OWL-P: A Methodology for Business Process Development

Nirmit Desai[1], Ashok U. Mallya[2], Amit K. Chopra[1], and Munindar P. Singh[1]

[1] Department of Computer Science
North Carolina State University, Raleigh, NC 27695-8206, USA
{nvdesai, aumallya, akchopra, singh}@ncsu.edu
[2] Veraz Networks Inc
926 Rock Avenue, Suite 20, San Jose, CA 95131, USA
amallya@veraznet.com

Abstract. Business process modelling and enactment are notoriously complex, especially in open settings where the business partners are autonomous, requirements must be continually finessed, and exceptions frequently arise because of real-world or organizational problems. Traditional approaches, which attempt to capture processes as monolithic flows, have proved inadequate in addressing these challenges. We propose an agent-based approach for business process modelling and enactment which is centred around the concepts of commitment-based agent interaction protocols and policies. A (business) protocol is a modular, public specification of an interaction among different roles. Such protocols, when integrated with the internal business policies of the participants, yield concrete business processes. We show how this reusable, refinable and evolvable abstraction simplifies business process design and development.

1 Introduction

Unlike traditional business processes, processes in open, Web-based settings typically involve complex interactions among autonomous, heterogeneous *business partners*. Conventionally, business processes are modelled as monolithic workflows, specifying exact steps for each participant. Because of the exceptions and opportunities that arise in open environments, business relationships cannot be pre-configured to the full detail. Thus, flow-based models are difficult to develop and maintain in the face of evolving requirements. Furthermore, such conventional models do not facilitate flexible actions by the participants.

This paper proposes an approach for business process modelling and enactment, which is based on a combination of protocols and policies. The key idea is to capture meaningful interactions as *protocols*. Protocols can involve multiple roles and address specific purposes such as ordering, payment, shipping and so on. Protocols are given a contractual semantics in terms of commitments among roles that capture the essence of the relationship among roles. In order to maximize participants' autonomy and to be reusable, protocols emphasize the essence of the interactions and omit local details. Such details are supplied by each participant's *policies*. For example, when a protocol allows a participant to choose from multiple actions, the participant's policy decides

M. Kolp et al. (Eds.): AOIS 2005, LNAI 3529, pp. 79–94, 2006.

which one to perform. Typically, policies are business logic that provide and process message contents.

This paper seeks to develop the main techniques needed to make this promising approach practical. Our contributions include a language and an ontology for protocols called OWL-P, which is coded in the Web Ontology Language (OWL) [1]. OWL-P describes concepts such as roles, the messages exchanged between the roles, and declarative protocol rules. OWL-P compiles into Jess rules which then can be integrated with the local policies in a principled manner.

Protocols are not only reusable across business processes but also amenable to abstractions such as refinement and aggregation [2]. The key benefits of this approach are (1) a separation of concerns between protocols and policies in contrast to traditional monolithic approaches; and (2) reusability of protocol specifications based on design abstractions such as specialization and aggregation.

1.1 Running Example

As a running example, let's consider a business process involving a small number of parties. Fig. 1 depicts a purchase process where items to be purchased have already been selected and the price has been agreed upon. Each participant is shown by a separate shaded region, the graph made of dark edges denotes the *flow* of the given participant. Circular nodes represent the participant's internal business logic or policies, e.g. to decide the parameters of an out-bound message. Rectangular nodes represent external interfaces through which a participant receives messages. Thus, an ordering of dark arrows, circles and rectangles represents the local process of the participant. When there

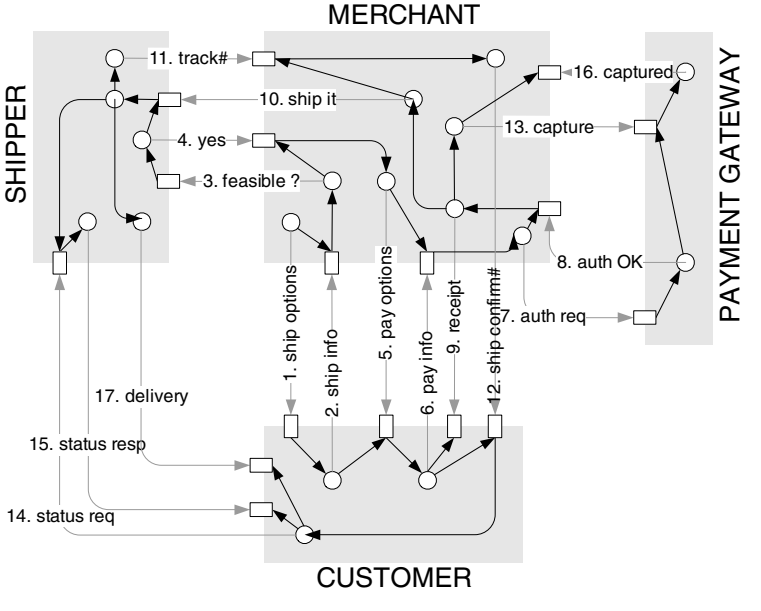

Fig. 1. A purchasing process

are multiple out-edges from a node, all of them are taken concurrently. The messages are labelled with numbers to indicate a possible order in which they might occur.

1.2 Shortcomings of Traditional Approaches

The process of Fig. 1 can be captured via a traditional flow-based modelling approach. Such a representation would be functionally correct, but inadequate from the perspectives of open environments. The following are its shortcomings:

Lack of Contractual Semantics. Traditional approaches expose low-level interfaces, e.g. via WSDL [3], but associate no contractual semantics with the participants' actions. To control the autonomy of the participants and enforcing compliance, such a semantics is crucial. This lack precludes flexible enactment (as needed to handle exceptions) as well as reliable compliance checking. For this reason, we cannot determine if a deviation from a specific sequence of steps is significant.

Lack of Reusable Components. The local processes of the partners are not reusable even though the patterns of interaction among the participants might be. Local processes are monolithic in nature and formed by *ad hoc* intertwining of internal business logic and external interactions. Since business logic is proprietary, local processes of one partner are not usable by another. For instance, if a new customer were to participate in this open environment, its local process would have to be developed from scratch.

Organization

Section 2 introduces some key concepts and terminology. Section 3 describes our protocol specification language and its semantics. Section 4 discusses composite protocols and their construction. Section 5 shows how augmenting policies with protocols can be used to develop processes. Section 6 compares our work with relevant research efforts in the area and Section 7 concludes the paper.

2 Concepts and Terminology

Fig. 2 shows our conceptual model for treatment of business processes based on protocols and policies. Boxed rectangles are abstract entities (interfaces), which must be combined with business policies to yield concrete 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 using rules termed *protocol logic* that specify the interactions of the participating *roles*. Roles are abstract and are adopted by agents to enable concrete computations. Whereas the protocol logic specifies the protocol from the global perspective, a *role skeleton* specifies the protocol from the perspective of the corresponding participant role. Thus, each role skeleton defines the behaviour of the respective role in a protocol.

When an agent needs to participate in multiple protocols, a *composite skeleton* can be constructed by combining the protocols according to some composition constraints and deriving the role skeleton. For example, in a supply chain process, a supplier would be

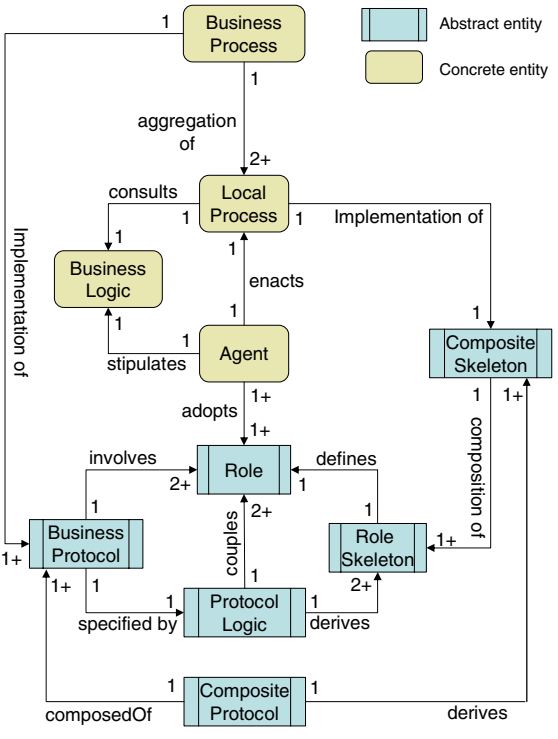

Fig. 2. Conceptual model

a merchant when interacting with a retailer in a trading protocol and would be an item-sender in a shipping protocol for sending goods to the retailer. A composite skeleton for such a supplier could be composed by combining trading and shipping protocols and then deriving the role skeleton for item-sender/merchant role. The resultant composite skeleton could also be published and then reused for developing local processes of other suppliers.

An agent's private policies or *business logic* are described via rules. The *local process* of an agent is an executable realization of a composite skeleton obtained by integration of the protocol logic of the composite skeleton and the business logic of the agent. A *business process* is the aggregation of the local processes of all the agents participating in it. Conversely, a business process is an implementation of the constituent business protocols.

2.1 Protocols and Commitments

Commitments are used to give semantics to agent interaction. As agents interact, they create and manipulate commitments. A commitment $\mathsf{C}(x, y, p)$ denotes that agent x is obligated to agent y for bringing about condition p. Commitments can be *conditional*, denoted by $\mathsf{CC}(x, y, p, q)$, meaning that x is committed to y to bring about q if p holds

where p is called the precondition of the commitment. For example, the conditional commitment $\mathsf{CC}(c, b, goods(g), pay(p))$ 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. Commitments are created, satisfied and transformed in certain ways [4]. The following are the operations defined on commitments:

Op1. CREATE(x, c) establishes the commitment c in the system.
Op2. CANCEL(x, c) cancels the commitment c.
Op3. RELEASE(y, c) releases c's debtor from commitment c without c being fulfilled.
Op4. ASSIGN(y, z, c) replaces y with z as c's creditor.
Op5. DELEGATE(x, z, c) replaces x with z as the c's debtor.
Op6. DISCHARGE(x, c) c's debtor x fulfils the commitment.

A commitment is said to be *active* if it has been created but not yet discharged. The rules regarding discharge of a commitment are given below.

Dis1. $\frac{C(x,y,p) \wedge p}{discharge(x, C(x,y,p))}$
Dis2. $\frac{CC(x,y,p,q) \wedge p}{create(x, C(x,y,q)) \wedge discharge(x, CC(x,y,p,q))}$
Dis3. $\frac{CC(x,y,p,q) \wedge q}{discharge(x, CC(x,y,p,q))}$

3 Protocol Specification

A business protocol is a specification of the allowed interactions between two or more participant roles. The specification focusses on the interactions and their semantics. What does it mean to send a certain message to a business partner? What is expected of the participants wishing to comply to a business protocol? How are the protocols specified? These are the questions we address in this section.

3.1 OWL-P: OWL for Protocols

OWL-P is an ontology based on OWL for specifying protocols; it functions as a schema or language for protocols. The main computational aspects of protocols are specified using rules. We employ the Semantic Web Rule Language (SWRL) [5] for defining rules. SWRL allows us to specify implication rules over entities defined as OWL-P instances. The availability of tools such as Protégé [6] is a motivation for grounding our approach in OWL.

The important OWL-P elements and their properties are shown in Fig. 3. An entity with a little rectangle represents the domain of the corresponding property. Many of the properties are self-explanatory and reflect the conceptual model introduced in Section 2.

Slots are analogous to data variables. A slot is said to be *defined* when it is assigned a value and it said to be *used* when its value is assigned to another slot. A slot in a protocol may be assigned a value produced by another protocol and hence be represented as an *External Slot*. An external slot is untyped until it is given the type of the external value to which it is bound. By contrast, a *Native Slot* is typed and defined inside the protocol.

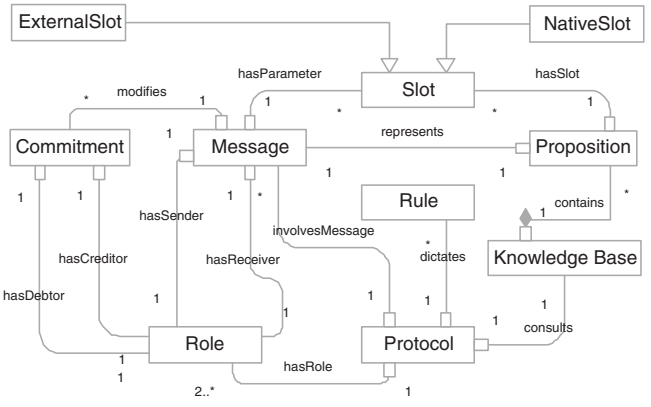

Fig. 3. Basic OWL-P ontology

A *Protocol* dictates several rules and consults a *Knowledge Base*. A knowledge base consists of a set of *Propositions*. A proposition in a knowledge base may correspond to a message, active commitments or other domain specific propositions.

Fig. 4 shows a protocol for ordering goods (along with others, to which we refer later). For readability, a leading and trailing * is placed around external slot names, as in *amount* and *itemID*. The customer requests a quote for an item, to which the merchant responds by providing a quote. Here, a commitment is created providing semantics for the message. The commitment means that the merchant guarantees receipt of the item if the customer pays the quoted price. The customer can either accept the quote or reject it (not shown). Again, the semantics of acceptance is given by the creation of another commitment from the customer to pay the quoted price if it receives the requested item. Below are the rules for the Order protocol in the "antecedents ⇒ consequents" notation.

Ord1. contains(KB, startProp) ⇒ send(B, reqForQuote(?itemID))
Ord2. contains(KB, reqForQuoteProp(?itemID)) ⇒ send(S, quote(?itemID, ?itemPrice)) ∧ createCommitment(S, CC(S, B, pay(?itemPrice), goods(?itemID)))
Ord3. contains(KB, quoteProp(?itemID, ?itemPrice))⇒send(B, acceptQuote(?itemID, ?itemPrice)) ∧ createCommitment(C, CC(C, M, goods(?itemID), pay(?itemPrice)))
Ord4. contains(KB, quoteProp(?itemID, ?itemPrice))⇒send(B, acceptQuote(?itemID, ?itemPrice))

In the above rules, reqForQuote, quote, and acceptQuote are OWL-P message instances (individuals in OWL terminology). Corresponding proposition instances are reqForQuoteProp, quoteProp, and acceptQuoteProp. Propositions pay and goods are commitment conditions, while itemID and itemPrice are native slots. Readers may notice that the itemID variable in the first rule is not assigned any value by the antecedents. It means that the rule is abstract and not executable and, as we will see in Section 5.2, it can be augmented with business logic that produces such values. Rules having unde-

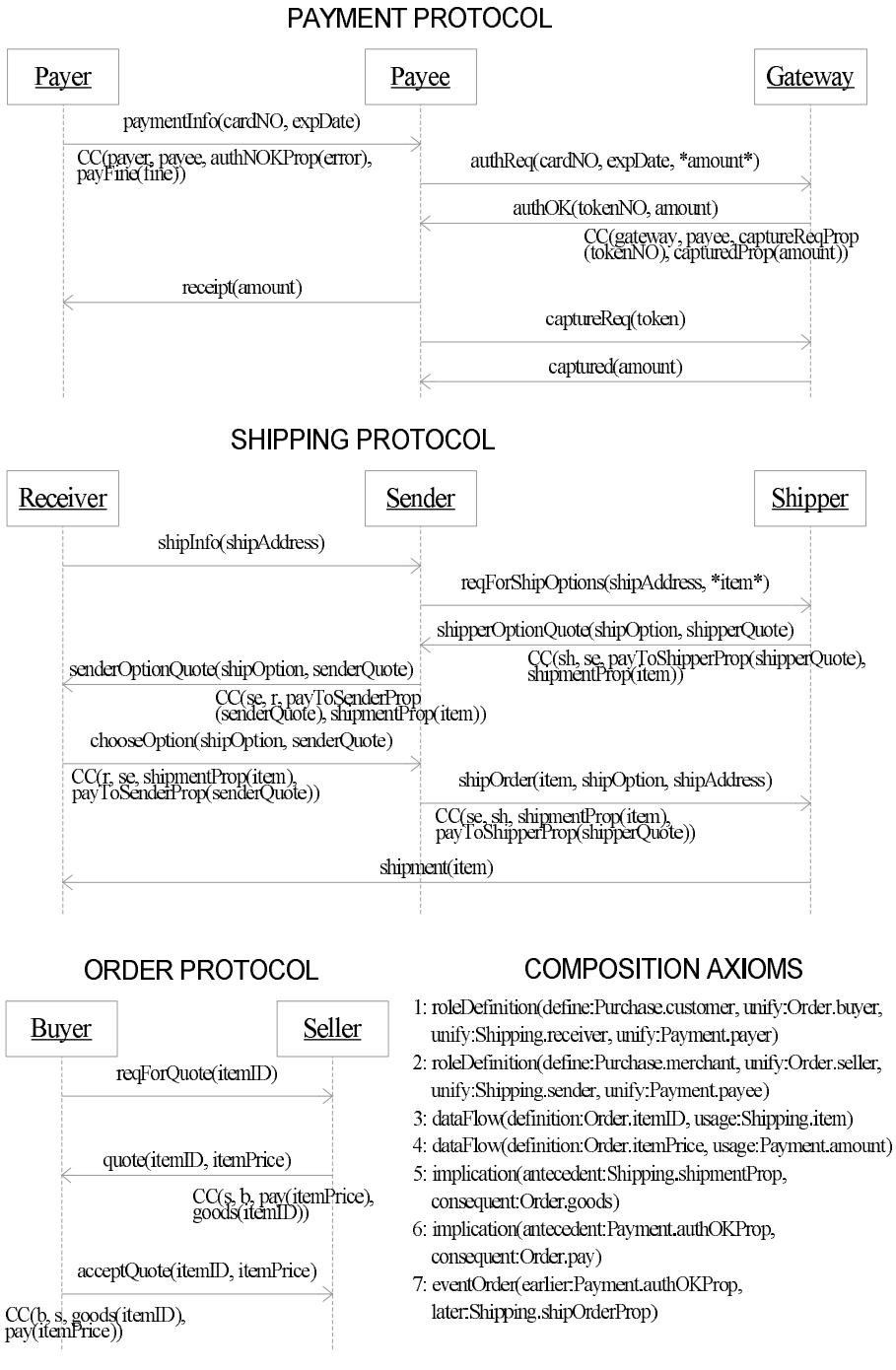

Fig. 4. Example: Order, Shipping, and Payment protocols and their composition

fined native slots must be augmented with the business logic that produces such values. How do these rules define the protocol? The next section describes the operational semantics of the protocol rules. The OWL-P ontology and protocol instance examples in their RDF/XML serialization, and corresponding Protégé projects are available on the Web [7].

3.2 Operational Semantics

Protocols are specified from the global perspective with an assumption of an abstract global knowledge base and the rules are assumed to be forward-chained. OWL-P defines several property predicates with operational semantics. Table 1 lists the seman-

Table 1. Operational semantics of protocol rules

Predicate	Domain	Range	Meaning
contains	KB	Proposition	Proposition $\in$ KB ?
assert	Proposition	KB	KB $\leftarrow$ KB $\cup$ Proposition
send	Role	Message	Asynchronous send to the receiver assert(KB, MessageProp)
receive	Role	Message	Asynchronous receive from the sender assert(KB, MessageProp)
createComm	Role	Commitment	assert(KB, CommitmentProp)

tics for such property predicates of OWL-P. A proposition cannot be retracted from a knowledge base. In the forthcoming examples, we may omit the OWL-P properties, e.g. contains, send, createCommitment when the meaning is clear. Fig. 5 shows an inside view of an agent to demonstrate how the rules govern the interactions. For now, ignore steps 3, 4, and 5 dealing with policy rules. When a message is received, it is checked against the protocol rules to see if it may be consumed. If so, a corresponding proposition is asserted and any activated rules are executed. Doing so may activate other rules, resulting in further propositions being asserted and messages being sent.

4 Composite Protocols

The previous section described how to specify individual protocols. To meet the requirements of business processes, it is necessary to compose them from simpler protocols. Now we show how protocols can be composed.

Conceptually, each component protocol achieves a business goal. Thus, several such protocols composed together would achieve the goals of the larger business process. Composition also enables refinements of protocols with additional rules. The ability to compose protocols would allow significant reuse of published protocols. How can we construct such composite protocols? How do they facilitate reusability? How do they allow refinements of protocols? This section answers these questions.

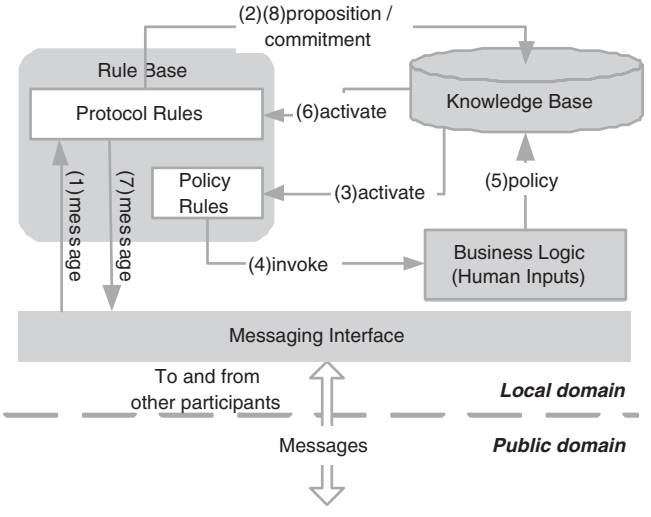

Fig. 5. Agent architecture: protocol and policy interplay

4.1 Construction of Composite Protocols

Fig. 6 describes the OWL-P classes and properties that deal with the problem of protocol composition. A *Composite Protocol* is an aggregation of component protocols and is defined by a *Composition Profile*. A composition profile describes the combination of two or more protocols by stipulating several *Composition Axioms*. Composition axioms

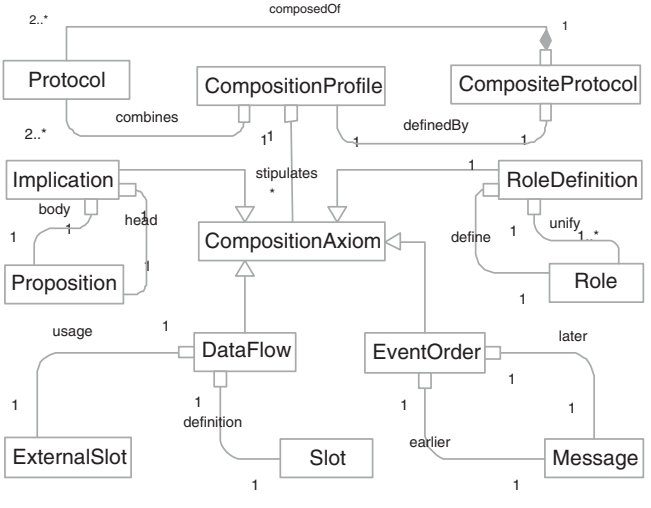

Fig. 6. OWL-P composition classes and properties

define relationships among the protocols being combined. The operational semantics of an axiom specifies the way in which the relationships affect the composite protocol. Fig. 4 depicts an Order protocol, a Shipping protocol, a Payment protocol and a set of composition axiom instances stating the relationships among them.

A *Role Definition* axiom states which of the roles in the component protocols are to be adopted by the same agent and defines the name of the unified (coalesced) role in the composite protocol. In the example, the first axiom states that the roles of a customer in Order, a payer in Payment and a receiver in Shipping protocol are played by an agent who will play the role of a customer in the Purchase protocol.

A *Data Flow* axiom states a data-flow dependency among the protocols. A component protocol might be using a slot defined by another component protocol, possibly with a different name. Since a slot can be defined only once, and native slots must be defined inside the protocol, they cannot use a value defined by another protocol. Hence, the range of the *usage* property must be an external slot. In the example, the fourth axiom states that the slot amount in the Payment protocol gets its value from the slot itemPrice in the Order protocol. Such a dependency exerts an ordering among the rule defining the slot and all the rules using it: none of the the rules using the slot can fire before the slot is assigned a value by the defining rule.

An *Implication* axiom states that an assertion of proposition X in a component protocol implies an assertion of proposition Y in another component protocol. For example, the sixth axiom states that an assertion of authOKProp in the Payment protocol means an assertion of pay in the Order protocol. This can be easily achieved by adding an implication rule to the composite rulebase.

Unlike the DataFlow axiom, an *EventOrder* axiom explicitly specifies an ordering among the messages of the component protocols. For example, the seventh axiom states that an authOK message from the payment gateway must be received before a shipOrder message is sent to the shipper. This can be achieved by making the rule for the later event depend on the rule for the earlier event.

Operational semantics of these axioms are given in [8]. Composition axioms have to be specified by a designer. There might be several ways of composing the component protocols yielding different composite protocols. As a special case, if the component protocols are completely independent of each other, no axioms need be specified and their OWL-P specifications can be simply aggregated yielding the OWL-P specification of the composite protocol. If deemed necessary, more types of composition axiom can be defined along with their properties and operational semantics. A composite protocol exposes its compositionProfile and possesses all the properties of the component protocols. Hence, a composite protocol itself can be a component protocol in some other composition profile instance. How can we determine whether additional component protocols are needed? To answer this question, we define *closed* and *open* protocols. A protocol is closed if it has no external slots, and all the commitments created in the protocol can be discharged by the protocol. A protocol is *open* if it is not closed. A designer's goal is to obtain a closed protocol by repeated applications of composition. Observe that, in Fig. 4, the Order protocol is open as its rules do not assert propositions pay and goods necessary for discharging the commitments created. The Payment,

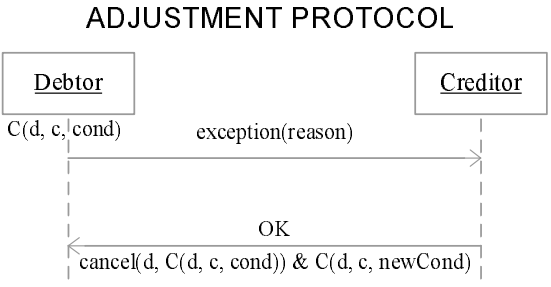

Fig. 7. Handling refinements by composition

Shipping and Purchase protocols are also open according to the definition. A designer would choose protocols that assert these missing propositions and combine them with the Purchase protocol to obtain a closed composite protocol.

4.2 Refinement by Composition

Business protocols evolve continually as new requirements and new features routinely arise. Therefore, the ability to systematically refine protocols is valuable. In the composite Purchase protocol, consider a situation in which the customer has already paid the merchant for the goods and hence the commitment C(S,B,goods(itemID)) is active. However, while trying to order the shipment, if a fire destroys the merchant's warehouse, the merchant will not be able to honor its commitment to ship the item. How can such exceptions be handled? The protocol could detect the violation due to an unfulfilled commitment and the merchant could be held legally responsible. However, A more flexible solution would be to allow the merchant to refund money and cancel its commitment to ship, provided the customer agrees to it. We can achieve this flexibility by combining the purchase protocol with the adjustment protocol shown in Fig. 7 yielding the composite protocol *Flexible* with these composition axioms:

AdjAx1. roleDefinition(define: Flexible.customer, unify: Purchase.customer, unify: Adjustment.creditor)
AdjAx2. roleDefinition(define: Flexible.merchant, unify: Purchase.merchant, unify: Adjustment.debtor)
AdjAx3. implication(body: Purchase.C(B,S,goods(itemID)), head: Cancel.C(D,C, cond))
AdjAx4. implication(body: Cancel.C(D,C,newCond), head: Purchase.C(S,B,refund))

Similar protocols for assigning, delegating, and releasing commitments can be defined. Adding new functionalities would involve composition of a set of rules for the new requirements with the original protocol.

5 Processes

As described in Section 2, a process is an aggregation of the local processes of participating agents. However, an OWL-P specification of a protocol is a model of the interaction from a global perspective. To construct the local process of a participant, we need to derive the participant's view of the protocol, called its *role skeleton*. Section 5.1 describes the generation of role skeletons from an OWL-P specification.

5.1 Role Skeletons

A role skeleton is one role's view of the protocol. Here, we provide the intuition behind generating role skeletons from an OWL-P protocol specification. The complete algorithm is given in [8]. OWL-P describes a protocol from the global perspective where the propositions are added to the global state and there are no distributed sites. As in all distributed systems, the state of a protocol as seen by a role is changed only when a message is sent or received by that role. This observation forms the basis for deriving role skeletons.

As an example, we show a rule in the Shipping protocol in Fig. 4, and the same rule in the generated skeleton of the receiver. As the receiver would not be aware of the previous exchanges between the sender and the shipper, the antecedent of the rule for receiving senderOptionQuote should be adjusted as shown below.

Protocol Rule
shipperOptionQuoteProp(.,.) ⇒ senderOptionQuote(.,.) ∧
CC(Se,Re,payToSenderProp(.),shipmentProp(.))

Receiver Skeleton Rule
shipInfoProp(?shipAddress) ⇒ receive(senderOptionQuote(.,.))
∧ CC(Se,Re,payToSenderProp(.),shipmentProp(.))

5.2 Policies

Generation of a role skeleton is not enough to obtain a local process of a participant. As we mentioned earlier, some of the rules of the protocols may be abstract, meaning that values of some of the native slots in the rule must be produced by the role's business logic. Hence, a role skeleton must be augmented with the business logic to obtain a local process. How can we determine whether an augmented role skeleton is a local process? To answer this question, we first define *concrete* and *abstract* role skeletons, as well as a *local process*. A role skeleton is *concrete* if all of its native slots are defined. A role skeleton is *abstract* if it is not concrete. A *local process* is a role skeleton that is concrete and derived from a closed protocol.

Seller skeleton rules:
startProp ⇒ receive(C, reqForQuote(?itemID))

reqForQuoteProp(?itemID) ∧ **quotePolicy(?itemPrice)** ⇒
quote(?itemID, ?itemPrice) ∧ CC(S, B, pay(?itemPrice), goods(?itemID))

quoteProp(?itemID, ?itemPrice) ⇒ receive(C, acceptQuote(?itemID, ?itemPrice)) ∧
CC(C, M, goods(?itemID), pay(?itemPrice))

Seller policy rule for quote:
reqForQuoteProp(?itemID) ⇒ **call(policyDecider, quotePolicy(?itemID))**

We propose that the business logic be specified in terms of the local policy rules of the agents. The skeleton of the merchant role in the Order protocol augmented with the policy rules of the seller agent is shown above. The last rule is the policy rule that calls a business logic operation to decide how much to quote. The operation would assert the quotePolicy proposition and that would activate the second protocol rule. Observe that this pattern of augmenting policy rules is general and will be applied to the rules where the agent has to make a decision and respond. It would also assign a value to native slots that are not defined.

5.3 Usage

Fig. 8 summarizes our methodology with a scenario involving a customer interested in purchasing goods online. Software designers design protocols and register them with protocol repositories. They may also construct composite protocols and reuse the existing component protocols from the repository. A merchant wishing to sell goods online looks up the repository for a suitable Purchase protocol. It generates the skeleton for the merchant role, augments it with its local policies and deploys the result as a service. The service profile for this service would contain an OWL-P description of the Purchase protocol. The service can be registered with a UDDI registry. If a customer wishes to buy goods online, it searches the UDDI registry, finds the merchant and acquires the OWL-P skeleton for the customer role from the merchant. The customer enacts its local process by augmenting the skeletons with its local policies and starts interacting with the merchant. We have developed tools to support these development scenarios and a prototype implementation based on the agent architecture of Fig. 5 [9]. Note that we propose only a methodology for development and there might be other issues to be resolved for realizing an e-commerce enterprise.

6 Related Work

Several areas of research are relevant to our work. We discuss each of them briefly and highlight the differences.

Composition. BPEL [10] is a language designed to specify the static composition of Web services. However, it mixes the interaction activities with the business logic making it unsuitable for reuse. OWL-S [11], which includes a process model for Web Services, uses semantic annotations to facilitate dynamic composition. A composed service is produced at runtime based on constraints. While dynamic service composition has some advantages, it assumes a perfect markup of the services being composed. Dynamic composition in OWL-S involves ontological matching between inputs and outputs. Such a matching might be difficult to obtain automatically given the heterogeneity of the web. For this reason, we do not emphasize dynamic service composition.

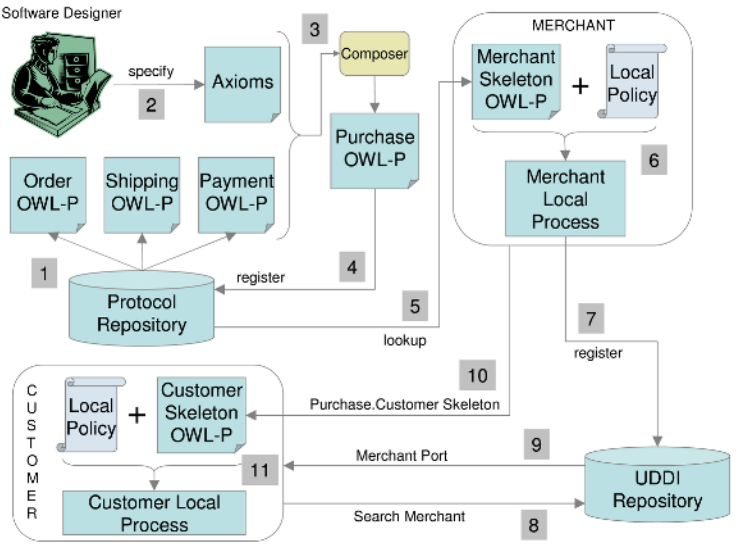

Fig. 8. Usage scenario

Our goal is to provide a human designer with tools to facilitate service composition. Unlike BPEL, which specifies the internal orchestration of services, WSCI [12] specifies the conversational behaviour of a service using control flow constructs. However, these specifications lack a semantics, which makes them difficult to compose and reuse.

Several other approaches aim to solve the service composition problem by emphasizing formal specifications to achieve verifiability. Solanki *et al.* [13] employ interval temporal logic to specify and verify ongoing behaviour of a composed service. Their use of "assumption" and "commitment" (different meaning than here) assertions allows better compositionality. Gerede *et al.* [14] treat services as activity-based finite automata to study the decidability of composability and existence of a look-ahead delegator given a set of existing services. However, these approaches consider neither the autonomy of the partners, nor the flexibility of composition.

Software Engineering. Our methodology advocates and enables reuse of protocols as building blocks of business processes. Protocols can not only be composed, they can also be systematically refined to yield more robust protocols. Mallya and Singh [2] treat these concepts formally. The MIT Process Handbook [15], 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 some similarity to the Handbook. RosettaNet [16] is similar to our approach in that it centres around publishing protocols and designing the business processes around them. However, it is currently limited to two-party request-response interactions called Partner Interface Processes (PIPs) and, more importantly, PIPs lacks a formal semantics.

Agent-oriented software methodologies aim to apply software engineering principles in the agent context e.g. Gaia, KAOS, MaSE, and SADDE [17]. Tropos [18] differs from these in that it includes an early requirements stage in the process. Gaia [19] differs from others in that it describes roles in the software system being developed and identifies processes in which they are involved as well as safety and liveness conditions for the processes. It incorporates protocols under the *interaction model* and can be used with commitment protocols. Baïna *et al.* [20] advocate a model-driven Web service development approach to ensure compliance between a service's implementation and its external protocol specifications. Our work differs from these in that it is aimed at achieving protocol re-usability by separation of protocols and policies and it addresses the problem of protocol compositions.

7 Conclusions

We have presented an approach for designing processes that recognizes the fundamental interactive nature of open environments where the autonomy of the participants must be preserved. Commitments provide the basis for a semantics of the actions of the participants, thereby enabling the detection of violations. The significance of this work derives from the importance of processes in modern business practice. With over 100 limited business protocols having been defined [16], this approach will permit the development and usage of an ever-increasing set of protocols for critical business functions. We demonstrated the practicality of our approach by embedding it in an ontology and language for specifying protocols. Not only is this approach conducive to reuse, refinement and aggregation but it has also been implemented in a prototype tool. It would be ineteresting to see theoretical foundations of this work in the process algebra. It would allow one to establish properties of the protocols and relationships among them.

Acknowledgments

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

References

1. OWL Web Ontology Language: Overview. www.w3.org/TR/owl-features/ (2004) W3C Recommendation.
2. Mallya, A.U., Singh, M.P.: An algebra for commitment protocols. Autonomous Agents and Multiagent Systems (2006) http://dx.doi.org/10.1007/s10458-006-7232-1.
3. WSDL: Web Services Description Language (2002) http://www.w3.org/TR/wsdl.
4. Singh, M.P.: An ontology for commitments in multiagent systems: Toward a unification of normative concepts. Artificial Intelligence and Law **7** (1999) 97–113
5. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language combining OWL and RuleML (May, 2004 (W3C Submission)) http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.
6. Protégé: The Protégé ontology editor and knowledge acquisition system (2004) http://protege.stanford.edu/.

7. OWL-P Examples: (Business protocols modeled with owl-p) http://research.csc.ncsu.edu/mas/OWL-P/.
8. Desai, N., Mallya, A.U., Chopra, A.K., Singh, M.P.: Interaction protocols as design abstractions for business processes. IEEE Transactions on Software Engineering **31** (2005) 1015–1027
9. OWL-P Project: (Software, tools, and documentation) http://projects.semwebcentral.org/projects/owlp/.
10. BPEL: Business process execution language for web services, version 1.1 (2005) www-106.ibm.com/developerworks/webservices/library/ws-bpel.
11. DAML Services Coalition: DAML-S: Web service description for the semantic Web. In: Proceedings of the 1st International Semantic Web Conference (ISWC). (2002)
12. WSCI: Web service choreography interface 1.0 (2002) wwws.sun.com/ software/ xml/ developers/ wsci/ wsci-spec-10.pdf.
13. Solanki, M., Cau, A., Zedan, H.: Augmenting semantic web service descriptions with compositional specification. In: Proceedings of the International World Wide Web Conference. (2004) 544–552
14. Gerede, C.E., Hull, R., Ibarra, O., Su, J.: Automated composition of e-services: Lookaheads. In: Proceedings of the International Conference on Service Oriented Computing. (2004)
15. Malone, T.W., Crowston, K., Herman, G.A., eds.: Organizing Business Knowledge: The MIT Process Handbook. MIT Press, Cambridge, MA (2003)
16. RosettaNet: Home page (1998) www.rosettanet.org.
17. Bergenti, F., Gleizes, M.P., Zambonelli, F., eds.: Methodologies and Software Engineering for Agent Systems. Kluwer (2004)
18. Bresciani, P., Perini, A., Giorgini, P., Guinchiglia, F., Mylopolous, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8** (2004) 203–236
19. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The Gaia methodology. ACM Transactions on Software Engineering Methodology **12** (2003) 317–370
20. Baïna, K., Benatallah, B., Casati, F., Toumani, F.: Model-driven web service development. In: Proceedings of Advanced Information Systems Engineering: 16th International Conference, CAiSE. (June 2004)