Toward Verification of Commitment Protocols and their Compositions *

Nirmit Desai, Zhengang Cheng, Amit K. Chopra, and Munindar P. Singh
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
{nvdesai, zcheng, akchopra, singh}@ncsu.edu

ABSTRACT

Commitment protocols have been proposed as a basis for modeling and enacting interactions among agents, such as those needed to carry out business processes. A central idea is that protocols would be developed and shared via libraries, and refined and composed to produce protocols that serve specific needs. Success in this program, therefore, presupposes that individual protocols and their compositions can be formally verified with respect to the properties of interest. This paper outlines an approach for verifying the correctness of commitment protocols and their compositions that exploits the well-known software engineering technique of model checking.

1. INTRODUCTION

Designing business processes for open settings is notoriously complex. Business partners are autonomous and heterogeneous, the configuration of a process can change dynamically, and exceptions frequently arise because of real-world or organizational problems. To address these challenges, Desai *et al.* propose an approach for modeling and enacting business processes based on specifying the interactions among autonomous, heterogeneous partners [1]. A *(business) protocol* is a public specification of an interaction. A protocol typically deals with a well-defined business purpose such as payment or order placement. Moreover, a protocol is specified in terms of the commitments among the different parties. Doing so reduces the emphasis on the rigid sequence of actions that the different parties must take, and enables flexible processing based on the commitments. Commitments also enable an independent observer to verify conformance which is a desirable property for agent communication [2].

A protocol serves as an interface, specifying only the key aspects of interactive behavior, not how the partners' software components are implemented. Each participant's perspective of the interaction, known as a *role*, can be extracted from a protocol. An *agent* is a computational entity that represents a business partner. An agent plays one or more roles and incorporates the private *business policies* of the partner it represents. A business process is enacted when the appropriate agents interact. Modeling interactions based on commitments facilitates loose coupling among the parties. Loose coupling promotes autonomy and heterogeneity, and yields a system that handles exceptions and opportunities perspicuously. By contrast, traditional approaches operate at a lower level, specifying the actions that different parties must take, and result in a tight coupling among business partners.

As a design abstraction, protocols are akin to software components. Protocols are inherently modular; they can be created ahead of time and stored in libraries. Particular protocols may then be selected and composed to support a desired business application. We are developing a framework, OWL-P, which supports the specification and composition of business protocols [1]. Composition is achieved by specifying a set of operational dependencies between the constituent protocols. This paper outlines extensions to the OWL-P framework by discussing automated verification of protocol compositions. Verifying a protocol composition is nontrivial as the constituent protocols may be complex, and their composition may exhibit subtle, potentially undesirable properties. Moreover, it is unlikely that a designer would be able to specify the correct dependencies at one shot. Like other intellectual efforts, protocol composition is best performed iteratively, through progressive refinement.

This paper discusses the steps needed to enable verification of protocols composed via OWL-P. Figure 1 summarizes the steps, which are briefly described in this paper.

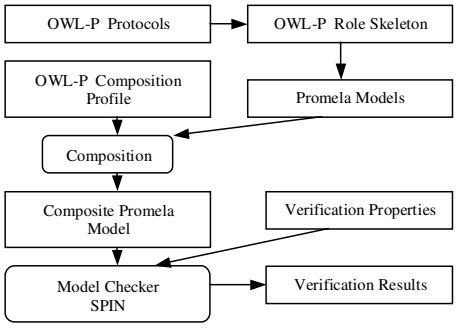

Figure 1: The steps for model-checking composite protocols

2. OWL-P PROTOCOLS

We adopt the protocol specification language OWL-P introduced

*This research was partially supported by the National Science Foundation under grant DST-0139037.

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

978-81-904262-7-5 (RPS) ©2007 IFAAMAS

by Desai *et al.*[1]. Protocols are sets of rules that specify patterns of interactions among partners from a global view. The meaning of these interactions is given by commitments.

A commitment $C(x, y, p)$ captures that an agent or role x is obligated to an agent or role y for a state of affairs in which condition p holds. Here x and y are known as the debtor and creditor of the commitment, respectively. The above are called *base-level* commitment. A *conditional commitment* is written $CC(x, y, p, q)$. It means that the base-level commitment $C(x, y, q)$ comes into being if and when the precondition (p) of the conditional commitment holds. Commitments support operations such as create, discharge, and others.

As an example, consider an *Order* protocol. The buyer requests a quote for an item; the seller responds with a quote. The semantics of quote is that it creates a commitment from the seller to the buyer guaranteeing delivery if the buyer pays the quoted price. In this scenario, the buyer accepts the quote. The semantics of accept is to create a commitment from the buyer to the seller to pay the quoted price if it receives the requested item. Below are the OWL-P rules for *Order* in the "antecedent ⇒ consequent" notation.

```
start ⇒ reqForQuote(?itemID)

reqForQuoteP(?itemID) ⇒
quote(?itemID, ?itemPrice) ∧
CC(S, B, pay(?itemPrice), goods(?itemID))

quoteP(?itemID, ?itemPrice) ∧
¬rejectQuote(?itemID, ?itemPrice) ⇒
acceptQuote(?itemID, ?itemPrice) ∧
CC(B, S, goods(?itemID), pay(?itemPrice))

quoteP(?itemID, ?itemPrice) ∧
¬acceptQuote(?itemID, ?itemPrice) ⇒
rejectQuote(?itemID, ?itemPrice)

acceptQuoteP(?itemID, ?itemPrice) ⇒ end

rejectQuoteP(?itemID, ?itemPrice) ⇒ end
```

Whereas protocols specify an interaction from a global view point, role skeletons derived from the protocols represent the role's perspective of the protocols. Essentially, the messages are sent and received in the local perspetive, instead of being exchanged as in global the perspective. Assuming that protocols for *Payment* and *Shipping* are similarly defined, OWL-P enables composing these three protocols to produce a new protocol, *Purchase*. Listing 1 shows the axioms for the composition.

Listing 1: Composition axioms

```
roleDefinition(define:Purchase.customer,
  unify:Order.buyer,
  unify:Shipping.receiver
  unify:Payment.payer)
roleDefinition(define:Purchase.merchant,
  unify:Order.seller,
  unify:Shipping.sender,
  unify:Payment.payee)
dataFlow(definition:Order.acceptQuote.itemID,
  usage: Shipping.reqForShipOptionsa.item)
dataFlow(definition:Order.acceptQuote.itemPrice,
  usage: Payment.authOK.amount)
implication(antecedent:Shipping.shipmentProp,
  consequent:Order.goods)
implication(antecedent:Payment.authOKProp,
  consequent:Order.pay)
eventOrder(earlier:Payment.authOKProp,
  later:Shipping.shipOrderProp)
```

The first two are *role definition* axioms stating that the roles being unified should be played by the agent that plays the new role being defined. Next two are *data flow* axioms stating that the usage slots should be assigned the value of the defining slot. Next two are *implication axioms* stating that the antecedent in one protocols implies the consequent in another protocol. The last is an *event order* axiom stating that a message in one protocol must precede a message in another protocol.

3. MAPPING PROTOCOLS TO PROMELA

Spin is a widely used model checker for verifying correctness requirements for concurrent systems. To use Spin, a designer must (manually or mechanically) generate models that capture the essential elements of a distributed system, and formulate requirements as linear temporal logic (LTL) expressions.

Thus, our goal is to construct a Promela model for the composite protocol *Purchase*, given the Promela models of *Order*, *Payment*, and *Shipping* and the set of composition axioms.

Each role skeleton maps to a Promela process. A role asynchronously receives the messages from a channel designated to it. The process is a loop having a case conditions for the antecedents of each of the rules. For each role, and for each message in the protocol, a boolean variable records whether the message has already been observed (sent or received) by the role. For each role, for each message in the protocol, and for each parameter of the message, a variable records the value of the parameter known to the role.

For role definition axioms, a new process for the role being defined is created, which contains the processes of the roles being unified. For data flow axioms, to the process of the using role, a restriction is added to the case of the corresponding rule for receiving the value of the slot from another protocol. The skeleton of the defining role is modified to send a value when the defining slot is bound. Event order axioms are mapped similar to data flow axioms, but instead of a data value being exchanges, a boolean flag is signalled indicating occurrence of the preceding message.

A commitment process CP maintains an array of commitments. Each of the commitment maintains its creditor and debtor, designated by their channels, to receive commitment messages. It also maintains a state and a slot for possible dependency between commitments. CP processes the messages concerning commitments from role processes participating in a composite protocol. It also monitors several channels for messages and updates the status of corresponding commitment. The process CP uses channels to receive messages corresponding to various commitment operations. The role processes update CP with appropriate messages as the interaction prgresses and events occur. Each role process maintains a channel to communicate with CP. For implication axioms, a message is sent from the role sending the specified antecedent to CP to update the state of the commitment for the condition specified as consequent.

4. VERIFICATION OF PROTOCOL CORRECTNESS

Equipped with the translated Promela models of composite protocols, we can formulate and verify various general and protocol-specific properties. Spin verifies correctness properties expressed in propositional linear temporal logic (LTL).

4.1 General Properties

The general properties relate to protocol-independent concerns. Spin verifies deadlocks and livelocks by default if end states are identified. Deadlocks can result from contradictory composition

axioms. The termination point for each role process is labeled as $term_\rho$. If $term_\rho$ becomes true, it can never again become false. The termination corresponds to the following LTL formula, where i ranges over the role identifiers:

$$\Diamond\Box \bigwedge_i \rho_i @ term_{\rho_i}$$

This states that it is eventually true that each process ρ_i will be at its termination point labeled with $term_{\rho_i}$.

All commitments do not need to terminate: it is typical and correct for a conditional commitment process to be in the state CONDITIONAL when the protocol computation comes to an end. However, a commitment that ends in the state BASE represents an active commitment: termination in a state where some base commitment is active indicates a violation. This property can be formulated as the following LTL formula, where id ranges over commitment identifiers:

$$\Diamond\Box \bigwedge_i cc[id].state \neq \mathsf{BASE}$$

This states that eventually all commitment should not be in their BASE state. This formula will check for all possibilities that will render the commitment violated.

It is important to verify whether all the messages produced are guaranteed to be consumed. In Promela, $ch?[msg]$ tests whether *msg* exists in channel *ch* to be consumed. For a given set of channels ch_i, the above property is captured by the following LTL formula:

$$\Diamond\Box\neg(\bigvee_i ch_i?[msg])$$

This formula captures the property that eventually there should be no message in any Promela message channel.

4.2 Protocol-Specific Properties

The following properties are based on our example purchasing protocol. It is important that the protocol can ensure that after the buyer sends the payment, it should eventually receive a corresponding shipment. The following LTL formula captures this property:

$$\Box(\textit{gateway\_authOK} \rightarrow \Diamond(\textit{shipper\_shipOrder} \wedge$$

$$\textit{buyer\_acceptQuote\_itemID} = \textit{receiver\_shipment\_item}))$$

This states that when *gateway_authOK* holds signaling money is cleared, the seller ships the product represented as *shipper_shipOrder*, and that the item received is the same as the item negotiated in the process.

From the view point of a seller, he would like to ensure that the buyer will pay the agreed amount, when the product is shipped. The following formula checks that if the shipment is sent, the buyer pays eventually:

$$\Box(\textit{shipper\_shipOrder} \rightarrow \Diamond(\textit{gateway\_authOK} \wedge$$

$$\textit{seller\_acceptQuote\_amount} = \textit{payee\_captured\_amount}))$$

In above formula, the variable $shipper\_shipOrder$ records that the seller has shipped the goods, and variable $gateway\_authOK$ being true means that payment is sent. The above formula states that it is always the case that if the seller ships products, eventually the seller will receive the payment of the agreed upon amount.

The buyer can choose to accept or reject a quote. However, when he rejects a quote, the goods would not be shipped. The following LTL formula verifies this:

$$\Box\neg(\textit{buyer\_rejectQuote} \wedge \textit{shipper\_shipOrder})$$

This states that it is always the case that if the buyer rejects a quote, no goods are shipped.

5. RELATED WORK

Venkatraman and Singh [7] presented an approach for testing whether the behavior of an agent in open systems complies with a commitment protocol with the commitment protocols specified in temporal logic. Using a formal model of logic-based negotiation, Wooldridge and Parsons [8] define two computational problems: *success* and *guaranteed success*, which loosely correspond to liveness properties. Our approach complements these works by supporting the verification of properties geared toward the composition of commitment-based protocols.

Several research efforts study verification of service compositions [4, 3, 5, 6]. These approaches are similar to each other in emphasizing the problems arising from asynchrony such as reachability, realizability, compatibility, and synchronizability. We focus on higher-level subtleties such as commitment compliance and other protocol-specific properties. Also, our mapping mechanism yields concise and modular Promela models, facilitating the verification of computationally intensive properties. The ability to verify the composition of protocols is novel to our approach.

6. REFERENCES

[1] N. Desai, A. U. Mallya, A. K. Chopra, and M. P. Singh. Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering*, 31(12):1015–1027, 2005.

[2] R. M. V. Eijk, F. S. D. Boer, W. V. D. Hoek, and J.-J. C. Meyer. A verification framework for agent communication. *Autonomous Agents and Multiagent Systems*, 6(2):185–219, March 2003.

[3] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Tool support for model-based engineering of web service compositions. In *Proceedings of the IEEE International Conference on Web Services*, pages 95–102, 2005.

[4] X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL web services. In *Proceedings of the 13th International World Wide Web Conference*, pages 621–630, 2004.

[5] S. Narayanan and S. A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the 11th World Wide Web Conference*, pages 77–88, 2002.

[6] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Automated synthesis of composite BPEL4WS web services. In *Proceedings of the International Conference on Web Services*, pages 295–301, 2005.

[7] M. Venkatraman and M. P. Singh. Verifying compliance with commitment protocols: Enabling open Web-based multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, Sept. 1999.

[8] M. Wooldridge and S. Parsons. Languages for negotiation. In *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI)*, pages 393–397, Aug. 2000.