

# A Semantic Protocol-Based Approach for Developing Business Processes \*

Amit Chopra Nirmal Desai Ashok Mallya Leena Wagle Munindar P. Singh

{akchopra, nvdesai, aumallya, lvwagle, singh}@ncsu.edu

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

## ABSTRACT

A (business) protocol is a modular, public specification of an interaction among different roles that achieves a desired purpose. We model protocols in terms of the commitments of the participating roles. Commitments enable reasoning about actions, thus allowing the participants to comply with protocols while acting flexibly to exploit opportunities and handle exceptions. A policy is a (typically private) rule-based description of a participant's business logic that controls how it participates in a protocol. We propose that a business process be conceptualized as a cohesive set of protocols, and be enacted by agents playing specified roles in the protocols in which they participate. The agents would respect the given protocols while adhering to their local policies.

We propose OWL-P, a language for specifying protocols, and implement it using a multiagent architecture. We compile OWL-P specifications of protocols into skeletons for each role. Each skeleton corresponds to a set of rules with place-holders for policies. Developing an agent involves using the rules for its intended roles and supplying the necessary policies.

The key benefits of this approach are (1) a separation of concerns between protocols and policies in contrast to traditional monolithic approaches; (2) reusability of protocol specifications based on design abstractions such as specialization and aggregation; and (3) flexibility of enactment of processes in a manner that respects local policies while adapting continually.

This paper develops further results on a programming methodology through which agents can be implemented to realize desired processes. This methodology includes design patterns that ensure that agents built according to those patterns will be guaranteed to be compliant to the stated protocols.

## Categories and Subject Descriptors

[Service Computing & Applications]: e-Business; [Software engineering techniques for service-based development]: Service design principles; [Core service activities and technologies]:

\*This research was sponsored by NSF grant DST-0139037 and a DARPA project.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

Service composition; [Service & AI Computing]: Multi-agent based service models

## Keywords

Business Process, Service Composition, Multiagent Systems, Rule-based Systems

## 1. INTRODUCTION

Business processes typically span multiple *business partners* that are autonomous, and heterogeneous. Business processes involve complex patterns of interactions between the partners. These interactions are organized in the form of business protocols. In the past, business relationships were preconfigured and processes were customized and implemented to suit the partners, as in the Electronic Document Interchange (EDI) approach. However, the EDI approach is not conducive to the development of *open* business processes where partner relationships are developed on the fly. The autonomous and heterogeneous nature of participants poses difficult challenges to the development of such open business processes. This paper presents an approach of developing business processes (for open systems) based on the interaction protocols used and the policies of the partners.

Conceptually, a business process has two important elements, protocols that the partners use to interact, and business policies that drive the partners' enactment of the protocols. Protocols are specifications of interactions and represent the public part of the business process; for the process to be carried out effectively, the partners must adhere to the protocols. By contrast, policies are local to the partners; they capture the internal reasoning of partners. Protocols and policies are related in that a partner's policies drive the execution of the protocols it is participating in. For example, when a protocol allows a participant to choose from multiple actions (messages), the local policy of the participant decides which one to take. Similarly, policies also help decide the contents of the messages sent, and the processing of the messages received. The overall business process is realized as a result of the protocols between the partners.

Flexibility is an important consideration for business processes in open settings. Exceptions and opportunities routinely arise during the course of interactions in such settings. A business process will be flexible if the constituent protocols are flexible. Traditional specifications of protocols such as FSMs and Petri Nets specify a rigid sequences of interactions and lack a high-level semantics. A cornerstone of our approach is the use of commitments to give a declarative semantics to protocols [Yolum and Singh, 2002a]. A commitment is a directed obligation from one partner to another for achieving or maintaining a specified condition. Business pro-

protocols can naturally be seen as exchanges of commitments among the parties involved. Therefore, commitments represent an important ingredient of the semantics of business protocols. Flexibility in the protocol comes from reasoning about the commitments and taking actions accordingly.

We define an ontology for protocols using the Web Ontology Language (OWL) [OWL, 2004]. Our ontology is called OWL-P (OWL for Protocols). The ontology provides for concepts such as the roles in a protocol, the messages exchanged between the roles, and declarative rules that describe the effects of sending messages in terms of commitments. OWL-P rules can be converted into Jess rules [Jess] for execution and integrated with policies in a principled way. Our programming model is based primarily on rules. Rules lead to a declarative style of programming where the actual computations are inferred at runtime, thereby enhancing dynamic behavior.

From the software engineering point of view, the clear separation of protocols and policies offers certain advantages. Protocols can be reused across business processes. Protocols may not only be reused directly, they are also amenable to abstractions such as refinement and aggregation [Mallya and Singh, 2004]. However, in our programming methodology, protocol rules consult policies. The integration of policies and protocols at this level encourages a designer to think about the soundness of the business policies with respect to the protocol.

## Organization

Section 2 motivates our approach, lists our contributions and the scope of this work, and introduces the basic concepts and terminology. Section 3 describes our proposal for developing protocol specifications and policies. It also describes the system architecture, and the method of generating local flows from an OWL-P specification, with a sketch of correctness proof for the generated local flows. Section 4 compares our work with current research efforts in the area and charts out directions for future work.

## 2. MOTIVATION

Here, we describe an example of a business process developed with contemporary methodologies and tools, e.g., BPEL [BPEL, 2003] and list their shortcomings. Later sections contrast our approach with current trends to demonstrate the advantages of our approach. Figure 1 depicts a general procurement process where items to be purchased are already selected and the price has been agreed upon. The agents involved in the process are a Customer who wants to buy items, a Merchant who sells items, a Shipper who is a logistics provider, and a Payment Gateway who authorizes payments. The payment-related interactions are imported from the Secure Electronic Transactions (SET) standard [SET, 2003]. Empty circles in the flow of a participant represent the execution of internal business policies, whereas filled circles are the external interfaces through which the participants receive messages. Dark arrows represent the internal control flow. Thus, a sequence of dark arrows, empty circles, and filled circles (in some order) represents the local flow (local process) of the participant. When there are multiple out-edges from empty circles, all of the out-edges are executed in parallel. Since there are multiple participants possibly acting concurrently, the ordering of the messages shown is just one of the possible orders. For example, all the messages after message 8 (messages 9-17) could occur in any order.

Although this process is functionally correct and serves the purpose of its participants, its shortcomings are exposed when examined under the light of service-oriented computing (SOC) environments and open systems. Significant efforts to overcome these have

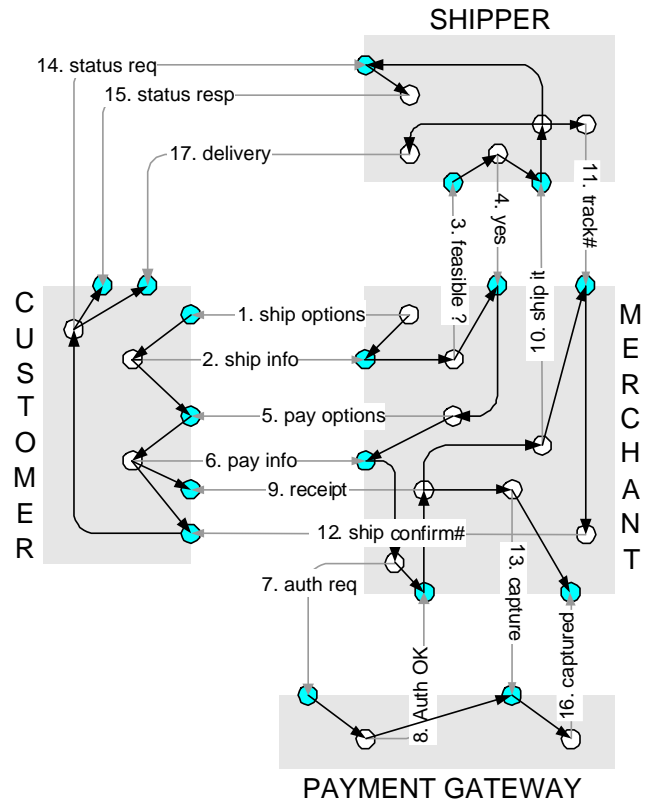


Figure 1: A Business Process

been made by the research community but with limited success. The outline in the earlier section hints at the solution we present. Section 3.6 explains in detail how our approach tackles the challenges of SOC environments.

*Lack of Semantics.* More often than not, local flows are independently developed by autonomous participants. Although the interfaces of the local flows are exposed in a standard language, e.g., WSDL [WSDL, 2002], no semantics is attached to these interfaces. If the business partners were to be discovered dynamically, as would be the case in open systems, it would be difficult at best to ascertain whether interoperability with them is possible.

*Lack of Reusable Components.* The local flows of the partners are not reusable. They are monolithic in nature, and formed by ad hoc intertwining of business logic and interactions. Since business logic is proprietary, flows of one partner are not usable by another. If a new customer were to participate in this SOC environment, its local flow would need to be developed from scratch. This is in spite of the fact that interaction patterns are generally reusable.

*Aiding Exception Handling.* Since in the merchant's local flow, interactions with the payment gateway may be intertwined with interactions with the shipper, a failure on part of one may affect the other. Ideally, as they serve distinct business purposes in the flow, the effect of exceptions of other flows should be minimized.

*Runtime Adaptation.* Suppose the merchant wishes to change the way it interacts with a customer (maybe because he is a special customer). Say the goods are to be shipped before the payment is

received from this customer. Now the local flow cannot adapt to this change at runtime. Also, the customer agent may no longer be able to interact correctly. Similar problems arise in the face of a change of business policy of an agent: it is not clear what updates must be made and where.

## 2.1 Contributions, Scope, and Significance

Our general contribution is a methodology and design principles for specifying flexible protocols and dynamic processes. Specifically, we show how employing modular protocols can result in reusability and ease of development. We present a novel conceptual model for business processes that allows us to decouple internal local policies of agents from their external interaction behavior. We show how commitments provide the basis for a semantics of the actions of the participants, thereby making the resultant protocols flexible and verifiable. We ground our concepts by introducing an ontology for specifying protocols termed OWL-P. By example, we show how our approach for process enactment is conducive to reuse and aggregation. Finally, we develop a prototype for enacting simple processes having one constituent protocol in a rule-based system.

This paper is a first step in realizing our vision. It emphasizes the modeling and software engineering aspects, and doesn't address technical challenges such as failure and recovery semantics for the protocols, versioning of the protocols and runtime compatibility verification of protocol skeletons. Further, it assumes synchronous communication. Lastly, it defers the composition of multiple protocol skeletons to future work.

The presented work is significant because it yields a new approach for the specification and enactment of business processes. Just as the standardization of network protocols enabled the expansion of the lower layers of Web architectures, business protocols will enable the development of processes for open systems. For this reason, we expect to see an increasing set of business protocols being published, and custom protocols to be designed in narrow domains. This will further increase the significance of our contributions for developing processes based on protocols and local policies.

## 2.2 Concepts and Terminology

Figure 2 shows our conceptual model for a protocols and policies based treatment of business processes. Boxed rectangles are abstract entities (interfaces), which must be implemented and combined with business policies to yield configurable entities that can be fielded in a running system (rounded rectangles). Abstract entities should be published, shared, and reused among the process developers. We specify a business protocol in terms of a set of rules termed *protocol logic*. Protocol logic encodes the interactions of participating *roles*, who would be bound to specific partners when a process consisting of the given protocol is enacted.

Whereas the protocol logic specifies the protocol from the global perspective, a *protocol skeleton (P-Skel)* specifies the protocol from the perspective of one of the participant roles. Thus, each P-Skel defines the behavior of the respective role with respect to the given protocol.

An *agent* is an implementational entity, representing a real-world, autonomous business partner with its local business rules. An agent may participate in multiple business protocols by adopting a role in each of them. For example, a bookstore may adopt the role of a seller while interacting with customers and the role of a buyer while interacting with publishers. When an agent needs to participate in multiple protocols, a *composite skeleton (C-Skel)* can be created by splicing in the P-Skels, one for each role that the agent plays in the

given protocols. For example, in a supply chain process, a supplier would be a merchant when interacting with a retailer in a trading protocol and would be a client in a shipping protocol for sending goods to the retailer. The C-Skel for such a supplier would be composed by splicing in P-Skels for a trading merchant and a shipping customer. This C-Skel specification could be published and then reused for developing other supplier processes.

An agent stipulates its internal business policies in terms of a set of rules we term as *business logic*. The *local flow* of an agent is an executable realization of a C-Skel. Local flow consults the business logic to make decisions. Thus, the combination of a C-Skel with business logic entails the desired local flow, which may be represented in a flow language, e.g., BPEL. A *business process* is the aggregation of the local flows of all the agents participating in it. Conversely, a business process is an implementation of the constituent business protocols.

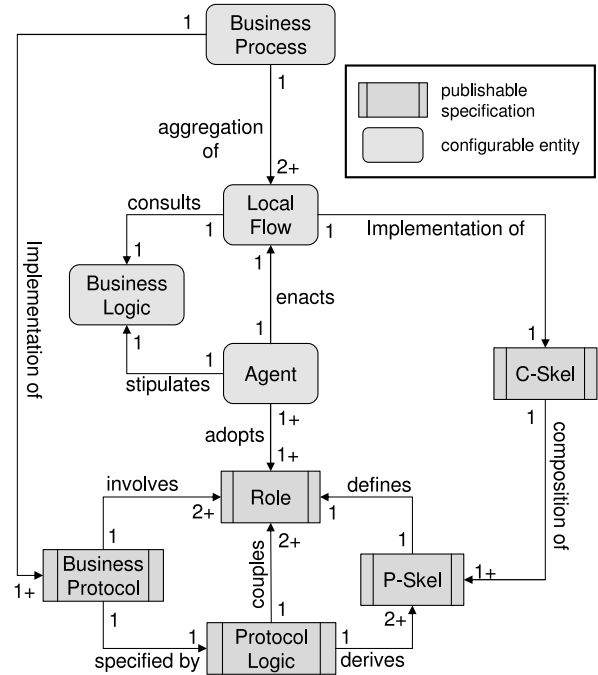


Figure 2: Conceptual Model

### 2.2.1 Commitments

To talk about how a protocol allows flexibility during enactment presupposes that we can characterize the computations allowed by a protocol and the evolving states of those computations so we can consider whether a particular refinement or detour is legitimate. For business protocols, therefore, this means that we must represent not only 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 gained importance in the field of multiagent systems [Castelfranchi, 1993]. Commitments capture the obligations of one party to another. For example, the customer's agreement to pay the price for the item after it is delivered is a commitment that the customer has towards the merchant. Using commitments enables us to model not only a participant actions, but also how they advance the ongoing business interaction,

which enables us to more readily detect and accommodate business exceptions and opportunities.

Commitments lend coherence to the agents' 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—by penalizing participants who do not comply with their commitments. Enforceability of contracts is necessary in practical settings where the participants are autonomous and heterogeneous [Singh, 1998].

To apply commitments presupposes that we are modeling the participants as *agents*. Agents naturally represent parties such as the business partners involved in an e-business scenario, who might collaborate but retain their autonomy. Commitments have been used to model interaction protocols, especially in e-business settings [Verdicchio and Colombetti, 2002, Yolum and Singh, 2002b]. Such formulations generally afford more flexibility to the participants in choosing the actions they may take at different stages. Much of our technical development is based on established concepts of distributed computing and temporal logic. Since commitments might be unfamiliar to some readers, we introduce them first.

**DEFINITION 1.** 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. The condition is expressed in a suitable formal language.

Commitments can also be *conditional*, denoted by  $CC(x, y, p, q)$ , meaning that  $x$  is committed to  $y$  to bring about  $q$  if  $p$  holds. For example, the conditional commitment  $CC(c, b, goods_g, pay_p)$  means that the customer  $c$  is committed to pay the bookstore  $b$  an amount  $p$  if the bookstore delivers the book  $g$  to the customer. When the bookstore delivers the goods, i.e., when the  $goods_g$  proposition holds, the conditional commitment  $CC(c, b, goods_g, pay_p)$  is automatically converted into the base-level commitment  $C(c, b, pay_p)$ .

### 2.2.2 Commitment Operations

Commitments are created, satisfied, and transformed in certain ways. The following operations are conventionally defined for commitments [Singh, 1999].

1. **CREATE(X, C)** establishes the commitment  $c$  in the system. This can only be performed by  $c$ 's debtor  $x$ .
2. **CANCEL(X, C)** cancels the commitment  $c$ . This can only be performed by  $c$ 's debtor  $x$ . Generally, cancellation is compensated by making another commitment.
3. **RELEASE(Y, C)** releases  $c$ 's debtor  $x$  from commitment  $c$ . This only can be performed by the creditor  $y$ .
4. **ASSIGN(Y, Z, C)** replaces  $y$  with  $z$  as  $c$ 's creditor.
5. **DELEGATE(X, Z, C)** replaces  $x$  with  $z$  as the  $c$ 's debtor.
6. **DISCHARGE(X, C)**  $c$ 's debtor  $x$  fulfills the commitment.

A commitment is said to be *active* if it has been created, but not yet discharged.

## 3. APPROACH

Protocols need to have a clear definition and their specification should use terminology that is understood by all the participants. The intent of drawing up an unambiguous protocol specification is to reap the benefits of standardization as has been done in other areas of computer science. Unlike networking protocols, business

protocols need to be specified at a higher level of abstraction so that the protocol designers are not bogged down by low-level details of the protocol execution and can focus on the business aspects. To further ease the job of business protocol designers, our framework separates the local business logic and other legal concerns from the public protocol. Such a separation of concerns makes for better engineered processes that enable reuse and are easy to understand. This section describes our approach for specifying flexible, commitment-based protocols and how we separate local policies from protocols.

### 3.1 Protocols

A protocol has a unique identifier. The protocol ontology defines the following elements, most of which are shown in Figure 3.

1. *Roles* that participate in the protocol. Each protocol has at least two roles.
2. *Messages* that the roles use to communicate with each other, along with message formats and meanings. The message formats detail various attributes that a message can have. For example, the `requestForQuote` message in the Netbill protocol, which is sent by the customer to the merchant asking for a price quote on a certain item, has one attribute for identifying the item, one attribute to identify the sender, and one for the receiver.
3. *Preconditions* and *effects* for each message, so that the participants derive the same meaning from a message and, consequently, have their locally maintained protocol states in sync. As a result, preconditions and effects provide unambiguous meanings to messages. Preconditions on messages enforce a (partial) ordering of the messages, allowing participants to detect a violation of the protocol. A message can be sent only if its precondition holds. If a message is received in a state at which the preconditions for that message do not hold, the sender of that message is in violation of the protocol. Effects of messages are actions that the participants should take to keep their states consistent. Message effects are realized through *actions* as described below. Message preconditions are evaluated in conjunction with the participant's policy. Therefore, an important aspect of the participants' freedom of enactment is that the policy can block a message precondition by returning a `false` value and potentially violating a protocol. This topic is discussed in Section 3.2.
4. *Commitments* and domain specific *propositions* that are needed by the participants to keep track of events in the protocol. Commitments are needed to identify and enforce obligations between participants.
5. *Actions* that the participants should perform as part of the effects of a message. Note that some of these actions are *internal* to the participant. That is, the participant takes these actions locally. Although messages have effects, participants are allowed to check with their policies before taking these actions. The actions are as follows.
  - Adding or removing a proposition from the state to keep track of the protocol execution. Messages between participants change the protocol state by adding or removing propositions from the working memory.
  - Invoking a procedure. Effect of a message might be invoking a local procedure.



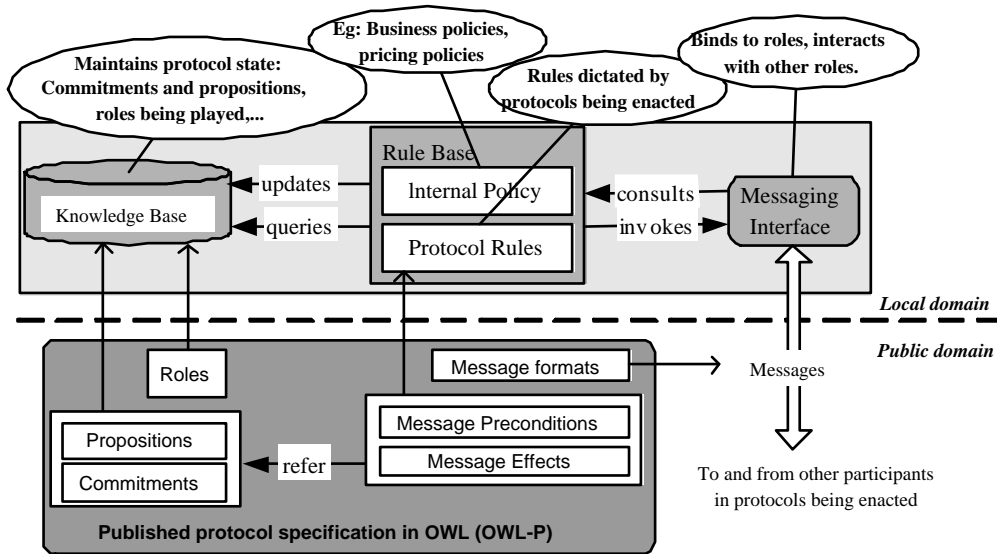


Figure 4: Policy-Based Protocol Enactment Framework

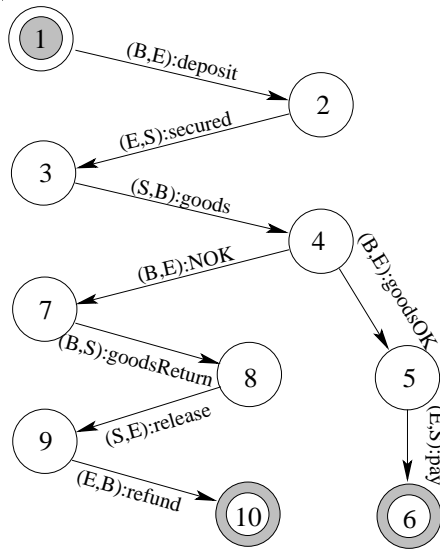


Figure 5: The Escrow Protocol

Escrow is aware of the amount of payment. Figure 6 is the corresponding OWL-P specification in convenient notations after abstracting away the details about message formats.

Let us formally define a Protocol in OWL-P. A Protocol  $\mathbb{P}$  defines a set of messages  $\mathbb{M}$  and a set of participant roles  $\mathbb{R}$ . Hence,  $\mathbb{P} = (\mathbb{M}, \mathbb{R})$ . A message  $M \in \mathbb{M}$  is defined as  $M = (\rho_s, \rho_r, C_{pre}, Eff)$  where,  $\rho_s \in \mathbb{R}$  is the sender,  $\rho_r \in \mathbb{R}$  is the receiver,  $C_{pre}$  is the precondition of the message and  $Eff$  is the set of all actions being effects of the message. A skeleton of a role  $\rho$  then, is  $\mathbb{P}_\rho$  where  $\mathbb{P}_\rho = (\mathbb{M}_\rho, \mathbb{R}_\rho)$ .  $\mathbb{M}_\rho$  is the set of all messages in which  $\rho$  is involved and  $\mathbb{R}_\rho$  is the set of all the roles with whom  $\rho$  interacts including  $\rho$  itself. Figure 7 describes the algorithm. Desai and Singh developed a similar algorithm in [2004 (To Appear)] which was based on state-based representations to specify protocols. However, it had no semantics attached with the actions and states. Note that OWL-P describes the protocol from a global perspective where the propositions are added to a global

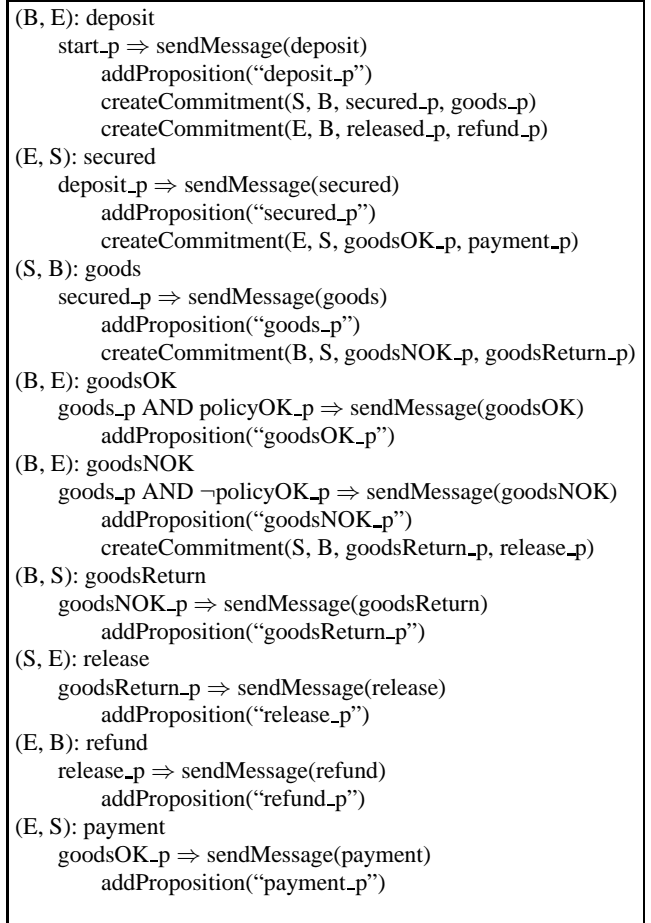


Figure 6: OWL-P for the Escrow Protocol

state and there are no distributed sites. The role skeletons describe the protocol from the perspective of the corresponding participant. When the Partition algorithm is run through the OWL-P for the

```

1  Partition( $\rho$ )
2   $M_\rho \leftarrow \phi$ 
3  For all  $M \in \mathbb{M}$ 
4    If  $\rho = \rho_s$  Then
5       $M_\rho \leftarrow M_\rho \cup M$ 
6    Else If  $\rho = \rho_r$  Then
7      ModifyPrecondition( $C_{pre}$ )
8      If  $sendMessage \in Eff$  Then
9        Replace  $sendMessage$  with  $ReceiveMessage$ 
10      $M_\rho \leftarrow M_\rho \cup M$ 
11 ModifyPrecondition( $cond$ )
12 If  $cond$  is an atomic proposition Then
13   FindReplacement( $cond$ )
14   return
15 Else
16   For all components  $cond\_comp$  of  $cond$ 
17     ModifyPrecondition( $cond\_comp$ )
18 FindReplacement( $cond$ )
19 For all  $M(\rho_s, \rho_r, C_{pre}, Eff) \in \mathbb{M}$  such that
20    $addProposition(cond) \in Eff$ 
21   If  $\rho = \rho_s \vee \rho = \rho_r$  Then
22     return
23   Else
24      $cond \leftarrow C_{pre}$ 
25     ModifyPrecondition( $cond$ )

```

**Figure 7: Partition Algorithm: Generating a Role Skeleton from OWL-P**

Buyer role in the Escrow protocol, it generates the skeleton for the Buyer role as shown in Figure 8. Notice that for the goods message the precondition  $secured\_p$  is replaced by  $deposit\_p$  and for the refund message the precondition  $release\_p$  is replaced by  $goodsReturn\_p$ . This is due to the fact that when the Buyer receives these messages, it will not know about  $secured\_p$  and  $release\_p$ . This is because the Buyer is not involved in the interactions that cause those propositions to hold, namely,  $secured$  and  $release$ . Replaced propositions are found by the Partition algorithm. Procedures **ModifyPrecondition** and **FindReplacement** take a reference to a condition expression and replace its constituent propositions. Also, the  $sendMessage$  actions for the messages that the Buyer receives are replaced by  $receiveMessage$  actions.

*Prototype Implementation.* We showed how to generate skeletons for roles involved in one protocol. But more often than not, agents play roles in more than one protocol in a business process. To enact the local flow of such an agent, its role skeleton for each protocol it participates in should be generated from the protocol’s OWL-P specification. Next, the skeletons need to be spliced in together to generate a composite skeleton. Finally, to enact the local flow, the composite skeleton needs to be augmented with the agent’s local policies. This paper does not address the methodology for splicing in multiple role skeletons. Figure 9 shows the constituent protocols and the participants involved in a procurement business process from the global view. The entities shown are the participants and the edges are the protocols through which they interact. As a simple guideline, each participant will need as many role skeletons as the number of edges attached to it.

We enact the role skeletons generated by the Partition algorithm

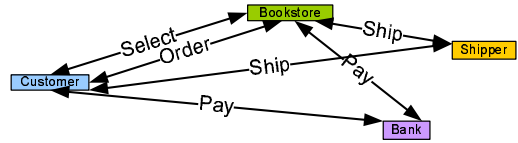
```

(B, E): deposit
  start_p  $\Rightarrow$  sendMessage(deposit)
  addProposition("deposit_p")
  createCommitment(S, B, secured_p, goods_p)
  createCommitment(E, B, released_p, refund_p)
(S, B): goods
  deposit_p  $\Rightarrow$  receiveMessage(goods)
  addProposition("goods_p")
  createCommitment(B, S, goodsNOK_p, goodsReturn_p)
(B, E): goodsOK
  goods_p AND policyOK_p  $\Rightarrow$  sendMessage(goodsOK)
  addProposition("goodsOK_p")
(B, E): goodsNOK
  goods_p AND  $\neg$ policyOK_p  $\Rightarrow$  sendMessage(goodsNOK)
  addProposition("goodsNOK_p")
  createCommitment(S, B, goodsReturn_p, release_p)
(B, S): goodsReturn
  goodsNOK_p  $\Rightarrow$  sendMessage(goodsReturn)
  addProposition("goodsReturn_p")
(E, B): refund
  goodsReturn_p  $\Rightarrow$  receiveMessage(refund)
  addProposition("refund_p")

```

**Figure 8: Skeleton for the Buyer role in Escrow Protocol**

in terms of Jess Rules. Jess rules can be directly mapped from the message rules in the protocol skeletons. The messaging system is implemented in Java. Jess rules invoke Java methods to send messages. On the receiving side the messaging system receives messages and dispatches events to the Jess base. Our messaging system is implemented in JADE (Java Agent Development Framework).



**Figure 9: A Procurement Business Process Global View**

### 3.5 Sketch of Correctness

Here, we informally argue that the skeletons generated by our algorithm are sound and complete with respect to the OWL-P specification. That is, the generated skeletons when executed together, realize the same computations as the input OWL-P and realize no other computations. We define a computation  $C$  as a sequence of message exchanges observed in a run of the protocol. So,  $C = M_0 \cdot M_1 \cdot M_2 \cdots M_n$ . Note that we do not consider the local policies as part of an OWL-P computation as they are not specified in OWL-P.

#### 3.5.1 Soundness

We show that if a computation  $C$  is allowed by the generated skeletons then OWL-P also allows it. Let  $C$  be allowed by the skeletons. Let  $M_i \cdot M_j$  be a step in  $C$ . Let  $\rho_1$  be the sender of  $M_i$  and  $\rho_2$  be the receiver. From the view of  $\rho_2$ , the next observable event will be  $sendMessage$  action in effects of  $M_i$ . Let that message be  $M_j$  and hence  $\rho_2$  be the sender of  $M_j$ .

Now, because  $M_i \cdot M_j$  is allowed by the skeletons, the following properties hold:

1. In the local state of  $\rho_1$ , precondition for sending  $M_i$  holds.
2. In the local state of  $\rho_2$ , precondition for receiving  $M_i$  holds.
3. After receiving  $M_i$ , the local state of  $\rho_2$  enables the precondition of sending  $M_j$ .

If we can show that these conditions also hold for OWL-P, it means that  $M_i \cdot M_j$  is allowed by OWL-P.

By line 4 of the Partition algorithm, in  $\mathbb{P}_{\rho_s}$ , the precondition of  $M_i$  is added unchanged from  $\mathbb{M}$  because  $\rho_s$  is the sender of  $M_i$ . So Condition 1 above holds for OWL-P.

All the propositions that hold in the local state of  $\rho_2$  also hold for OWL-P as  $\rho_2$  was involved in the latest interaction and OWL-P is the global view of the protocol and there are no distributed states. Hence, if the precondition for receiving  $M_i$  holds for  $\rho_2$ , it must hold for OWL-P. So, Condition 2 above holds for OWL-P.

Notice that while generating a skeleton, for the receiving role of a message, no effects are modified except for replacing `sendMessage` actions by `receiveMessage` actions. Hence, the local state of  $\rho_2$  after receiving the message is identical to the OWL-P view of the state. Because the precondition for sending  $M_j$  is enabled for  $\rho_2$ , it will also be enabled for OWL-P; hence Condition 3 above holds.

### 3.5.2 Completeness

We show that if a computation  $C$  is allowed by OWL-P then the generated skeletons also allow it. Let  $C$  be allowed by OWL-P. Let  $M_i \cdot M_j$  be a step in  $C$ . Let  $\rho_1$  be the sender of  $M_i$  and  $\rho_2$  be the receiver. Obviously,  $\rho_2$  is the sender of  $M_j$ .

Here, the three conditions listed in Section 3.5.1, already hold for OWL-P. If we can show that they also hold for the generated skeletons then  $M_i \cdot M_j$  will also be allowed by the skeletons. Condition 1 holds as discussed for Section 3.5.1.

Due to the procedure `FindReplacement` in the Partition algorithm,  $C_{pre}$  of  $M_i$  in  $\mathbb{P}_{\rho_2}$  is modified such that the propositions unknown to  $\rho_2$  are replaced by the propositions as last known to  $\rho_2$  that lead to  $M_i$ . Hence, if the  $C_{pre}$  holds for OWL-P, it will also hold for  $\mathbb{P}_{\rho_2}$ . Hence Condition 2 holds. Condition 3 holds as discussed for Section 3.5.1.

## 3.6 Revisiting the Motivating Problems

In this section we show how and to what extent does our approach aid in solving the problems identified in Section 2.

*Semantics for Protocols.* Our protocol specification is minimal in that it does not dictate all possible computations or runs. The protocol specifies the meanings of messages in terms of when they can be sent and what their effects are. Message sequencing is achieved as a consequence of planning on the part of the agents to reach a final state of the protocol. Participants can therefore use their policies to enact local flows that best serve their interests. Hence, when partnerships are formed dynamically, the role skeletons of the partners can be verified against each other. Because the protocols are standardized and published, skeletons generated from them are guaranteed to be compatible. However, if the participants have different versions of the protocol, or they customized the generated skeletons, their semantics must be reasoned about to verify the compatibility. We defer the discussion of compatibility verification to future work.

*Reusable Protocols.* The clear separation between protocols and policies allows for the specification of protocols independent of local policies. Hence, the OWL-P protocol specifications are modular and fit well into software engineering abstractions such

as refinement and aggregation. As these specifications are published, designers can reuse them to build local flows of new agents by generating corresponding role skeletons and augmenting them with their local policies.

*Runtime Adaptability.* The use of commitments to represent important events in the protocol allows a protocol to generate a variety of runs. Protocols can be refined into other protocols that constrain the runs more, or can be adapted into other protocols during runtime [Mallya and Singh, 2004]. Hence, if a change of business policy or interaction pattern occurs, modifying the rule base at runtime and reverifying the compatibility with the partners would allow the local flows to adapt at runtime.

*Exception Handling.* Commitments allow us to identify which agent is responsible for which event. Commitments bring accountability and enable agents to intelligently reason about exceptions. For example, an agent can decide to release another from a commitment if the need arises or if the debtor agent asks for an extension of the deadline of the commitment. Pending base-level commitments correspond to an absolute obligation of one agent to the other. Any failure at a point where base-level commitments are pending may result in undesirable results. Hence, protecting the scopes for the life-time of base-level commitments is a guideline to the designer.

## 4. DISCUSSION

Developing business processes for open systems poses significant challenges. Interactions in business processes can be quite complex making interoperation of autonomous partners a major concern in open systems. Most research efforts focus on alleviating the interoperation problems by documenting an expected choreography of message exchanges. The Web Services Choreography Interface (WSCI) [WSCI, 2002] specification describe the choreography of message exchanges, using control flow constructs. As such, they can support complex interactions. However, a WSCI specification gives no semantics to interactions, which makes it less amenable to reuse. Business Process Execution Language (BPEL) [BPEL, 2003] is a flow language designed to specify composition of Web services. However, a BPEL specification intertwines business logic and interactions, and therefore cannot be reused and lack the semantics of the activities. Also, BPEL specifications are not published; it represents the internal orchestration of a partner's local flow.

The Semantic Web Services Initiative has produced OWL-S [DAML-S, 2002]. OWL-S is in essence similar to WSCI, however service interfaces are given semantics in the model. In addition, OWL-S processes can be dynamically composed via planning. But the specifications are logically centralized and support the perspective of only one participant thereby limiting autonomy of others.

The RosettaNet [RosettaNet, 1998] effort centers around publishing protocols and designing the business processes around them. RosettaNet represents a step in the right direction. However, they are currently limited to two-party request-response interactions called Partner Interface Processes (PIPs). PIPs lack a semantics.

Whereas the local flow of a partner cannot be reused by another, the protocols can be. At the same time, given a formal semantics, protocols can be refined or aggregated, thus yielding new protocols. Mallya and Singh [Mallya and Singh, 2004] treat these concepts formally. The Business Process Handbook [Malone et al., 2003], in a similar vein, catalogues different kinds of business processes in a hierarchy. For example, *sell* is a generic business process. It can be qualified by *sell what*, *sell to who*, and so on. Our notion of

a protocol hierarchy bears similarity with the Handbook. Our effort is focused on providing formal semantics to the hierarchy.

Older technologies such as workflow systems lacked the flexibility and agility that current business processes need. Businesses today are moving their processes online, with the help of the Semantic web [Petrie and Sahai, 2004]. It is this area of confluence of business processes and the Semantic Web that we deal with. Our work draws from traditional concepts in distributed computing and multiagent systems, and some of our results apply to agent interaction protocols.

The representation of business contracts is an interesting area of research. Grosf and Poon [2003] represent agent contracts in OWL and RuleML. They develop an ontology for processes and contracts. Davulcu *et al.* [2004] develop a logic for specifying contracts in Web services. Commitment-based protocols serve precisely the purpose of specifying contractual arrangements. Commitments are, in principle, enforceable and partners can be held responsible for violation of commitments. It is reasonable to expect that not all parts of a complicated contract will be reflected in the protocol, because contracts are sensitive to the context in which the interactions take place. For example, the Uniform Commercial Code (UCC) [UCC] has certain stipulations for merchants and consumers. Such codes are more like policies than protocols. Some partners may be required to follow the UCC, perhaps based on their geographical location. How to layer and prioritize such policies is a direction of future research.

**Future Directions.** One important direction is the layering of contextual policies such as UCC, as described above. Contextual policies play an important role in a partner's decision-making and would presumably be given a higher priority. That is to say a local policy should behave differently under different contexts.

Another vital direction to explore is the OWL-P representation of composite protocols. To enact the local flow of a merchant in Figure 1, we need to splice in the skeletons for payment protocol, the shipping protocol and the procurement protocol in such a way that they realize the local flow as seen in Figure 1. Also, there might be dependencies between the protocol skeletons that are spliced in. How should we specify the data flow dependencies, control dependencies, failure-recovery dependencies and the relationship of one protocol to the other in OWL-P? Also, when a pair of agents dynamically form partnerships but have different versions of protocols, how can we verify their compatibility?

## References

- BPML. Business process execution language for web services, version 1.1, May 2003. [www-106.ibm.com/developerworks/webservices/library/ws-bpel](http://www-106.ibm.com/developerworks/webservices/library/ws-bpel).
- Cristiano 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.
- 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, Srinu Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara.
- H. Davulcu, M. Kifer, and I.V. Ramakrishnan. Ctr-s: A logic for

- specifying contracts in semantic web services. In *Proceedings of the 13th International World Wide Web Conference*, May 2004.
- Nirmit Desai and Munindar P. Singh. Protocol-based business process modeling and enactment. In *International Conference on Web Services*, 2004 (To Appear). <http://www4.ncsu.edu/~nvdesai/icws04.pdf>.
- Escrow.com. Online escrow process, 2003. <http://www.escrow.com/solutions/escrow/process.asp>.
- Benjamin N. Grosf and Terrence C. Poon. SweetDeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proceedings 12th International Conference on the World Wide Web*, 2003.
- Jess. <http://herzberg.ca.sandia.gov/jess/>.
- Ashok U. Mallya and Munindar P. Singh. A semantic approach for designing commitment protocols. In *Proceedings of the AAMAS-04 Workshop on Agent Communication*, July 2004. To appear.
- Thomas W. Malone, Kevin Crowston, and George A. Herman, editors. *Organizing Business Knowledge: The MIT Process Handbook*. MIT Press, Cambridge, MA, 2003.
- OWL. Web ontology language, Feb 2004. <http://www.w3.org/TR/2004/REC-owl-features-20040210/>.
- Charles Petrie and Akhil Sahai. Business processes on the web. *IEEE Internet Computing*, 8(1):28–29, January 2004.
- RosettaNet. Home page, 1998. [www.rosettanet.org](http://www.rosettanet.org).
- SET. Secure electronic transactions (SET) specifications, 2003. [http://www.setco.org/set\\_specifications.html](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.
- UCC. <http://www.law.cornell.edu/ucc/ucc.table.html>.
- Mario Verdicchio and Marco Colombetti. Commitments for agent-based supply chain management. *ACM SIGecom Exchanges*, 3(1):13–23, 2002.
- WSCI. Web service choreography interface 1.0, July 2002. [www.sun.com/software/xml/developers/wsci/wsci-spec-10.pdf](http://www.sun.com/software/xml/developers/wsci/wsci-spec-10.pdf).
- WSDL. Web Services Description Language, 2002. <http://www.w3.org/TR/wsdl>.
- Pinar Yolum and Munindar P. Singh. Commitment machines. In *Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL-01)*, pages 235–247. Springer-Verlag, 2002a.
- Pinar 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 2002b.