Engineering Foreign Exchange Processes via Commitment Protocols

Nirmit Desai[§], Amit K. Chopra[§], Matthew Arrott[¶*], Bill Specht[‡†], and Munindar P. Singh[§]

[§]North Carolina State University, Raleigh, NC 27695-8206, {nvdesai, akchopra, singh}@ncsu.edu
[¶]University of California at San Diego, La Jolla, CA 92093, marrott@ucsd.edu
[‡]Currenex, Inc., 1700 Seaport Blvd., Redwood City, CA 94063, bill.specht@currenex.com

Abstract

Foreign exchange markets see a transaction volume of over $2 trillion per day. A number of standard ways of conducting business have been developed in the foreign exchange (FX) industry. However, current FX specifications are informal and therefore inherently ambiguous. Their implementations tend to be complex and their compliance unverifiable. This results in potential loss of value due to incompatible business processes and possible trades not consummated.

This paper validates a formal, protocol-based approach via the specification of foreign exchange processes, as standardized by the TWIST consortium. It shows how the desired processes can be based on a small, core set of foreign exchange interaction protocols. Each of these protocols is rigorously defined in terms of the commitments undertaken and manipulated by the parties involved. The core protocols can be composed to yield a large variety of possible processes. Our approach found several shortcomings in TWIST specifications, mainly that they were ambiguous, incomplete, and highly redundant. Also, novel business scenarios were discovered that traditional approaches would have missed.

1. Introduction

With *daily* total traded cash volume to the tune of $2.3 trillion, foreign exchange markets are huge [18] (over double the US stock market). Owing to the growth in international business and the globalization of enterprises, FX trading has increased by more than 30% in the last year and has more than doubled since 2001 [2]. Electronic trades have grown more than six fold in the last ten years and the trend is expected to continue [6].

To enable such a growth in electronic trading, standardization of the messages and workflows is crucial. TWIST (Transaction Workflow Innovation Standards Team) Process Innovations is a not-for-profit industry group of corporate treasurers, fund managers, banks, system suppliers, electronic trading platform and market infrastructure vendors, and professional services firms. TWIST has collaborated with other industry partners and standards organizations such as FPL (FIX Protocol Ltd.) [7] and ISDA (International Swaps and Derivatives Association) [10] to define standard good practice processes throughout the transaction processing life cycle for wholesale FX trades [17].

Current FX standards specifications describe business processes informally in the form of natural language descriptions accompanied by sequence diagrams representing typical scenarios. Moreover, FX processes (as in other businesses) typically have several dimensions of variation, e.g., trading with or without credit checks and trading with or without a trading service. Existing FX specifications treat each scenario as a separate case despite commonalities, making it harder to determine the relationships among such variations and whether they can be combined to serve a particular need. As a result, a large number of processes are explicitly specified. Managing and—equally importantly—understanding such large sets of standards is difficult.

Interaction-oriented approaches represent a growing trend in business process modeling [8, 20, 15, 19]. RosettaNet's Partner Interface Processes (PIPs) support billions of dollars of business each year [13]. PIPs are interaction-oriented, but informally specified and limited to two-party request-response interactions.

By contrast, this paper advocates a formal approach to modeling interactions [5]. Interactions among parties are treated as first-class modeling abstractions. The formal semantics enables reuse, refinement, and composition. To emphasize the contractual semantics involved in such processes, commitments among the parties are explicitly modeled and the messages are formalized by defining how they affect the extant commitments among the partners.

∗ Matthew Arrott is TWIST CTO.
† Bill Specht is TWIST Wholesale Trade Prime.

This paper validates the formal protocol specification and composition techniques for the TWIST foreign exchange processes. The goal is to identify a set of core protocols and formalize them as protocols [3] in the action description language $\mathcal{C}+$ [9]. Such protocols can be composed with each other in a variety of ways to derive the large set of possible combinations. This exercise helps identify ambiguities and gaps in the specification. Thus, a large set of processes can be engineered using a small set of modular protocols. More importantly, new business scenarios are discovered while composing protocols. Given the sheer scale, variety, and critical nature of the FX transactions, the impact of such an engineering approach can be enormous.

Although the main theoretical ideas applied here have been reported in previous papers, this paper (1) validates the protocols approach [15] via an extensive knowledge engineering exercise and (2) develops a methodology for creating protocols. Specifically, in the case of TWIST specifications, we show that they are, at best, ambiguous and incomplete. Moreover, they reflect little consideration of modularity and composition; we found that 28 TWIST processes can be specified in terms of 12 core protocols.

Section 2 outlines TWIST processes for price discovery. Section 3 models the commitments in price discovery and points some of its shortcomings. Section 4 shows how the TWIST processes may be obtained by composing elementary protocols.

2. Price Discovery Processes

This section describes the price discovery processes from the TWIST specification [17]. Figure 1 describes bilateral price discovery processes 7.2.1 and 7.2.2. A *Taker* is trying to discover the price whereas a *Maker* provides it. The *Maker* indicates in the priceResponse if an execution confirmation is required. If confirmation is required (7.2.2), the quoted price is not binding to the *Maker* even if the *Taker* accepts it. Otherwise (7.2.1), the *Maker* is bound to trade at the quoted price if the *Taker* accepts it. Here, executionConfirmation means the quoted and accepted prices are agreed upon and the deal is reached. In either case, the *Taker* may not reveal whether the requested currency is to be bought or sold, forcing the *Taker* to respond with both bid and offer quotes, thereby revealing the spread. The *Taker* can choose the direction of the trade when he accepts any of the quotes.

In addition to the messages shown in the figures, the *Taker* may cancel a request by sending cancelPriceRequest, or reject quotes by sending nothingDone. Also, the *Maker* may cancel a quoted price by sending cancelPrice. A priceResponse may also indicate a refusal to quote a price, and an executionConfirmation may also indicate failure to execute the deal.

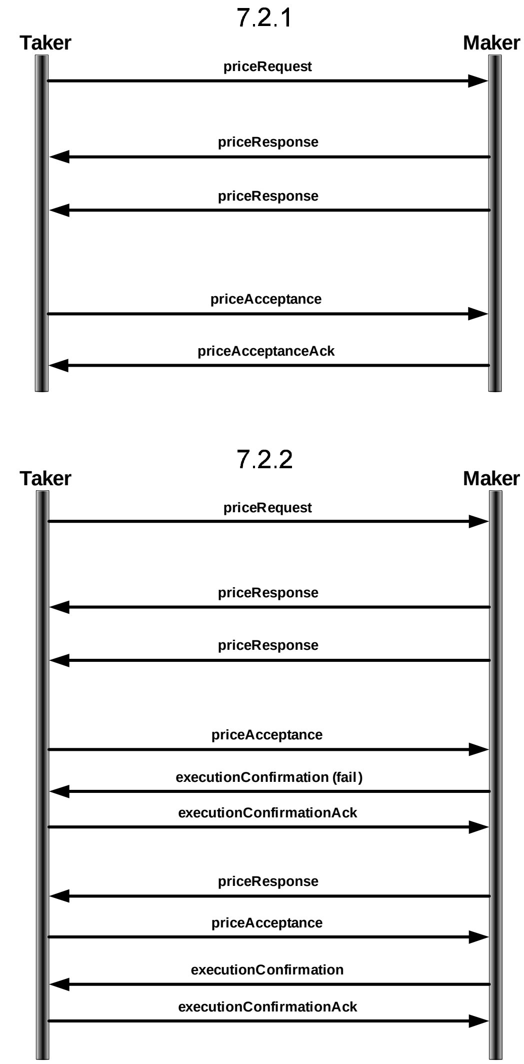

Figure 1. **Bilateral RFQ (7.2.1 and 7.2.2)**

Figure 2 describes multilateral price discovery wherein the *Taker* uses the trading service to discover the best price. The trading service may interact with multiple *Makers* to find a price for the *Taker*. To the *Makers*, this scenario is identical to the bilateral case. To the *Taker*, the only difference is that it receives responses from multiple *Makers*.

3. Protocols for Price Discovery

This section discusses the limitations of semiformal specifications (such as the current TWIST documents [17]) in terms of semantics, verifiability, and precision. It then formally specifies processes 7.2.1 and 7.2.2.

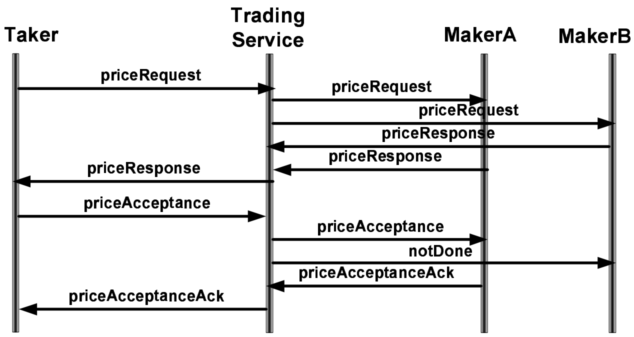

Figure 2. Multilateral RFQ (7.2.3)

3.1. Commitments in Price Discovery

Singh [14] formalized the notion of commitments. A commitment cc(x, y, p, q) obligates a *debtor* x to a *creditor* y for fulfilling the condition q if p holds. Here p is the *precondition* and q is the *condition* of the commitment. When the precondition is true (T), the commitment is termed a base commitment, else a conditional commitment.

Commitments can be manipulated: create, discharge, toBase (change to a base commitment), delegate (changing the debtor), assign (changing the creditor), release (creditor releasing the debtor from the commitment), and cancel (debtor canceling the commitment). Consider, for example, a scenario where a buyer and a seller are exchanging goods for payment. A conditional commitment cc(buyer, seller, goods, payment) denotes an obligation from the buyer to the seller that if the goods are delivered, the buyer will pay. In the event that the precondition goods holds, the conditional commitment changes to a base commitment cc(buyer, seller, T, payment). In the event that payment holds, the buyer's commitment is discharged and does not hold anymore. Note that the commitment does not imply a temporal ordering, i.e., payment may happen before goods, discharging the commitment.

Protocols declaratively specify the choreography of the messages exchanged among roles. They give messages a contractual semantics by defining how they affect the participants' commitments. For example, a message signifying shipment may cause the precondition goods, thereby causing the commitment to change to a base commitment. As a conversation progresses, commitments among the parties change to represent its evolving contractual state. Unless the precise meaning of the messages in terms of how they affect the extant commitments is specified, ambiguities ensue about the participants' obligations.

Assuming that the *Taker* is selling currency cur1 to the *Maker*, Figure 3 depicts various interpretations of the messages in 7.2.1. States denote newly created commitments in detail and commitments from previous states (if any) with

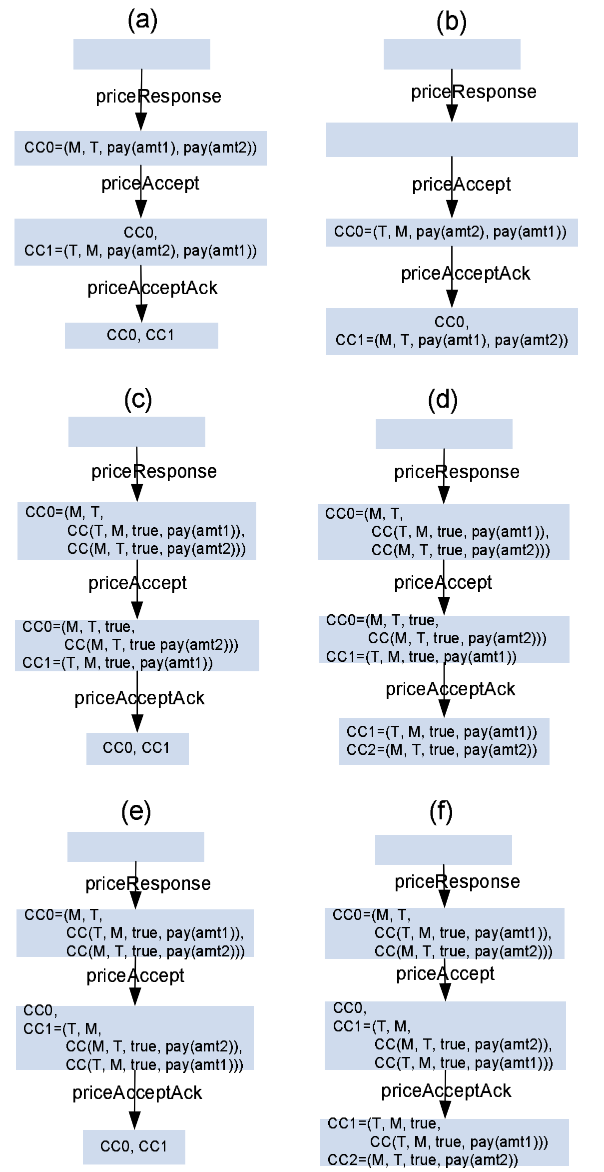

Figure 3. Possible interpretations of some TWIST messages in terms of commitments

their names. The *Maker* is denoted by M and the *Taker* by T. All the cases result in a state in which both parties have committed to each other for payment. However, there are subtle semantic differences in the way this state is reached. Also, the nested commitments as in (c), (d), (e), and (f) allow more flexibility than the unnested commitments as in (a) and (b) (as we shall see soon).

In (a), the *Maker* and the *Taker* commit to each other to pay via priceResponse and priceAcceptance, respectively. However, priceAcceptanceAck is superfluous in the sense that it does not affect the commitments. The final state is safe: regardless of the temporal ordering, payment is guaranteed if the commitments are not violated. In (b), priceResponse provides the price but is not binding. Whereas priceAcceptance has the same meaning as in (a), priceAcceptanceAck means that the *Maker* commits to the deal.

In (c), priceResponse creates a nested commitment: the *Maker* commits to the *Taker* to commit to paying if the *Taker* commits to paying. The condition and the preconditions being commitments enable the inner commitments to be delegated, assigned, and so on. For instance, the parties need not make payments themselves but may delegate the commitments to their banks. Without nesting, such flexibility would be lost. Precondition CC1 is caused by priceAccept, and priceAcceptanceAck is superfluous as in (a). The final state is safe as the *Taker* has committed and the *Maker* must commit to satisfying the nested commitment. In (d), the only difference is that priceAcceptanceAck is not superfluous and creates CC2.

In (e), the difference is that priceAcceptance causes a counter nested commitment instead of causing the precondition of CC0. Like in (c) and (e), priceAcceptanceAck is superfluous. In (f), the only difference is that the acknowledgment causes CC2—the condition of CC0.

The semantic differences among these variations highlight the importance of specifying such processes formally with commitment-based semantics. Similar interpretations exist for the case of 7.2.2. A benefit of the present formalization exercise was that it helped identify the possible points of confusion and disagreement, which would otherwise have been glossed over in the documents.

3.2. Choreography of Price Discovery

Although the typical scenarios are well-understood, an exhaustive set of possibilities can be covered rigorously and precisely only with formal methods. Our formalization exercise uncovered the following possibilities that are not clearly answered in the specification. For example, it is not clear whether nothingDone terminates the conversation or just rejects a specific priceResponse. Also, can two price responses in a conversation have different confirmation requirements? Can the time until which the quote is valid be different for different quotes? Even more troublesome, what if cancelPrice and priceAcceptance cross? If the confirmation was not required, should the *Maker* be allowed to cancel a quoted price? Maybe in such a case, the *Maker* would use priceAcceptanceAck to indicate failure, but that is not obvious.

What if the *Maker* confirms execution but the *Taker* does not acknowledge it? Or what if the confirmation is not required and the *Taker* accepts a price but the *Maker* does not acknowledge it? The commitments should be specified in such a way that the unresponsive party ends up violating a commitment.

Answering such questions is critical; if they are not answered, then it reflects gaps in the specification. Our approach may not supply the answers to these questions, but helps identify them via formal specification and verification techniques. This is the critical value of formal methods.

3.3. Specifying Protocols in $\mathcal{C}+$

Following Chopra and Singh [3], this paper specifies protocols in $\mathcal{C}+$, an action description language that gives primacy to causation [9]. $C+$ supports *elaboration tolerance* enabling refinement of a specification merely by adding to the existing specification; nothing need be removed from a specification even if the desired effect is that some inferences are disabled. For protocols, elaboration tolerance means that certain interactions can be added, removed, or modified simply by adding axioms to an existing specification.

A $\mathcal{C}+$ specification describes a transition system consisting of states and transition between them. A specification consists of a set of causal *laws*. A *fluent* is a proposition (true or false) whose value may change from state to state. *Actions* are performed by agents to *cause* fluents thus bringing about various conditions. Inertial fluents continue to hold unless an action occurs that causes their negation. The $\mathcal{C}+$ semantics ensures that all and only the caused fluents hold at any state. The general concepts relating to protocols are specified in $\mathcal{C}+$ as an ontology (Listing 1), to be included with specifications of individual protocols. The operator $++$ denotes logical OR and $<>$ denotes object inequality.

In $\mathcal{C}+$, an *exogenous* action is one that simply happens or not—the specification does not explain its cause. Messages are modeled as exogenous actions (line 9). The elaboration tolerance of $\mathcal{C}+$ allows placing constraints on the order of action occurrences. Inertial fluents (line 8) record the effects of all message occurrences (line 26). A static fluent initial ensures that the start state of a protocol is void of any fluents or commitments (lines 15, 21–24). Static fluents are not inertial; in each state the value of static fluents is determined by the value of other fluents in that state.

Commitments are modeled as inertial fluents (line 12) and their preconditions and conditions are modeled as actions (line 10) that are disabled by default (line 29). Also, occurrences of conditions are recorded in inertial fluents (line 11 and line 28). For simplicity, Listing 1 only describes create, discharge, cancel, and toBase (lines 13–14). Whereas discharge and toBase are caused when the appropriate conditions hold, other operations are caused directly by the actions of the parties.

Causing the conditions and preconditions of a commitment causes appropriate operations: discharge and toBase, respectively, provided the commitment is active or being created simultaneously (lines 31–35). If a commitment is discharged it is deemed fulfilled and ceases to hold (line 37). If toBase is caused, the original commitment ceases to exist, and a base level commitment is created, only if the original commitment is not being discharged or canceled simultaneously (lines 42–45). A commitment is asserted if create is caused and that commitment is not being simultaneously discharged, converted to base, being canceled, and the commitment does not already exist (lines 47–48). All commitment operations are disabled by default (lines 50–53). These laws collectively ensure correct behavior of commitment operations in the face of concurrent actions.

Listing 1: Protocol ontology

```
1  :- sorts Role; Slot; Message; Commitment; Condition.

3  :- variables
4    msg1 :: Message;   p,q :: Condition;
5    cc1 :: Commitment;  db1,cr1 :: Role.

7  :- constants
8    fl(Message) :: inertialFluent;
9    act(Message) :: exogenousAction;
10   cond(Condition) :: action;
11   fl_cond(Condition) :: inertialFluent;
12   comm(Commitment) :: inertialFluent;
13   create(Commitment), discharge(Commitment),
14   toBase(Commitment), cancel(Commitment} :: action;
15   initial :: sdFluent.

17 :- objects
18   T :: Condition;
19   CC(Role,Role,Condition,Condition) :: Commitment.

21 caused initial if initial.
22 caused -initial if comm(cc1).
23 caused -initial if fl(msg1).
24 caused -initial if fl_cond(p).

26 act(msg1) causes fl(msg1).

28 cond(p) causes fl_cond(p).
29 -cond(p) causes -cond(p).

31 caused discharge(CC(db1,cr1,p,q)) if cond(q) &
32 (comm(CC(db1,cr1,p,q)) ++ create(CC(db1,cr1,p,q))).

34 caused toBase(CC(db1,cr1,p,q)) if cond(p) &
35 (comm(CC(db1,cr1,p,q)) ++ create(CC(db1,cr1,p,q))) & p<>T

37 discharge(cc1) causes -comm(cc1).

39 cancel(cc1) & -discharge(cc1) causes -comm(cc1).

41 toBase(cc1) & -discharge(cc1)
42 & -cancel(cc1) causes -comm(cc1).

44 toBase(CC(db1,cr1,p,q)) & -discharge(CC(db1,cr1,p,q))
45 & -cancel(db1,cr1,p,q)) causes comm(CC(db1,cr1,T,q)).

47 caused comm(cc1) if create(cc1) & -(discharge(cc1)
48 ++ toBase(cc1) ++ cancel(cc1)) & -comm(cc1).

50 -create(cc1) causes -create(cc1).
51 -toBase(cc1) causes -toBase(cc1).
52 -cancel(cc1) causes -cancel(cc1).
53 -discharge(cc1) causes -discharge(cc1).
```

Messages, as exogenous actions, can happen on any transition by default. Protocols typically specify a set of restrictions on such messages and rules for their effect on commitments. As there are commonalities in 7.2.1 and 7.2.2, a common bilateral price discovery protocol *BPD* can be specified to cover all possibilities. Listing 2 specifies a rule governing the priceResponse message.

The parameters are declared variables of respective sorts. For each sort, relevant objects are declared, e.g., DONE and FAILED to indicate the result (res, res1, res2) in priceResponse and also in executionConfirmation. Variables resID and reqID (and similar) denote unique IDs for price response and request, respectively. Also, execConfReq can be YES or NO indicating whether or not a confirmation is required. The variable ttl indicates the valid time for the quoted rate as given in rate. As the request can be for two-way trades, the rate would typically include both bid rate and offer rate. We show one rate for simplicity. The dir in priceAcceptance indicates whether the *Taker* is buying or selling the currency cur1. Amounts involved in the currency pair cur1 and cur2 are amt1 and amt2, respectively. A disjunctive clause $[\backslash/a \mid f(a)]$ with variable a ground to distinct objects a_i is equivalent to $\bigvee_i f(a_i)$.

Listing 2: Specifying a rule for price response

```
nonexecutable act(priceResponse(m,t,resID,reqID,res,
                                execConfReq,ttl,rate))
if
-[\/cur1 \/cur2 \/amt1 |
  fl(priceRequest(t,m,reqID,cur1,cur2,amt1))]
++
(
  ( fl(nothingDone(t,m,resIDa)) ++
    fl(priceAcceptance(t,m,resIDb,dir))
  )
  &
 -( fl(executionConfirmation(m,t,resIDb,FAILED)) &
    fl(executionConfirmationAck(t,m,resIDb))
  )
)
++
act(priceResponse(m,t,resID,reqID,res2,
    execConfReq2,ttl2,rate2))
++
fl(priceResponse(m,t,resID,reqID,res,
   execConfReq,ttl,rate))
where
res<>res2 ++ execConfReq<>execConfReq2
++ ttl<>ttl2 ++ rate<>rate2.
```

The rule restricts occurrences of priceResponse if (a) no priceRequest with a matching reqID has happened, or (b) either a nothingDone has happened or a response has been already accepted, and confirmation on that acceptance has not yet failed and acknowledged, or (c) a priceResponse with the same ID but different result, confirmation requirement, ttl, or rate is happening simultaneously, or (d) a priceResponse with identical parameters has happened before. Note that (b) refers to confirmation as a priceResponse can not happen again after it has been accepted, but can happen again if executionConfirmation for the accepted price fails and an acknowledgment for this failure is sent (7.2.2, fourth, fifth, and sixth messages).

Listing 3 shows a specification of the nested commitment created as a result of a priceResponse. For brevity, only the case of confirmation not required is covered. A priceResponse creates the nested commitment (lines 1–3) as in cases (c), (d), (e), and (f) described in Section 3.1. However, to allow arbitrary levels of nesting, we substitute the inner commitments with place holder conditions that are caused when the inner commitments are created. For example, lines 5–8 cause the precondition of the nested condition if the *Taker* has accepted to buying cur1 within ttl and commits to paying for it in the other currency. Similarly, lines 10–12 cause the condition of the nested commitment if the *Taker* has accepted to buy and the *Maker* commits to paying in the currency being bought. Additional such rules would be needed for the case when the *Taker* is selling cur1.

Listing 3: Specifying a nested commitment

```
1  caused  create(cc(m,t,  priceResponsePrecond(resID),
2          priceResponseCond(resID)))  if
3  act(priceResponse(m,t,resID,reqID,DONE,NO,ttl,rate)).

5  caused  cond(priceResponsePrecond(resID))  if
6  fl(priceAcceptance(t,m,resID,TakerBuys))  &
7  -fl(ttlExpired(resID))  &
8  create(cc(t,m,T,pay(resID,amt2))).

10 caused  cond(priceResponseCond(resID))  if
11 fl(priceAcceptance(t,m,resID,TakerBuys))  &
12 create(cc(m,t,T,pay(resID,amt1))).
```

Messages such as priceAcceptance and priceAcceptanceAck would cause the creation of the inner commitments depending on the interpretation adopted from Figure 3. Here, we interpret the meanings as in the case (f). Also, a higher level of nesting can be modeled by having commitments as the conditions of the inner commitments. Rules for other messages and commitments can be specified similarly. Complete specifications are posted [1].

3.4. Querying the Specifications

The ability to query the formal specifications is crucial for discovering gaps, errors, and ambiguities. Protocol specifications can be compiled and queried via the causal calculator tool CCALC [16]. Essentially, it tries to find a model that satisfies the constraints of the specification, given a query. The following describes several important queries that can help uncover problems in the price discovery specifications.

Listing 4 specifies a query to see if it can ever happen that one of the party has a base commitment to another party but there is no counter commitment that either currently exists or has been fulfilled. Note that unfulfilled conditional commitments are safe, but same does not hold for base commitments. Thus, such a query should have no model in any protocol related to exchanges of any kind.

Here, p_i and q_i are variables of sort Condition. The label identifies this query and maxstep specifies the length of the history to be considered for search. Line 4 premises the query on the fact that initial holds in the starting state. The solver is asked to find the models for the formula of lines 5–10. Note that failure to find a model for this query is necessary but not sufficient to ensure commitment safety: a counter commitment may exist (resulting in no solutions), but it may not be a commitment with the intended condition.

Listing 4: Querying for commitment safety

```
1  :- query
2  label :: 1;
3  maxstep :: 0..infinity;
4  0: initial;
5  maxstep: ( comm(cc(t,m,T,p1)) &
6             -[\/q1 | comm(cc(m,t,T,q1)) ++ fl_cond(q1)]
7           ) ++
8           ( comm(cc(m,t,T,p2)) &
9             -[\/q2 | comm(cc(t,m,T,q2)) ++ fl_cond(q2)]
10          ).
```

Listing 5 specifies a query to see if the protocol does what is intended: get the deal done and end in a good state. The query formula represents the state at the end of case (f) with the case of *Taker* buying cur1. As fluents are inertial, such state queries are easy to formulate—it does not matter *when* the priceAcceptance happened, as long as it has happened in the history and the *Taker* has indicated to buy.

Listing 5: Querying for successful execution

```
:- query
label :: 2;  maxstep :: 0..infinity;  0: initial;
maxstep: comm(cc(t,m,T,priceAcceptanceCond(resID))) &
         comm(cc(m,t,T,pay(resID,amt1))) &
         fl(priceAcceptance(t,m,resID,TakerBuys)).
```

4. Composition of Protocols

With the repository of modular protocols specifications for the core FX interaction patterns, the natural next step would be to compose these modules to derive varieties of composite foreign exchange protocols as needed. The ability to reuse and compose existing protocols not only simplifies and improves engineering, but also provides new insights about the business processes. For example, the messages in 7.2.3 are not new; they have already been described in 7.2.1 and 7.2.2. The technique of protocol composition introduced by Desai *et al.* [5] enables defining 7.2.3 in terms of 7.2.1 and 7.2.2.

Say the formal bilateral price discovery protocol (*BPD*) is available. Figure 4 shows how multilateral price discovery (*MPD*) is specified by unioning two copies of *BPD* and stating additional constraints (known as *composition axioms*):

Ax1. MPD.Taker = BPD1.Taker
Ax2. MPD.Maker = BPD2.Maker
Ax3. MPD.TradeEx = BPD1.Maker, BPD2.Taker
Ax4. BPD1.priceRequest.cur1 $\rightsquigarrow$ BPD2.priceRequest.cur1
Ax5. BPD1.priceRequest.cur2 $\rightsquigarrow$ BPD2.priceRequest.cur2

Ax6. BPD1.priceRequest.amt1 $\leadsto$ BPD2.priceRequest.amt1
Ax7. BPD2.priceResponse.execConfReq $\leadsto$ BPD1.priceResponse.execConfReq
Ax8. BPD2.priceResponse.rate $\leadsto$ BPD1.priceResponse.rate

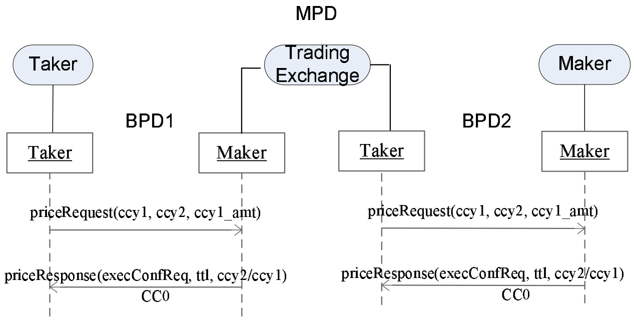

Figure 4. *MPD* by composing *BPD* with itself

Role definition axioms (first three in the above) define a new role on the left in the composite protocol in terms of the roles of the component protocols on the right. As a result, the roles of the component protocols are renamed to be the new role. In this example, the trading exchange role mediates between the traditional taker and multiple makers.

Data flow axioms (next five in the above) specify that the parameter on the right gets its value from the parameter on the left. Thus, the message on the right cannot happen until all the parameters it needs have been bound (i.e., the suitable messages have happened). In this example, data flow axioms specify the constraint that the currency pair and the amount for which the trading exchange requests the maker must be identical to those received from the taker. Also, the confirmation requirement and the rate indicated by the maker to the trading exchange should be propagated to the taker. Thus, the trading exchange reduces to a simple mediator. A new rule per axiom is added to the theory to effect the binding of the parameters and the temporal ordering of the messages. The result of the composition would be the formal specification *MPD* of 7.2.3, as posted [1]. Note that the resultant protocol *MPD* can be added to the protocol repository, and thus reused just like the core protocols. For example, Table 1 uses *MPD* (which is *BPD* $\oplus$ *BPD*) with *Credit* to derive 7.2.7. Additional kinds of composition axioms [5] are not needed here.

Various interesting business scenarios are possible depending on the composition axioms specified. Consider for example that Ax8 were not specified. This would mean that the trading exchange could act as a secondary price maker and manipulate the bid-offer spread received from the primary maker. If Ax7 were not specified, it would mean that the trading exchange could take risks of its own, and not require confirmation from the taker independent of the confirmation requirement indicated by the primary maker. Further, if Ax6 were not specified, the trading exchange could either fill the requested amount from multiple makers or fill multiple taker requests from a single maker deal. Thus, composition axioms act as elegant, vivid specifications of configuration parameters. Modifying the axioms enables us to model vastly different business requirements. These possibilities are lost when informal specifications are constructed. The TWIST specification is, at best, ambiguous about these possibilities. Highlighting such possibilities is an important contribution of this paper.

No.	Specification pattern	Protocols for pattern
1	7.2.1	*BPD*
2	7.2.2	*BPD*
3	7.2.3	*BPD* $\oplus$ *BPD*
4	7.2.4 (order)	*Order*
5	7.2.5 (order, cancel)	*Order*
6	7.2.6 (credit check)	*BPD* $\oplus$ *Credit*
7	7.2.7 (credit check–multi)	*BPD* $\oplus$ *BPD* $\oplus$ *Credit*
8	7.2.8 (price stream)	*ESP*
9	7.2.9 (price stream–multi)	*ESP* $\oplus$ *ESP*

Table 1. Modeling TWIST 7.2 interaction patterns in terms of protocols

Chapter 7 of the TWIST specification describes 28 interactions patterns in all. We have determined that, by using the methodology described above, such patterns can be modeled in terms of 12 core formally specified protocols and their compositions. More importantly, as demonstrated above, combinations *beyond* those described in the specification can be derived via novel compositions of the core protocols. Table 1 lists some of the patterns from Chapter 7.2, and shows how they can be modeled in terms of protocols. Here $\oplus$ denotes the composition of the operand protocols. Four protocols *BPD*, *Order*, *Credit*, and *ESP* are enough to model nine patterns. Table 1 also points to interaction possibilities not covered by the TWIST specification. For example, each of the processes 7.2.4, 7.2.5, 7.2.8, and 7.2.9 may be composed with credit checks, if needed.

5. Discussion

The idea of business processes based on conversation protocols is not new. WSCDL, a language for specifying such conversations among web services is being standardized by W3C [11]. Fu *et al.* specify conversation protocols as guarded automata [8]. Zaha *et al.* propose focusing on the global view of interaction among services in a SOA to see if all the constraints of the global interaction can be enforced locally [20]. However, as Section 3.1 demonstrates, a contractual semantics is essential to characterize business interactions unambiguously.

Singh *et al.* outlined a vision for commitment-oriented modeling for engineering large-scale business processes [15]. Winikoff provides a set of guidelines for designing and implementing interactions based on commitments [19]. Desai *et al.* offer intuitions behind composition of commitment protocols [5]. The above works, however, are not formal and thus do not support verification as an integral engineering activity.

Chopra and Singh have formalized protocols in $\mathcal{C}+$ [3]. Davulcu *et al.* proposed contractual semantics for interacting services [4]. Also, various verification techniques have been applied to services conversations [12, 8]. However, these approaches only formalize toy examples and have not been extensively evaluated.

For all of the above approaches, an evaluation and validation with a practical case study is lacking. This is a key point in favor of the present effort.

Further, the impact of applying the modeling abstractions such as commitments to the FX business processes has not been studied before. The impact is not limited to the FX processes alone. Rather, any such business interactions standards specifications can benefit from the methodology presented. The fact that the high-level specifications can be queried to check for inconsistencies and subtle semantic errors makes this methodology more valuable.

Conclusions

The contributions of this paper include:

- Evaluation and validation of protocol-based approaches.
- Formal methodology for specification and engineering of FX business processes
- Identification of gaps and ambiguities in specifications via the CCALC verification tool
- Discovery of new business scenarios by varying composition axioms
- Compactly representing a rich variety of business processes through the technique of protocol composition

Although it might appear that formal specifications are not as easy to develop as sequence diagrams, sequence diagrams must be accompanied by informal descriptions of the messages. Formal specifications account for such additional documentation as well. Rigor and precision are indispensable for critical business processes.

Future work includes the development of graphical tools to simplify the specification and verification of protocols and their compositions.

References

[1] Price discovery protocol specifications in $\mathcal{C}+$, 2007. http://research.csc.ncsu.edu/mas/ causal/.

[2] Bank for International Settlements. Triennial central bank survey of foreign exchange and derivatives market activity. http://www.bis.org/triennial.htm.

[3] A. K. Chopra and M. P. Singh. Contextualizing commitment protocols. In *Proceedings of AAMAS*, pages 1345–1352, 2006.

[4] H. Davulcu, M. Kifer, and I. V. Ramakrishnan. Ctr-s: a logic for specifying contracts in semantic web services. In *Proceedings of WWW*, pages 144–153, 2004.

[5] 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.

[6] Electronic Broking Services (EBS). History of EBS. http://www.ebs.com.

[7] FIX Protocol Ltd. (FPL). FIX 5.0 protocol specifications. http://tinyurl.com/auzx9.

[8] X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and verification of reactive electronic services. *Theoretical Computer Science*, 328(1–2):19–37, 2004.

[9] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1-2):49–104, 2004.

[10] International Swaps and Derivatives Association (ISDA). FpML 4.1 recommendation. http://tinyurl.com/3yle3u.

[11] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web services choreography description language 1.0 (w3c recommendation), 2005. http://www.w3.org/TR/ws-cdl-10.

[12] R. Kazhamiakin and M. Pistore. Static verification of control and data in web service compositions. In *Proceedings of ICWS*, pages 83–90, 2006.

[13] RosettaNet. Home page. www.rosettanet.org.

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

[15] M. P. Singh, A. K. Chopra, N. Desai, and A. U. Mallya. Protocols for processes: programming in the large for open systems. *ACM SIGPLAN Notices*, 39(12):73–83, 2004.

[16] Texas Action Group at Austin. The causal calculator CCALC. http://www.cs.utexas.edu/users/tag/cc/.

[17] Transaction Workflow Innovation Standards Team (TWIST). TWIST wholesale trade lifecycle. http://tinyurl.com/yswnqq.

[18] Wikipedia. Foreign exchange markets. http://en.wikipedia.org/wiki/ Foreign_exchange_market.

[19] M. Winikoff. Implementing commitment-based interaction. In *Proceedings of AAMAS*, 2007. To appear.

[20] J. M. Zaha, M. Dumas, A. ter Hofstede, A. Barros, and G. Decker. Service interaction modeling: Bridging global and local views. In *Proceedings of EDOC*, pages 45–55, 2006.