many short computations are the most flexible because there are numerous ways to satisfy such protocols. Protocols that allow fewer computations limit the ways in which they can be satisfied. Protocols that allow a few long computations are the most restrictive because there are only a few ways to satisfy them. Protocols that allow many long computations are more flexible than the above, because they offer more alternatives.

Protocols with a few long computations restrict the autonomy of the participants. Such protocols are not necessarily bad, though. They have a distinct advantage in checking compliance. In an extreme case, if a protocol allows exactly one sequence of actions, you will know immediately if it has been violated when one of the expected actions does not occur. There is thus a tradeoff between the flexibility of protocol execution on the one hand and the complexity of verifying compliance on the other.

2.3.3 Protocols and commitments

An advantage of incorporating commitments is that they directly represent contractual relationships, are flexible, and lend coherence to the otherwise disjointed interactions of the participants in a process. The formalization of the specialization and generalization hierarchy of protocols is made more interesting and useful because of the presence of commitments and roles in our model. Instead of considering uninterpreted sequences of states, we can consider how the commitments of the various roles evolve over different computations. The use of commitments enables more sophisticated business reasoning than in traditional approaches. In particular, it enables us to characterize the similarity of states and subsumption among protocols in potentially subtle ways. For example, we can look beyond the messages exchanged to the commitments of the participants. An example is when a participant from its local perspective considers two states as interchangeable simply because it features as the creditor and debtor in the same commitments regardless of the other parties. In other words, instead of merely considering raw computations, we would often need to normalize them in terms of commitments so as to make more precise judgments about how protocols relate to one another.

2.4 Designing Processes

Let us consider a simple but effective methodology for process design. This can be considered a top-down methodology. First,

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, 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 C(c, b, p) to k.

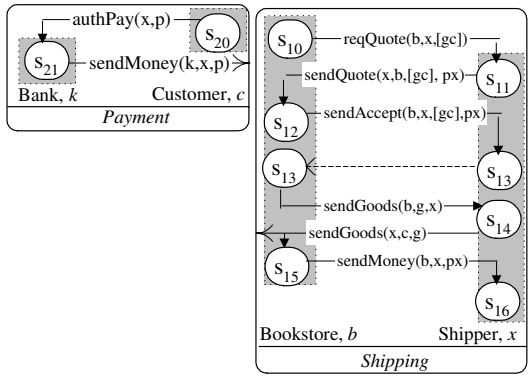

Figure 3: A shipping and a payment protocol

model a process in terms of a short sequence of abstract actions, which take the process execution from one *stage* to the next. Processes are refined by using protocols to implement stages. This is where it helps to have a library of protocols, organized in a taxonomy according to the above discussion.

Consider *purchase* again. At a high level of abstraction, this example is implemented by a process that has three stages: an initial *negotiation* stage, followed by a *shipment* stage, and finally a *payment* stage. In the *purchase* scenario, 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.

The first motivation is simplifying design through abstraction and reusability. However, it also potentially yields an improved flexibility for the participants, because the protocols more clearly delineate what a participant should do and thus leave more choices to be made by each. For instance, an alternative process that allows the payment stage to precede the shipping stage essentially achieves the same goal as *purchase*, that of exchanging goods for money.

For simplicity, let us adhere to the following order for the purchase process: negotiation, then shipment, then payment. This three-stage process can be refined, for example, by substituting, or *splicing in*, any one of the numerous shipping protocols available. One can ship via regular mail or use return-receipt mail. Whatever the shipping protocol used, the splicing works because *shipping* is specified as an interface, and the shipping protocol adheres to the interface. For example, the shipping protocol of Figure 3 can be spliced into *purchase* as shown in Figure 4. Similarly, the payment stage can be substituted for by the *payment* protocol of Figure 3.

2.4.1 Role of semantics

In general, protocols may be combined in the above way only because the underlying representation of commitments tells us whether the combination is sound. If there were no underlying content, any modification to the protocol would be fraught with risk of unsoundness. A protocol can be spliced into by another if the contractual relationships between the participants in each protocol are preserved. That is, the commitments of the parties involved are not affected except as desired. Commitments help reason about legal and illegal designs since commitments have a clear operational semantics across domains.

Armed with the semantics, we can support operations that are richer than the case where one protocol is nested within another (i.e., becomes a subprotocol of the other). Specifically, when a protocol p_2 is spliced into a protocol p_1, p_2 itself might be spliced into by parts of p_1. As a particular case, consider *shipping*, a protocol that takes a circuitous run to achieve the delivery of the goods to the customer. However, the shipper is not paid by the bookstore until the customer has paid the bookstore. Therefore, *shipping* has essentially been spliced in between stages s_{14} and s_{15}. From the point of view of *shipping*, the bookstore waits at state s_{15}. From the point of view of *purchase*, the customer receives the goods, and performs the payment via the bank.

In this manner, our approach supports the design abstractions of refinement (specialization and generalization) and aggregation (substituting a protocol for a place-holder stage or for another protocol). It also supports relationships among protocols where one protocol achieves the state that is needed by another to make further progress. This leads to a subtle kind of aggregation where two protocols might even be interwoven so as to make the best progress in the given process. Such aggregations, in general, would require rules to capture the local behaviors of the participants.

2.5 Enacting Processes

An important challenge is to enact processes in a distributed manner based on the stated protocols. As explained in Section 2.1, each protocol yields a P-Skel for each of the roles in it.

In preliminary work, we have prototyped a tool to generate local skeletal flows of the roles in a protocol. These skeletal flows can be augmented with business rules to produce local flows that each participant can execute.

To show how our approach could be layered above the current flow-centric infrastructure, we render these flows in BPEL [2003]. We expose the protocol interfaces of each participant as services via the corresponding WSDL specifications. An example is discussed next.

Consider a finite state representation of NetBill as shown in Figure 5. Figure 6 shows the BPEL skeletal flow of the customer generated from the above protocol representation. A similar skeleton is generated for the merchant along with WSDL specifications for both roles. These generated flows are complete except for the business rules to handle each message received from other participants. We use Collaxa BPEL Orchestration Engine 2.0 [2003] to deploy, instantiate, and execute the BPEL flows of each role. The above

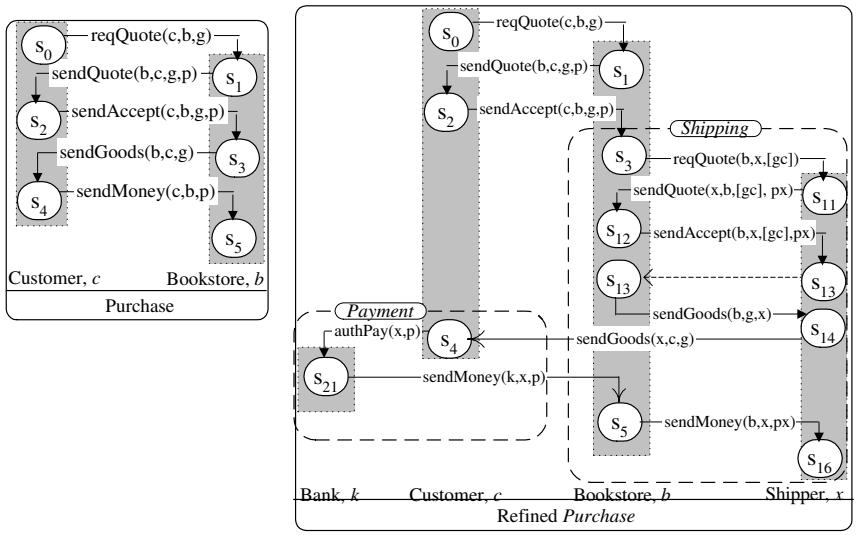

Figure 4: Refinement of purchase by splicing in shipping and payment

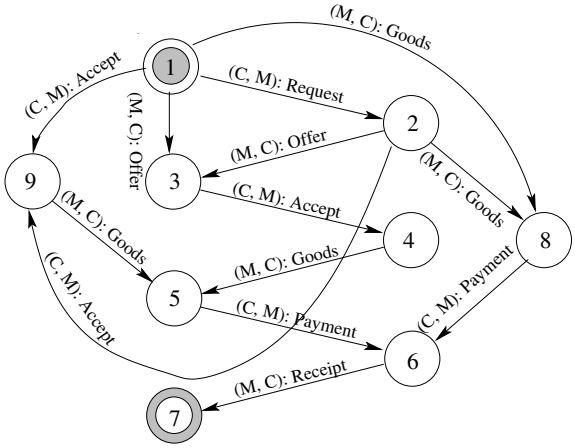

Figure 5: NetBill Protocol

tool is limited in that it doesn't accommodate the commitments of the participants.

3. DISCUSSION

We introduced a technical approach for modeling protocols that provides a natural basis for principled methodologies for designing custom business protocols. 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. Related work can broadly be classified under the following research areas.

3.1 Existing or Emerging Standards

Among conventional standards, BPEL [2003] primarily captures a flow model. BPEL also includes a significant component of data handling. BPEL also includes so-called protocols, which are modeled as processes whose variables are bound late to values. WSCI describes conversations in which a given service may participate. WSCI [2002] conversations are not business protocols. Protocols impose interrelated requirements on all participants, not just on the users of a particular service. Each WSCI specification corresponds to a role in our scheme. WSCI Conversations are specified without any semantics, so that transitions and states cannot be reasoned about.

The transactional requirements of the composition could be partially derived from the nature of the composition. A leading approach for transactional support for Web service computations is described in the WS-Coordination [Cabrera et al., 2003] and WS-Transaction specifications [Cabrera et al., 2002], which has support for Atomic Transactions (ATs) and Business Activities (BAs). Most of our intended applications will fall under business activities. OASIS's Business Transaction Protocol (BTP) introduces an alternative, but similar, framework for coordinating Web transactions [Dalal et al., 2003]. BTP includes atom (all or none) and cohesion (application-specific) transactions. The Web Services Composite Application Framework (WS-CAF) [Bunting et al., 2003] is another industry initiative to support transactional properties of business processes. The above approaches are similar enough for our purposes. They take a flow-oriented stance on processes and attempt to encapsulate certain steps as transactions. Our approach would apply transactional properties at the level of protocols.

Semantic Web efforts have converged into the Web Ontology Language (OWL) [McGuiness and van Harmelen, 2003]. One of the best current works on semantic Web services is OWL-S (derived from DAML-S) [DAML-S, 2002]. OWL-S is an OWL ontology for services, which includes service grounding, service profile, and process model. The profile is key for specifying and discovering services. The process model describes how a service may be implemented in terms of a set of scripting constructs, such as for sequencing, concurrency, branching, and iteration. In this respect, OWL-S resembles BPEL, which describes processes with which services can be implemented as compositions of others. Although such process specifications may be useful in tools to implement

```
<process name="NetbillConsumer" …>
<partnerLinks>
  <partnerLink name="merchant"
     partnerLinkType="cns:merchantLT"
     myRole="netbillConsumer"
     partnerRole="netbillMerchant"/>
  ...
</partnerLinks>
<variables>
  <variable name="requestM"
     messageType="mns:RequestMessage"/>
  <variable name="offerM" …/>
  ...
  <variable name="state" …/>
</variables>
<sequence>
  <!--First basic activity must be "start"-->
  <receive partnerLink="initiator"
           operation="start"
           variable="startM"
           createInstance="yes" ...>
    ...
  </receive>
  <!-- start up logic -->
  <switch>
    <!-- decide among request/accept/wait for
      merchant -->
    ...
    <!-- update state accordingly-->
    ...
    <case condition="bpws:getVariableData(
      'state' ,'value') = X">
      <!--Take action: current state is X-->
      ...
      <invoke partnerLink="merchant" ...
         operation="..."
         inputVariable="...">…</invoke>
      <!--Change current state accordingly-->
      ...
    </case>
    ...
  </switch>
  <!-- P-FSM logic -->
  <while condition="…">
    <pick>
    <onMessage partnerLink="merchant" ...
      operation="receiveOffer"
      variable="offerM">
      <switch>
        <!--OfferM OK in current state ?-->
        <case condition="…">
          ...
          <invoke partnerLink="merchant"...>
          </invoke>
          <!-- Change current state -->
        </case>
        <case …/>
        …
        <otherwise …/>
      </switch>
    </onMessage>
    <onMessage partnerLink="merchant" ...>
      ...
    </onMessage>
    ...
    </pick>
  </while>
</sequence>
</process>
```

Figure 6: BPEL flow for NetBill customer

services, they are not directly suited for standardization. In our approach, some of this functionality shifts to protocol specifications.

3.2 Relevant Literature

3.2.1 Traditional formal methods

Current modeling formalisms, such as finite state machines and Petri Nets, originated in distributed computing and apply at lower levels of abstraction than needed for flexible business interactions [Emerson, 1990; Harel and Gery, 1997]. When applied to business protocols, these formalisms result in specifications that are overconstrained to the level of specific sequences of actions. However, the rigor of these approaches is attractive. Our approach employs temporal logic augmented with commitments to accommodate flexibility.

A recent work in this direction is on conversation protocols, e.g., [Bultan et al., 2003]. This body of work essentially models services as components that send messages to one another. It is a direct application of traditional formal techniques to services. This is valuable. However, this work is ultimately limited by fact of ignoring the challenges of business processes. In particular, because it lacks semantics in the form of commitments (or other model of contractual relationships), it has no account of flexible behavior by the participants. Without the benefit of such semantics, the refinement of protocols (not addressed by the cited paper) would at best consider sets of computations but would not recognize when syntactically distinct computations were actually alike. In simple terms, you don't know if a deviation is legitimate, because you don't know what the conversation means.

3.2.2 Semantic Web services

OWL-S and some semantic composition approaches were alluded to above. In the Web Services Modeling Framework (WSMF) [Bussler et al., 2002], the process implementing a given service is, in general, not exposed. However, WSMF enables a variety of constraints on the invocations of services, e.g., concurrent execution, data and control flows among services, and a compensation strategy for a service (what to do if it fails). In this sense, WSMF supports a process model similar to the OWL-S process model. Capabilities are modeled in terms of preconditions and postconditions to enable service discovery. WSMF also includes components of ontologies and mediators for data sharing. These are complementary to our present approach.

3.2.3 Process modeling

There has been an enormous amount of work on process modeling. Much of that work has developed languages for expressing flows and systems for enacting such flows. Of greater relevance intellectually is the MIT Process Handbook (MITPH) project [Malone et al., 2003], which aims to create an extensive classification and systematic organization of business processes based on two dimensions of process hierarchies, one that composes the *uses* of a process out of its constituent *parts*, and another that subclasses *generalizations* of a process into *specializations*.

Grosof and Poon [2003] develop a system to represent and execute business rules from MITPH. Wyner & Lee [2003] study specialization for data flow diagrams. Their approach can form the basis of the processes identified in MITPH. These concepts turn out to be complex and not readily applicable to entire business processes. Further, since Wyner & Lee do not capture the content through a high-level representation such as commitments, the results are not intuitive.

The pi-calculus [Milner, 1991] has recently been suggested as an approach for modeling processes, e.g., [Meredith and Bjorg, 2003]. This can be potentially quite useful, but only if applied at the level of interaction protocols. It is conventionally applied simply to en-

code orchestrations as in XLANG [Thatte, 2001] (now absorbed into BPEL) or to even to specify choreographies as in WSCI. However, even according to some of its proponents, some of the subtle features of the pi-calculus, e.g., reconfigurability, end up not being needed [Wischik, 2003]. A more challenging and potentially more valuable application of the pi-calculus would be in the context of business protocols as in our approach. Because protocols involve commitments and operations on commitments, they can potentially benefit from the advanced features of the pi-calculus, e.g., to type-check whether a given participant can play a certain role in a protocol and evolve its commitments in a certain manner. We defer this to future work.

3.2.4 Commitments and contracts

Commitments have been studied at length in the multiagent systems literature. Commitments fare better than traditional deontic logic because they are directed, whereas deontic logic only deals with what is obligatory or permissible and thus disregards an agent's obligations to another agent. Previous work on commitments has not been used for operational characterization of protocols. A notable exception is Verharen, who develops a contract specification language, CoLa, to specify transactions and contracts [1997]. Verharen's approach benefits from commitments in expressing actions, but it treats commitments as obligations, and does not allow manipulation of commitments as in our approach. Further, Verharen only considers base-level commitments, without capturing conditional commitments as we have done through metacommitments [Singh, 1999].

3.2.5 Commitment-based protocols

Yolum and Singh [2002] is one of the first accounts of the use of commitments in modeling agent interaction protocols and the flexibility that it affords the participating agents. Fornara and Colombetti [2003] describe how commitments relate to FIPA-ACL messages and demonstrate this with an example. Both approaches highlight the benefits of a commitment-based approach to interaction protocol design. Johnson *et al.* [2003] 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.

Bussmann *et al.* [2002] 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.

3.3 Contributions and Directions

To summarize, we have argued that the problems of programming in the large arise with renewed vigor in open environments, as exemplified by the world of cross-enterprise business process management. Current programming abstractions are ineffective for such settings. A simple, but promising, idea is to modularize interactions analogous to the way we traditionally modularize local behaviors: let's have roles and protocols as first-class entities just as classes and methods were in traditional programming. We described key elements of our research program, which in simple terms, seeks to take the above idea to its logical conclusion. We hope to have convinced the reader that a number of benefits can be derived from this exercise.

We don't claim that all important problems are solved or even framed with precision, but we think we are getting there. To this end, we have identified some avenues of research that are especially promising.

Design. Sophisticated process design tools based on our theory of processes and protocols would help designers create process models with greater productivity. Key questions are about design rules and sanity checks on processes and protocols.

Enactment. An execution framework for agents to play roles in different protocols so as to enact a process in a truly distributed manner would maximize the autonomy of business partners. It would eliminate the need for centralized process execution with its concomitant bottlenecks and failure modes.

Monitoring. Compliance verification is a key challenge for open environments. It becomes even more important when the participants are given more autonomy as in our approach. The richer semantics of our approach, in terms of commitments and temporal models, would help us develop stronger results than in other approaches.

Negotiation. We can frame the choice of protocol as a run-time decision to be made by the various participants. In particular, participants would then use the refined version of a protocol that best matches their local preferences. For example, a customer might choose to deal with a business that requires no authentication steps rather than with a business that offers similar services, but requires a number of authentication steps. In other cases, the parties involved may negotiate with each other to settle upon the particular refinement of a protocol that they would follow in their mutual interactions.

Overall, we believe this could prove to be a fruitful research program for our community.

4. ACKNOWLEDGMENTS

This research is supported by DARPA and by the National Science Foundation under grant DST-0139037.

References

BPEL. Business process execution language for web services, version 1.1, May 2003. www-106.ibm.com/developerworks/webservices/library/ws-bpel.

Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation specification: A new approach to design and analysis of e-service composition. In *Proceedings of the Twelfth International World Wide Web Conference (WWW)*, pages 403–410, 2003.

Doug Bunting, Martin Chapman, Oisin Hurley, Mark Little, Jeff Mischkinsky, Eric Newcomer, Jim Webber, and Keith Swenson. Web services composite application framework (WS-CAF), July 2003. http://www.iona.com/ devcenter/ standards/ WS-CAF/ WS-CAF.pdf.

Christoph Bussler, Dieter Fensel, and Alexander Maedche. A conceptual architecture for semantic Web enabled Web services. *ACM SIGMOD Record*, 31(4):24–29, December 2002.

Stefan Bussmann, Nicholas R. Jennings, and Michael Wooldridge. Re-use of interaction protocols for agent-based applications. In *Proceedings of the 3rd International Workshop on Agent-Oriented Software Engineering*, 2002.

Felipe Cabrera, George Copeland, Bill Cox, Tom Freund, Johannes Klein, Tony Storey, and Satish Thatte. Web services transaction (WS-Transaction), August 2002. http://www-106.ibm.com/ developerworks/ webservices/ library/ ws-transpec/.

Luis Felipe Cabrera, George Copeland, William Cox, Max Feingold, Tom Freund, Jim Johnson, Chris Kaler, Johannes Klein, David Langworthy, Anthony Nadalin, David Orchard, Ian Robinson, John Shewchuk, and Tony Storey. Web services coordination (WS-Coordination), September 2003. ftp://www6.software.ibm.com/ software/ developer/ library/ ws-coordination.pdf.

Jorge Cardoso and Amit Sheth. Semantic e-workflow composition. *Journal of Intelligent Information Systems (JIIS)*, 12(3): 191–225, November 2003.

Amit Chopra and Munindar P. Singh. Nonmonotonic commitment machines. In Frank Dignum, editor, *Advances in Agent Communication: Proceedings of the 2003 AAMAS Workshop on Agent Communication Languages*, LNAI. Springer-Verlag, 2003.

Collaxa. Collaxa BPEL orchestration server, 2003. http://www.collaxa.com/home.index.jsp.

Sanjay Dalal, Sazi Temel, Mark Little, Mark Potts, and Jim Webber. Coordinating business transactions on the Web. *IEEE Internet Computing*, 7(1):30–39, January 2003.

DAML-S. DAML-S: Web service description for the semantic Web. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*, July 2002. Authored by the DAML Services Coalition, which consists of (alphabetically) Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara.

Frank DeRemer and Hans H. Kron. Programming-in-the-large versus programming-in-the small. *IEEE Transactions on Software Engineering*, 2(2):80–86, June 1976.

ebXML. Electronic business using eXtensible markup language, 2002. Technical Specifications release, URL: http://www.ebxml.org/specs/index.htm.

E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. North-Holland, Amsterdam, 1990.

Escrow.com. Online escrow process, 2003. http://www.escrow.com/solutions/escrow/process.asp.

Nicoletta Fornara and Marco Colombetti. Defining interaction protocols using a commitment-based agent communication language. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 520–527. ACM Press, July 2003.

Benjamin N. Grosof and Terrence C. Poon. SweetDeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proceedings of the 12th International Conference on the World Wide Web*, pages 340–349, 2003.

David Harel and Eran Gery. Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42, July 1997.

Wesley Newcomb Hohfeld. *Fundamental Legal Conceptions as Applied in Judicial Reasoning and other Legal Essays*. Yale University Press, New Haven, CT, 1919. A 1919 printing of articles from 1913.

IOTP. Internet open trading protocol (IOTP), October 2003. IETF: Internet Engineering Task Force, http://www.ietf.org/html.charters/trade-charter.html.

Mark W. Johnson, Peter McBurney, and Simon Parsons. When are two protocols the same? In Marc-Philippe Huget, editor, *Communication in Multiagent Systems: Agent Communication Languages and Conversation Policies*, volume 2650 of *LNAI*, pages 253–268. Springer-Verlag, Berlin, 2003.

Tom Krazit. Intel conducts $5b in transactions via RosettaNet, December 2002. http://archive.infoworld.com/ articles/ hn/ xml/ 02/12/10/ 021210hnintelrose.xml.

Thomas W. Malone, Kevin Crowston, and George A. Herman, editors. *Organizing Business Knowledge: The MIT Process Handbook*. MIT Press, Cambridge, MA, 2003.

Deborah L. McGuiness and Frank van Harmelen. Web Ontology Language (OWL): Overview. www.w3.org/TR/2003/WD-owl-features-20030210/, February 2003. W3C working draft.

Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web services. *IEEE Intelligent Systems*, 16(2):46–53, March 2001.

L. Greg Meredith and Steve Bjorg. Contracts and types. *Communications of the ACM*, 46(10):41–47, October 2003.

Robin Milner. The polyadic pi-calculus: A tutorial. TR LFCS report ECS-LFCS-91-180, School of Informatics, University of Edinburgh, 1991.

Chris Peltz. Web service orchestration and choreography. *IEEE Computer*, 36(10):46–52, October 2003.

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

SET. Secure electronic transactions (SET) specifications, 2003. http://www.setco.org/ set_ specifications.html.

Munindar P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, December 1998.

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

Marvin A. Sirbu. Credits and debits on the Internet. *IEEE Spectrum*, 34(2):23–29, February 1997.

Satish Thatte. XLANG, Web services for business process design, 2001. www.gotdotnet.com/team/xml-wsspecs/xlang-c/default.htm.

Egon M. Verharen. *A Language-Action Perspective on the Design of Cooperative Information Agents*. Catholic University, Tilburg, Holland, 1997.

Gio Wiederhold, Peter Wegner, and Stefano Ceri. Toward megaprogramming. *Communications of the ACM*, 35(11):89–99, November 1992.

Lucian Wischik. Process calculi for Web choreography, March 2003. http://www.wischik.com/lu/ research/ lucian-piforweb-w3c-mar2003-handout.pdf.

WSCI. Web service choreography interface 1.0, July 2002. wwws.sun.com/ software/ xml/ developers/ wsci/ wsci-spec-10.pdf.

George M. Wyner and Jintae Lee. Defining specialization for process models. In *Malone et al. [2003]*, chapter 5, pages 131–174. MIT Press, 2003.

Pınar Yolum and Munindar P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 527–534. ACM Press, July 2002.

Pınar Yolum and Munindar P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence*, 8(I–II):47–71, 2003.