Specifying and Verifying Cross-Organizational Business Models: An Agent-Oriented Approach

Pankaj R. Telang and Munindar P. Singh, *Fellow, IEEE*

Abstract—Cross-organizational business processes are the norm in today's economy. Of necessity, enterprises conduct their business in cooperation to create products and services for the marketplace. Thus business processes inherently involve autonomous partners with heterogeneous software designs and implementations. Therefore, it would be natural to model such processes via high-level abstractions that reflect the contractual relationships among the business partners. Yet, in today's IT practice, cross-organizational processes are modeled at a low level of abstraction in terms of the control and data flows among the participants. This paper makes the following contributions. First, it proposes a simple, yet expressive declarative way to specify business models at a high level based on the notion of commitments. Second, it shows how such a high-level model maps to a conventional operational model. Third, it provides a basis for verifying the correctness of the operational representations with respect to the declarative business model using existing temporal model checking tools. This paper validates the above claims using the well-known Quote To Cash business process, e.g., as supported by vendors such as SAP and applied in large enterprises. In this manner, this paper helps bridge the gap between high-level business models and their IT realizations.

Index Terms—Methodologies, patterns, specification, model checking.

1 Introduction

No enterprise is an island. In the modern economy, especially, enterprises are best understood as participating in business ecosystems where they carry out subtle interactions with one another. A business process describes how an enterprise conducts some element of its business. We are particularly interested in *cross-organizational business processes*, which involve the aspects of business that cut across enterprise boundaries.

A distinguishing feature of cross-organizational business processes is that they involve the interactions of mutually independent (autonomous and heterogeneous) business partners [1]. For this reason, agents provide a natural metaphor with which to model the partners that engage in a cross-organizational business process [2]. The business relationships among the partners are key both to characterize a narrow and correct interface among them and to enable them to alter their internal implementations with minimal effect on each other.

In contrast, with only a few notable exceptions, much of the existing technical work on business models emphasizes low-level or operational details of the interactions. Existing approaches specify the control and data flow among the business partners, but they fail to specify their business relationships. Even standards such as BPEL [3] and WS-CDL [4] mandate the exchange of messages in an unnecessarily restrictive temporal order, while failing to capture the business intent of the interactions. Such business relationships are known to business analysts and motivate the specification of the processes. Therefore, losing track of the relationships in the operational perspective is unfortunate: losing the notion of business correctness by which to judge operational execution complicates the creation and maintenance of each partner's software systems. For example, in a simple purchasing scenario, the business intent is for the buyer and seller to exchange goods for payment. The order in which the goods and the payment are brought about is unimportant, and either operational ordering is correct at a business level.

Part of the reason for the above is the lack of suitable business modeling approaches. We review the literature in Section 6, but we summarize the situation here. Recent business modeling approaches talk of high-level concepts but are not sufficiently technical or operationalizable. In contrast, the traditional approaches such as BPEL talk of operational notions but lack high-level concepts. Accordingly, to fill the above gap, we propose a high-level approach for specifying cross-organizational business models. Commitments [1], [5], [6], [7] are an extensively studied notion in agent research, and can help specify the essence of business interactions in a natural manner. A commitment abstracts from operational details, minimally constraining business executions. Through a set of well-defined operations, commitments enable modeling complex cross-organizational business relationships.

In our approach, a *business model* is a high-level specification of how (cross-organizational) business is to be conducted. We develop reusable, composable patterns that business analysts can employ to develop a business model for a desired scenario. First, a model composed from these patterns serves as a formal specification that can be used to verify an operational model defined in any technical standard: for concreteness, we adopt UML

• *The authors are with the Department of Computer Science, NC State University, Raleigh, NC 27695. E-mail: {prtelang,singh}@ncsu.edu*

Manuscript received 03 Jun. 2010; revised 16 Nov. 2010; accepted 25 Dec. 2010; published online XX Jan. XXXX.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2010-16-0082. Digital Object Identifier no. 10.1109/TSC.2011.XX.

2.0 sequence diagrams [8] here. Second, organizations frequently migrate their business process implementations to newer technologies to benefit from the improvements those technologies offer. In such a case, a business model provides a basis for establishing the correctness of the new implementation.

An operational model captures the interactions that realize a business model. By *verification*, we mean checking if a given *operational model* satisfies a given *business model*. It is important to perform such verification during the early phases of development to prevent costly rework during the later phases in case of misalignment between the two models.

Our verification approach can be used in multiple methodologies. In the simplest methodology, a business analyst would develop a business model (e.g., by composing business patterns from a library). An IT analyst would develop an operational model for implementing the business model. Our approach (based on an automated verification tool) is to verify that the operational model satisfies the business model prior to creating or modifying the implementation. In case verification fails, the tool generates an execution scenario that exemplifies the failure. Based on this scenario, the business or IT analyst would modify one or both of the two models and iterate. In a top-down methodology, the operational model would be generated from the business model, and then verified as above. The top-down methodology may be used in a disciplined setting where the specifics of the processes are known from previous experience. In a bottom-up methodology, the business model would be induced from an existing operational model based on analyst insights. This is necessary when no business model exists. In either methodology, business and operational models would then be compared as above.

Contributions: The key contributions of this paper are as follows. First, it proposes a simple, yet expressive declarative way to specify business models at a high level based on the notion of commitments. Second, it shows how such a high-level model maps to a conventional operational model. Third, it provides a basis for verifying the correctness of the operational representations with respect to the declarative business model using existing temporal model checking tools. This paper validates the above claims using the well-known Quote To Cash business process, e.g., as supported by vendors such as SAP and applied in large enterprises.

Organization: Section 2 introduces our commitment-based business metamodel and shows the main patterns of business interactions, as expressed in our model. Section 3 evaluates our patterns by applying them on a real-world quote-to-cash business process. Section 4 presents our approach for formalizing our business patterns in temporal logic in a manner as to enable model checking. Section 5 evaluates our verification approach by applying it on the same quote-to-cash process. Section 6 reviews the related research. Section 7 summarizes our results. The online appendix describes a way to map a UML sequence diagram to NuSMV, an additional pattern, and a second case study.

2 Approach: Metamodel and Patterns

We concern ourselves with business models that involve two or more participants. The business *partners*, abstracted as *roles*, participate in a *business relationship*. The participants create, manipulate, and satisfy *commitments* in each relationship. They execute *tasks* for each other that enable them to achieve their respective *goals*.

Three distinct phases characterize business execution. First, in the *agreement* phase, participants enter into an agreement, and *create* commitments toward each other. Second, in the *assembly* phase, the participants *delegate* or *assign* commitments to others. A participant may delegate a commitment to gain competence or price advantages. Third, in the *enactment* phase, participants execute tasks to *satisfy* their commitments.

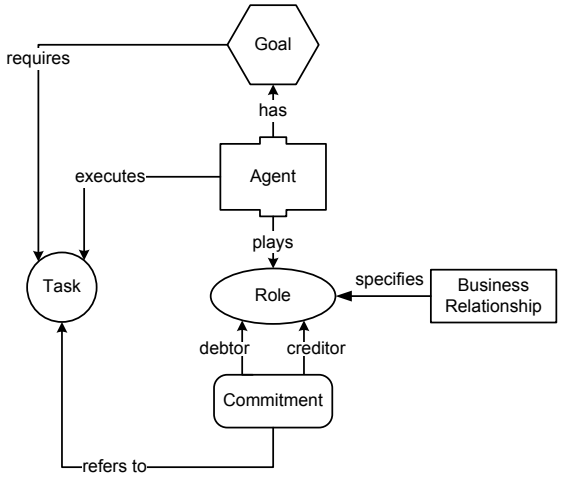


Fig. 1. A commitment-based business metamodel.

The following are the main concepts of our business metamodel, as shown in Figure 1, adopted from [9].

Agent: a computational representation of a business partner. An agent has goals, and executes business tasks. For each business relationship in which an agent participates, it enacts one or more roles.

Role: an abstraction over agents that helps specify a business relationship in templatic form. Each role specifies the commitments expected of the agents who play that role along with the tasks they must execute to function in that role.

Goal: a state of the world that an agent desires to be brought about, i.e., an achievement goal [10]. An agent achieves a goal by executing appropriate tasks or negotiating with other agents to have them execute appropriate tasks.

Task: a business activity viewed from the perspective of an agent.

Commitment: an element of a contractual business relationship. A commitment C(DEBTOR, CREDITOR, antecedent, consequent) denotes that the DEBTOR commits to the CREDITOR to bring about the consequent

if the antecedent begins to hold [7]. A commitment, when active, functions as a directed obligation from a debtor to a creditor. However, unlike a traditional obligation, a commitment may be manipulated, e.g., delegated, assigned, or released.

Business relationship: a set of interrelated commitments among two or more roles that describe how they carry out the given business process.

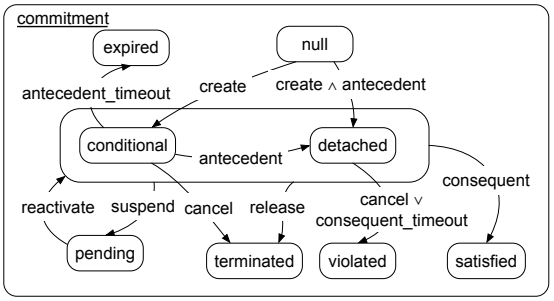


Fig. 2. The life cycle of a commitment (based on [11]).

Since the notion of commitment is central to our approach, we describe commitments in greater detail now. Figure 2 shows the life cycle of a commitment as a state diagram based on [11]. The rounded rectangles represent the states, and the directed edges represent the transitions. The label on a rectangle is the commitment state, whereas the label on an edge is an action or an event that causes the transition. The edge is directed from the start state to the end state of the transition. At runtime, commitments arise between agents, but at design-time we specify them between roles.

A commitment can be in one of the following states: *null*, *conditional*, *expired*, *detached*, *satisfied*, *pending*, *terminated*, or *violated*. Before a commitment is created, it is *null*. Only a debtor may *create* a commitment. By default, a commitment is *conditional*, meaning that its antecedent is not true. For a *conditional* commitment, an *antecedent timeout* may occur if neither its antecedent nor its consequent is brought about within a specified time period. In that case, the commitment *expires*. In our treatment, an expired commitment cannot be reactivated and remains forever expired. If the antecedent of a *conditional* commitment is brought about prior to the antecedent timeout (even immediately upon creation), then the commitment is *detached*. Similar to an antecedent timeout, for a *detached* commitment, a *consequent timeout* may occur if its consequent is not brought about, causing the commitment to be *violated*. Alternatively, the debtor may *cancel* a detached commitment, thereby also violating it. Conversely, if the consequent of a *detached* commitment is brought about prior to the consequent timeout, it is *satisfied*. After a commitment is *satisfied*, if its antecedent is brought about, it remains *satisfied*. A commitment can be placed in the *pending* state, i.e., suspended, as explained below. A *pending* commitment may be reactivated. A commitment is *terminated* if the debtor *releases* the creditor from the commitment, or *cancels* a conditional commitment.

Note that for ease of modeling, we write the antecedent and consequent timeouts separately, but incorporate them into the antecedent and consequent, respectively, in the formalization. That is, we can write the corresponding commitment as C(DEBTOR, CREDITOR, antecedent $\wedge \neg$antecedent_timeout, consequent $\wedge \neg$consequent_timeout).

2.1 Business Model Patterns

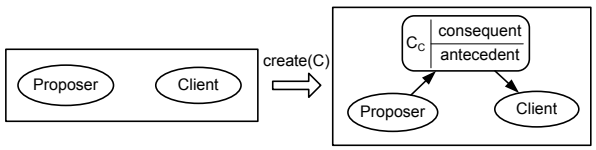
Being able to manipulate commitments yields the flexibility needed in open interactions. A pattern, in the present setting, is a recipe for modeling recurring business scenarios in terms of manipulations of commitments. This section describes a key set of such patterns, adopted from [9], which could seed a potential business model pattern library. Section 3 demonstrates the effectiveness of this simple set of patterns on the Quote To Cash business process.

We use the attributes *name*, *intent*, *motivation*, *implementation*, and *consequences* to describe our patterns [12]. Here the *consequences* of a pattern allude to the practical consequences of applying the pattern, i.e., the assumptions underlying the model.

We express a pattern in terms of commitments, where each commitment references roles and tasks. A pattern would be instantiated by the agents who adopt the specified roles. A set of commitments fully specifies a pattern. A pattern diagram illustrates the progression of the commitments in a typical execution. The diagrams use the notation of Figure 1, and additionally show two directed edges for each commitment: from the debtor to the commitment and from the commitment to the creditor. The subscript on a commitment indicates its state: **C** for conditional, **D** for detached, **S** for satisfied, and **P** for pending. The expired and violated states do not appear in the pattern diagrams since the diagrams illustrate normal execution, that is, an execution in which none of the commitments expire or are violated.

2.1.1 Conditional Offer Pattern

This is the simplest possible pattern. It merely views a commitment (as described in Figure 2) as an offer.



C(PROPOSER, BENEFICIARY, antecedent, consequent)

Fig. 3. The conditional offer pattern.

Intent: A proposer conditionally offers to execute a task for a client.

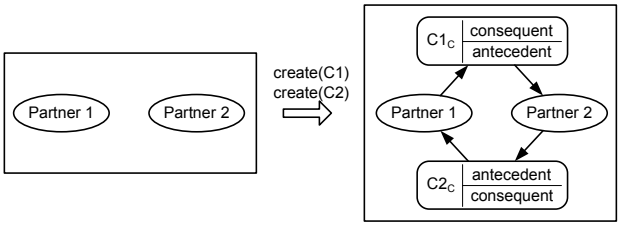
Motivation: For example, a conference committee member commits to a program chair to review a paper

that the program chair requests the member to review. The chair makes no converse commitment.

Implementation: A commitment is created in which the proposer is the debtor, the client is the creditor, the consequent is the task that the proposer executes, and the antecedent is a condition that brings the commitment into force. Figure 3 shows this pattern.

Consequences: This pattern presumes a benefit to the proposer from the antecedent of the commitment.

2.1.2 Commercial Transaction



C_1 C(PARTNER 1, PARTNER 2, antecedent, consequent)
C_2 C(PARTNER 2, PARTNER 1, consequent, antecedent)

Fig. 4. Commercial transaction.

Intent: This pattern expresses a value exchange between two trading partners. The trading partners negotiate and, upon agreeing, commit to executing certain tasks for each other.

Motivation: A typical situation would be where a seller and a buyer agree to exchange goods for payment or, generally, to barter goods or services.

Implementation: A pair of reciprocal commitments between the trading partners treated symmetrically specify the pattern. Figure 4 shows this pattern.

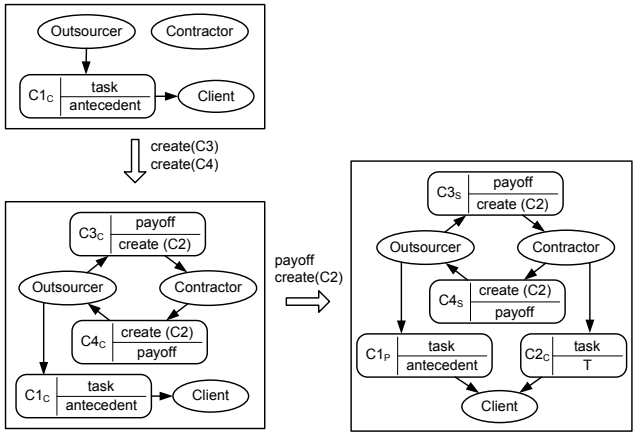
Consequences: In general, the antecedents and consequents of the commitments are composite expressions. Importantly, we need a mechanism to ensure progress by in essence breaking the symmetry, e.g., via a form of concession [13].

2.1.3 Outsourcing

Intent: An outsourcer delegates a task to a subcontractor, typically because the outsourcer lacks the necessary capabilities or expects some other benefit, such as reduced costs or a lower risk of failure.

Motivation: Many business organizations outsource noncore activities. Consider a customer who signs up for cable television service. The cable operator commits to the customer for performing the installation. Instead of staffing its entire service area directly, the cable operator outsources the installation task to its local partners in various regions.

Implementation: The outsourcer has a commitment C_1 towards its client to execute a task. The outsourcer and the contractor negotiate, and agree that the contractor will create the commitment C_2 to execute the task if the outsourcer pays. Conversely, the outsourcer commits to paying the contractor if the contractor creates C_2. Note that the antecedent of this commitment is true ($\top$), which means that it is detached. We say that the commitment C_2 *covers* the commitment C_1. Eventually when the contractor creates C_2 the original commitment becomes pending. Figure 5 shows this pattern.

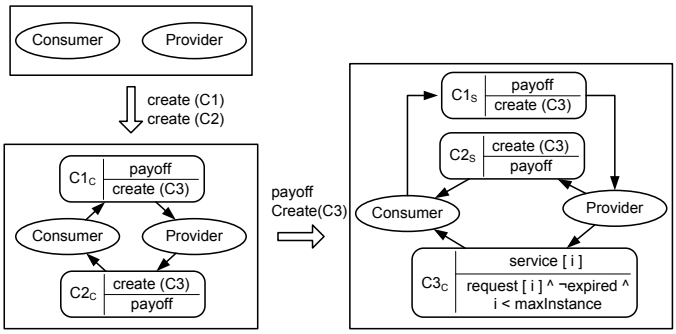


C_1 C(OUTSOURCER, CLIENT, antecedent, task)
C_2 C(CONTRACTOR, CLIENT, $\top$, task)
C_3 C(OUTSOURCER, CONTRACTOR, create(C2), payoff)
C_4 C(CONTRACTOR, OUTSOURCER, payoff, create(C2))

Fig. 5. Outsourcing.

Consequences: The commitment from the outsourcer is pending and must either be discharged or reactivated depending on how the contractor performs.

2.1.4 Standing Service Contract



C_1 C(CONSUMER, PROVIDER, create(C3), payoff)
C_2 C(PROVIDER, CONSUMER, payoff, create(C3))
C_3 C(PROVIDER, CONSUMER, request[i] $\wedge$ ¬expired $\wedge$(i $\leq$ maxInstance), service[i])

Fig. 6. Standing service contract.

Intent: A provider sells a long-lived service to a consumer. The service can be bounded by a combination of duration and the total number of requests.

Motivation: A business service such as building maintenance refers to (potentially) numerous service instances. Whenever the faucet leaks (within specified limitations), a plumber will be sent to fix it.

Implementation: The provider and consumer negotiate, and upon agreement, create a pair of commitments

C_1 (the consumer commits to paying the service provider if the service provider commits to provide service), and C_2 (the converse of C_1). Commitment C_3 is the standing service contract. In C_3, the provider commits to the consumer to serve each request prior to expiration up to a fixed number of requests. Figure 6 shows this pattern.

Consequences: Each service request should take a bounded amount of effort.

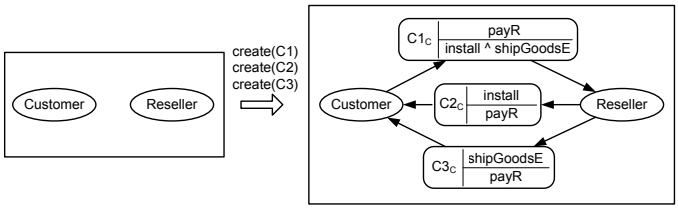
2.1.5 Applying the Patterns

Our patterns capture recurring business scenarios in a natural manner. A business modeler analyzes a scenario description to identify its agents (e.g., FedEx), tasks (e.g., shipping), and business-specific roles (e.g., Shipper). The modeler develops the business model by successively applying the relevant patterns, identifying each applicable patterns based on its specified *intent*. To apply each pattern, the modeler associates each business role (e.g., Shipper) with the appropriate pattern role (e.g., Contractor) and the tasks with the antecedents and consequents of the relevant commitments.

The patterns compose naturally when the business roles referenced by the patterns overlap. Our graphical representation emphasizes the composition by showing one node for each business role, thereby highlighting the interrelationships among the commitments from the composed patterns.

3 Evaluation: Applying the Patterns

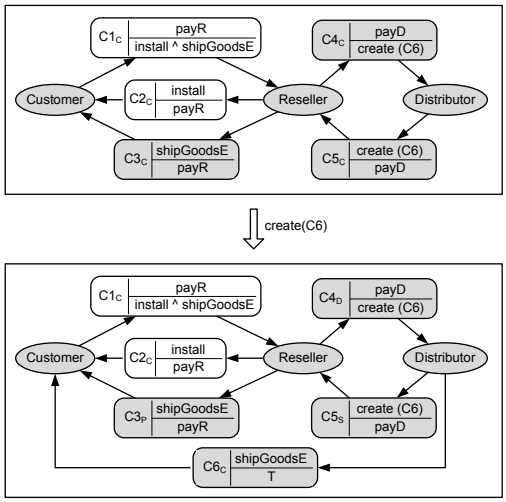
This section evaluates our proposed methodology on the Quote To Cash (QTC) business process. It is loosely based on the public descriptions of Cisco System's QTC implementation. Cisco is in the business of selling networking products and services. The QTC process encompasses all of the key business activities that begin from a customer requesting a quote, and end in Cisco receiving cash from the customer.



C_1 C(CUSTOMER, RESELLER, install $\wedge$ shipGoodsE, payR)
C_2 C(RESELLER, CUSTOMER, payR, install)
C_3 C(RESELLER, CUSTOMER, payR, shipGoodsE)

Fig. 7. Commercial Transaction: Customer purchases the goods and the installation service from the reseller.

Some of the key partners in this process are customers, resellers, distributors, logistics providers, banks, contract manufacturers, and service providers. These participants engage in a number of complex interactions for transacting business. The resellers and the distributors serve as sales channels. A customer purchases goods either directly from Cisco, or from a reseller. In addition to selling the goods, a reseller also provides value-added services of installing and configuring the goods. A reseller purchases goods either from a distributor, or from Cisco. A distributor always purchases goods from Cisco. Unlike a reseller, a distributor may purchase goods, and stock the goods in its warehouse. To build its products, Cisco utilizes a set of contract manufacturers, and for shipping a set of transportation providers. The participants use different banks and credit companies for making payments.

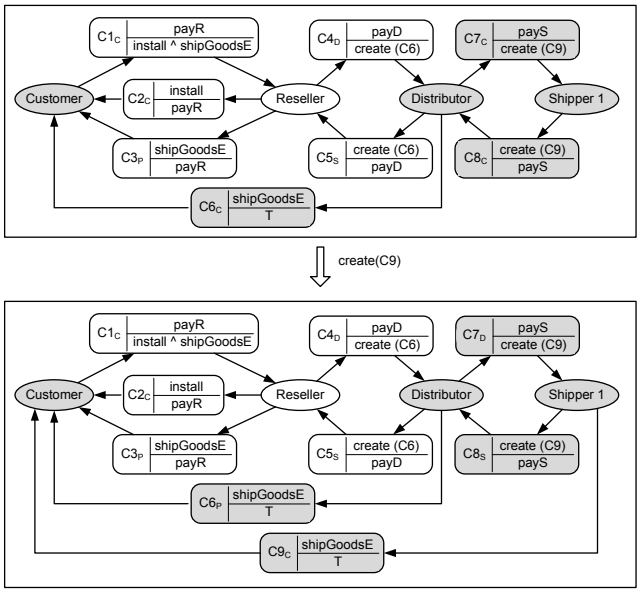


C_4 C(RESELLER, DISTRIBUTOR, create(C_6), payD)
C_5 C(DISTRIBUTOR, RESELLER, payD, create(C_6))
C_6 C(DISTRIBUTOR, CUSTOMER, $\top$, shipGoodsE)

Fig. 8. Outsourcing: Reseller outsources the shipping of the goods to the distributor.

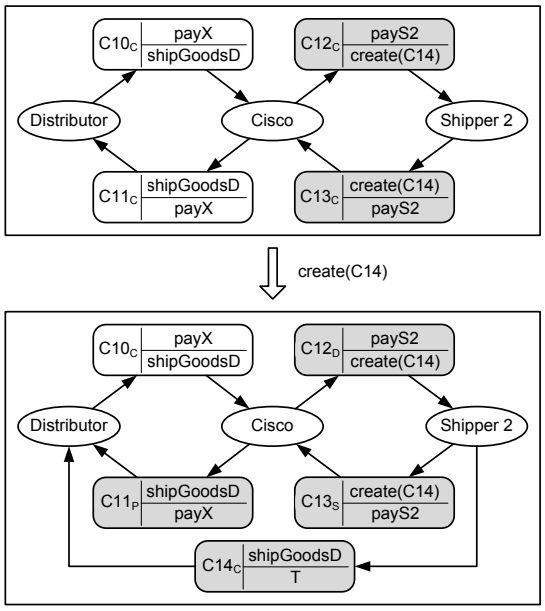
Next we apply our approach to a fragment of the QTC process. To satisfy a business goal, a customer desires to install Cisco products. The customer selects a suitable reseller for the purchase. The *commercial transaction pattern* models this scenario. The customer and the reseller negotiate the purchase price. Upon agreement, they create commitments C_1, C_2, and C_3, as Figure 7 shows. The customer commits (C_1) to the reseller to pay if the reseller ships and installs the goods, and the reseller commits to the customer to ship (C_2) and install (C_3) the goods if the customer pays. Note that we model the commitment to ship (C_2), and the commitment to install (C_3) as separate commitments since the reseller outsources the shipping (C_3) to a distributor, but installs the goods by itself (C_2).

The reseller outsources to a distributor its commitment C_3 to the customer to ship the goods. Figure 8 applies the *outsourcing pattern* to this scenario. The reseller and the distributor negotiate, and upon agreement, create C_4 and C_5. The reseller commits (C_4) to the distributor to pay if the distributor commits to (C_6) ship the goods to the customer. The distributor conversely commits (C_5) to create C_6 if the reseller pays the distributor. When the distributor creates C_6, C_5 discharges, C_4 detaches,



C_7 C(DISTRIBUTOR, SHIPPER1, create(C_9), payS)
C_8 C(SHIPPER1, DISTRIBUTOR, payS, create(C_9))
C_9 C(SHIPPER1, CUSTOMER, $\top$, shipGoodsE)

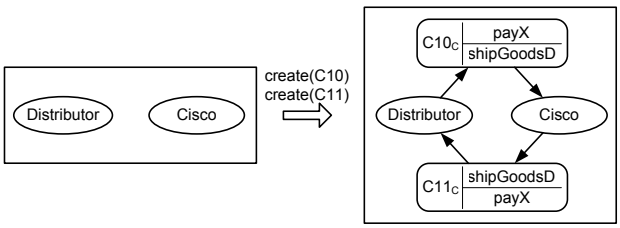
Fig. 9. Outsourcing: Distributor outsources shipping to a shipper.



C_{12} C(CISCO, SHIPPER2, create(C_{14}), payS2)
C_{13} C(SHIPPER2, CISCO, payS2, create(C_{14}))
C_{14} C(SHIPPER2, DISTRIBUTOR, $\top$, shipGoodsD)

Fig. 11. Outsourcing: Cisco outsources shipping.

and C_3 transitions to the pending state. The reseller may pay (payD) the distributor either before or after the distributor commits (C_6) to shipping the goods to the customer. Figure 8 shows a possible progression in which the distributor commits.

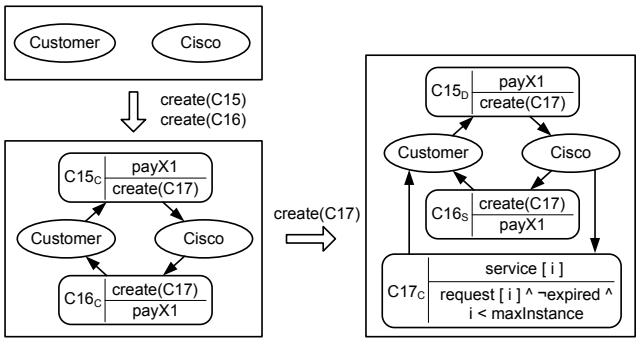


C_{10} C(DISTRIBUTOR, CISCO, shipGoodsD, payX)
C_{11} C(CISCO, DISTRIBUTOR, payX, shipGoodsD)

Fig. 10. Commercial Transaction: Distributor buys goods from Cisco.

Figure 9 shows another scenario modeled by the *outsourcing pattern*. If the distributor has the goods in stock, it outsources the shipping of the goods to a shipper. The distributor commits (C_7) to the shipper to pay if the shipper commits (C_9) to ship the goods to the customer. The shipper conversely commits (C_8) to create C_9 if the distributor pays the shipper. The shipper satisfies C_8 by creating C_9, which detaches C_7, and causes C_6 to transition to the pending state.

If the distributor lacks the stock of goods that it needs to ship to the customer to satisfy C_9, it purchases the goods from Cisco. Figure 10 shows how the *commercial transaction pattern* models this purchase. Upon successful negotiation, the distributor and Cisco create the reciprocal commitments C_{10} and C_{11}. The distributor commits (C_{10}) to Cisco to pay if Cisco ships the goods to the distributor. Conversely, Cisco commits (C_{11}) to the distributor to ship the goods if the distributor pays Cisco.



C_{15} C(CUSTOMER, CISCO, create(C_{17}), payX1)
C_{16} C(CISCO, CUSTOMER, payX1, create(C_{17}))
C_{17} C(CISCO, CUSTOMER, reqService[i] $\wedge$ $\neg$expired, service[i])

Fig. 12. Service Contract: Customer contracts with Cisco.

In Figure 11, Cisco outsources the shipping of the goods to the distributor, to a shipper. The *outsourcing pattern* models this scenario, and the figure shows the commitments that are created.

Figure 12 shows how the service contract pattern applies to the scenario in which the customer buys a service contract from Cisco. The customer commits (C_{15}) to Cisco to pay if Cisco creates the service commitment C_{17}. Cisco commits to the customer to create the service commitment C_{17} if the customer pays Cisco. Here C_{17} means that Cisco commits to the customer to provide

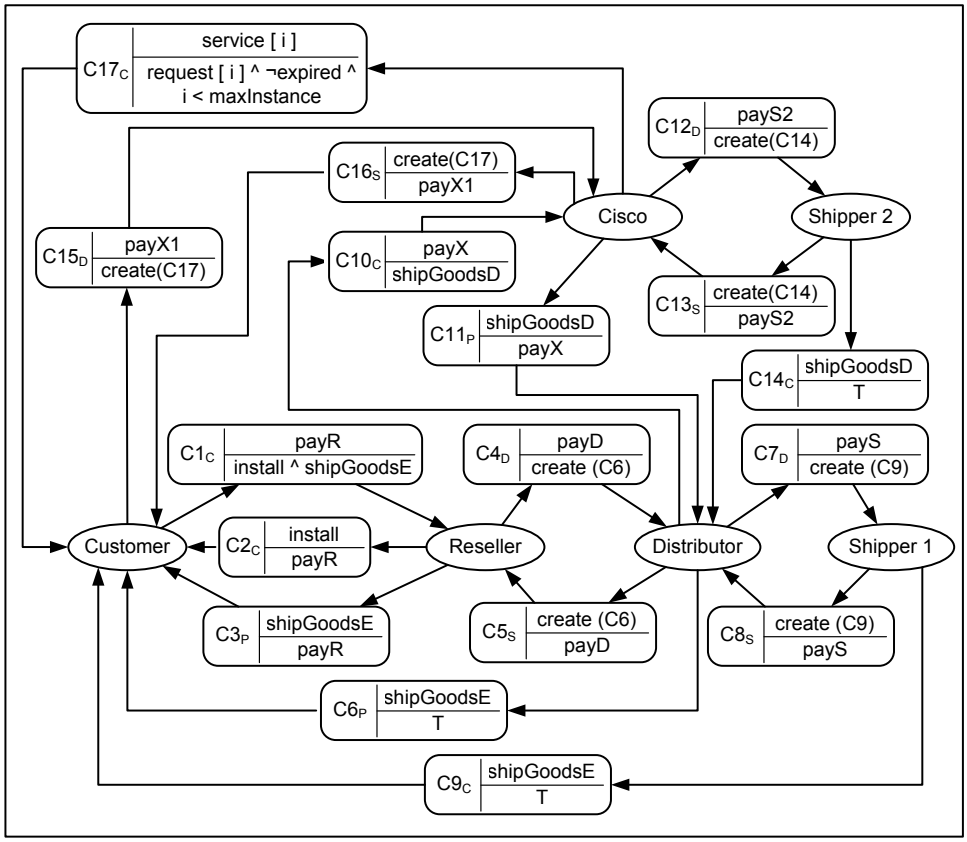


Fig. 13. The Quote To Cash (QTC) business process expressed as a business model.

service on the goods if the customer requests the service prior to its expiration. The customer can request the service multiple times. To satisfy C_{17}, Cisco needs to provide service for each of those requests as long as the request is made prior to C_{17}'s expiration.

The above exercise shows how one can naturally construct a business model for a scenario by applying our patterns. Figure 13 combines Figures 7, 8, 9, 10, 11, and 12 to show the complete model of Cisco's QTC process. Note that commitments C_1 through C_{17} fully specify the model. Figure 13 shows the state of each commitment in a possible progression of this model.

4 Verification

We seek to build tools to verify if an operational model correctly supports a business model. Leading methods for verification include model checking, theorem proving, and manual testing. Theorem proving in general cannot be fully automated, and exhaustive test coverage with the manual testing is infeasible. Model checking provides a happy middle since it automatically and exhaustively verifies if a model satisfies a stated property.

Figure 14 shows the main components of our verification approach. We map a business model to a temporal logic specification regarding the progression of the states of the relevant commitments. As explained above, we capture a business model as an aggregation of business patterns. We can map each pattern to a CTL specification,

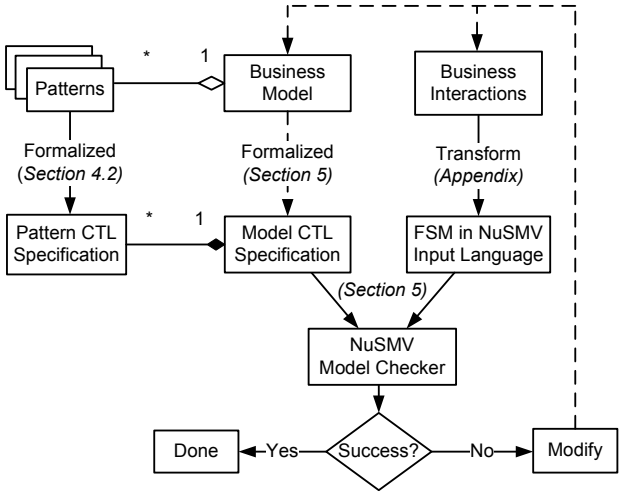


Fig. 14. Our approach in conceptual terms.

and can compose a CTL specification for a business model based on the specifications for the patterns that the model aggregates. UML 2.0 sequence diagrams capture operational interactions, which we map to FSMs specified in the NuSMV input language (as described in the online appendix of this paper).

4.1 Technical Background

We now review temporal logic and model checking concepts [14] that are required for our technical approach.

Computation Tree Logic (CTL)

CTL is a temporal logic that conceptualizes time as having a tree-like structure branching into the future. The Backus-Naur Form for CTL formulae is:

$\phi ::= \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid \mathsf{AX}\phi \mid \mathsf{EX}\phi \mid \mathsf{AF}\phi \mid \mathsf{EF}\phi \mid \mathsf{AG}\phi \mid \mathsf{EG}\phi \mid \mathsf{A}[\phi \mathsf{U} \phi] \mid \mathsf{E}[\phi \mathsf{U} \phi]$

where p is an atomic proposition. The logical operators $\neg, \wedge, \vee,$ and $\rightarrow$ have the usual meanings of negation, conjunction, disjunction, and implication, respectively. The symbols $\bot$ and $\top$ mean false and true, respectively. Each temporal operator of CTL is a pair of symbols: a path quantifier and a linear-time operator. The path quantifier can be either A, meaning along all paths, or E, meaning along at least one path. The linear-time operator can be X, meaning in the next state, G, meaning in all future states, F, meaning in a future state, and U, meaning until.

Thus, $\mathsf{AX}p$ means on all paths, p holds in the next state, $\mathsf{EG}p$ means there exists a path on which p holds in all states, and $\mathsf{AG}(x \rightarrow \mathsf{AF}y)$ means that on all paths if x holds in state s, then on all paths, emanating from s, y holds in some future state.

New Symbolic Model Checker (NuSMV)

Temporal logic model checking is a family of techniques that automatically verify if a model characterizing all possible executions of a system satisfies a given specification. NuSMV is a well-known tool for model checking CTL specifications with respect to a temporal model expressed as a finite state machine. The rest of this section introduces the syntax of the NuSMV input language that our approach uses.

The VAR keyword declares a state variable of the finite state machine. We mainly use Boolean variables (0 (false) or 1 (true)). For example,

```
VAR reqQuote: boolean;
    response: boolean;
```

declares two Boolean variables, reqQuote and response.

The DEFINE keyword is a macro that declares a symbol for an expression. For example,

```
DEFINE condition := reqQuote & response;
```

means that condition expands to reqQuote & response.

The CONSTANTS keyword declares symbolic constants. For example,

```
CONSTANTS NULL, CONDITIONAL;
```

defines NULL and CONDITIONAL as symbolic constants that other NuSMV statements can use.

A MODULE declaration in NuSMV is a parameterized construct that contains multiple declarations, constraints, and specifications. A VAR declaration instantiates a module with specified arguments. For example,

```
MODULE commitment(create, ant, con,
    ant_t, con_t)
```

defines a module with five Boolean parameters. Here, `create` activates the commitment, and `ant`, `con`, `ant_t`, and `con_t` respectively denote its antecedent, consequent, antecedent timeout, a consequent timeout. And,

```
MODULE main
  VAR C1: commitment(quoteResponse, pay,
      goods, 0, 0);
```

asserts that variable C1 in the main module instantiates the commitment module. That is, C1 is a commitment

An ASSIGN constraint assigns initial values to the state variables. For example:

```
ASSIGN init(reqQuote) := 0;
```

A TRANS constraint specifies the transition relation of the operational model in terms of the current and next state values of the state variables. For example,

```
TRANS reqQuote & !response & next(response)
= 1 & next(reqQuote) = reqQuote;
```

defines a transition from a state in which reqQuote holds and response does not hold to a state in which both reqQuote and response hold.

The CTLSPEC keyword defines a CTL specification, which NuSMV would check. For example:

```
CTLSPEC AG(order-> AF ship);
```

In our approach, NuSMV verifies if an operational model FSM satisfies the business model CTL specification. If the FSM fails to satisfy the CTL specification, NuSMV generates a counterexample, which shows an execution that violates the specification. In that event, the modeler may choose to modify either or both of the business and the operational model.

4.2 Patterns Formalized

We formalize a business model pattern as a set of CTL specifications. These specifications capture the essence of the pattern in terms of how the commitments progress during the business execution. The subsections below formalize each pattern from Section 2.1.

4.2.1 Conditional Offer Pattern Formalization

In a business execution, each commitment must progress according to the life cycle that Figure 2 shows. For each of the seven states in the commitment life cycle, we develop a CTL formula that specifies all legal transitions from that state. These exhaustively cover all possible transitions from within the commitment life cycle.

1) If a commitment is null in the current state, then in the next state it may be null, conditional, or detached.

$$\mathsf{AG}(\text{NULL} \rightarrow \mathsf{AX}(\text{NULL} \vee \text{CONDITIONAL} \vee \text{DETACHED}))$$

2) If a commitment is conditional in the current state, then in the next state it may be conditional, expired, satisfied, detached, or terminated.
   AG(CONDITIONAL → AX(CONDITIONAL ∨ EXPIRED ∨ SATISFIED ∨DETACHED ∨ TERMINATED)
3) If a commitment is detached in the current state, then in the next state it may be detached, satisfied, violated, or terminated.
   AG(DETACHED → AX(DETACHED ∨ SATISFIED ∨ VIOLATED ∨TERMINATED))
4) If a commitment is pending in the current state, then in the next state it may be pending, conditional, or detached.
   AG(PENDING → AX(PENDING ∨ CONDITIONAL ∨ DETACHED))
5) A satisfied commitment remains satisfied.
   AG(SATISFIED → AX(SATISFIED))
6) An expired commitment remains expired.
   AG(EXPIRED → AX(EXPIRED))
7) A violated commitment remains violated.
   AG(VIOLATED → AX(VIOLATED))
8) A terminated commitment remains terminated.
   AG(TERMINATED → AX(TERMINATED))

Further, for business compliance, unless the creditor releases the debtor, the debtor must satisfy a detached commitment.

9) AG(DETACHED → AF (SATISFIED ∨ TERMINATED))

Listing 1 shows a fragment of the NuSMV module for this pattern.

Listing 1. NuSMV module for the offer pattern.

```
1  MODULE commitment(create, antecedent,
2  consequent, suspend, reactivate, release,
3  cancel, ant_t, con_t)
4
5  CONSTANTS NULL,CONDITIONAL,EXPIRED,PENDING,
6  DETACHED,SATISFIED,VIOLATED,TERMINATED;
7
8  DEFINE
9  status :=
10 case
11 !create & !antecedent & !consequent &
12 !suspend & !reactivate & !ant_t &
13 !release & !cancel & !con_t:NULL;
14
15 create & !antecedent & !consequent &
16 !release & ((!suspend & !reactivate)
17 |(suspend & reactivate)) & !ant_t &
18 !cancel & !con_t:CONDITIONAL;
19
20 create & !antecedent & !consequent &
21 !release & ((!suspend & !reactivate) |
22 (suspend & reactivate)) & ant_t &
23 !cancel & !con_t:EXPIRED;
24
25 create & antecedent & !consequent &
26 & !release & ((!suspend & !reactivate)
27 | (suspend & reactivate)) & !ant_t &
28 !cancel & !con_t:DETACHED;
29
30 create & antecedent & !consequent &
31 !release & ((!suspend & !reactivate) |
32 (suspend & reactivate)) & !ant_t &
33 (cancel | con_t):VIOLATED;
34
35 create & consequent & !release &
36 !ant_t & !con_t:SATISFIED;
37
38 create & !consequent & suspend &
39 !reactivate & !release &
40 !ant_t & !con_t:PENDING;
41
42 create & !consequent & !ant_t &
43 !con_t & ((!suspend & !reactivate)
44 | (suspend & reactivate)) & (release |
45 (!antecedent & cancel)):TERMINATED;
46
47 esac;
48
49 CTLSPEC
50  AG(status=NULL->AX(status=NULL|
51  status=CONDITIONAL|status=DETACHED));
```

We use the MODULE declaration to specify the pattern in a reusable manner. Line 1 declares the module with parameters of create, antecedent, consequent, antecedent timeout, and consequent timeout. Lines 5–6 use the CONSTANTS keyword to declare the commitment states as symbolic constants. Lines 8–47 compute the state of the commitment based on the input parameters. Lines 49–51 use the CTLSPEC keyword to declare the CTL specifications. Note that this is NuSMV encoding of the CTL specification 1 from the above list. We omit the NuSMV listings for the remaining patterns to save space.

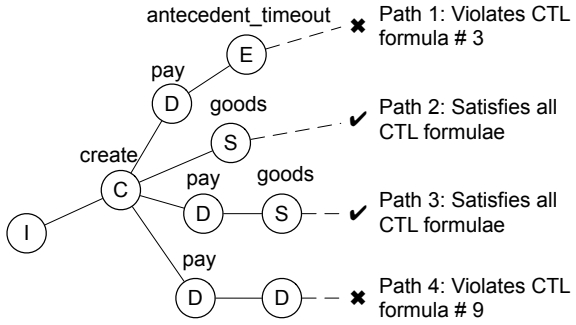


Fig. 15. Example: Verifying the conditional offer pattern.

Next, we demonstrate how verification works. Consider the commitment C(SELLER, BUYER, pay, goods): a seller commits to a buyer to shipping the goods if the buyer pays. Figure 15 shows possible executions of an operational model. A circle depicts a state, and its label shows the state of the commitment. Above each state, the figure shows the action or event that holds in that state. Further, it labels the paths that satisfy or violate the conditional offer CTL specifications above. For example, Path 1 violates Specification 3 from above since the commitment transitions from the detached to the expired state.

4.2.2 Commercial Transaction Pattern Formalization

The commercial transaction pattern consists of a pair of reciprocal commitments as Figure 4 shows. The CTL specifications of the conditional offer pattern individually apply to each of the commitment.

In this pattern, if one commitment is satisfied, it detaches the other commitment, and vice versa. The CTL specifications that specify this part of the pattern are:

1) $\mathsf{AG}(\text{DETACHED}_1 \leftrightarrow \text{SATISFIED}_2)$
2) $\mathsf{AG}(\text{DETACHED}_2 \leftrightarrow \text{SATISFIED}_1)$

4.2.3 Outsourcing Pattern Formalization

Figure 5 shows the four commitments that are part of the outsourcing pattern. Among these, C_3 and C_4 are the reciprocal commitments between the outsourcer and the contractor. The CTL specifications of the commercial transaction pattern apply to these commitments, and the CTL specifications of the conditional offer pattern individually apply to commitments C_1 and C_2.

The CTL specifications unique to this pattern are:

1) If the outsourcer updates the commitment C_1 to be pending, then on all paths the contractor must eventually create the outsourced commitment C_2.
   $\mathsf{AG}(\text{PENDING}_1 \rightarrow \mathsf{AF}(\neg\text{NULL}_2))$
2) If the contractor satisfies C_2, then it satisfies the outsourcer's commitment C_1.
   $\mathsf{AG}(\text{SATISFIED}_2 \rightarrow \text{SATISFIED}_1)$
3) If the contractor violates the commitment C_2, then the outsourcer's commitment C_1 is reactivated.
   $\mathsf{AG}(\text{VIOLATED}_2 \rightarrow \mathsf{AF}(\neg\text{PENDING}_1))$

4.2.4 Service Contract Pattern Formalization

Figure 6 shows the service contract pattern consisting of three commitments. Of these, the CTL specifications of the commercial transaction pattern apply to C_1 and C_2, which are reciprocal commitments between the provider and the consumer.

The standing service commitment C_3 applies to multiple service request instances. If the consumer sends a service request prior to its expiration, and if the request count is smaller than the upper bound, then the provider creates a new detached commitment to service that request. The provider creates a new commitment for each service request. The CTL specifications of the offer commitment pattern apply to each of these commitments.

5 Evaluation: Applying Verification

This section applies the verification approach to the QTC business process introduced in Section 3. Figures 16, 22, and 23 (from the online appendix) show the sequence diagrams that model the operational interactions between the QTC participants. The participant roles shown as lifelines on these diagrams map to the model roles from Figure 13.

Figure 16(a) captures the quoting interactions. The customer requests a quote from the reseller to purchase and install the desired goods. The reseller requests a quote from the distributor for the goods. Upon receiving a quote from the distributor, the reseller sends a quote to the customer. The customer responds, and either accepts the quote or requests a new quote with an additional discount. The reseller responds similarly to the distributor. The quote and the response interactions continue until either the customer accepts the quote, or the iteration count exceeds five. We notate the meaning of the messages below the diagram as message $\rightarrow$ create(C), where C is the commitment that the message creates. The customer's acceptance of the reseller's quote creates C_2 and C_3. And the reseller's acceptance of the distributor's quote creates C_5. These commitments are defined in the QTC business model of Figure 13.

In Figure 16(b), the customer sends an order to the reseller, and the reseller in turn sends an order to the distributor. This message sequence is guarded by the customer's quote accept message that appears in Figure 16(a). The customer's purchase order sent to the reseller creates C_1, the reseller's purchase order sent to the distributor creates C_4, and the distributor's confirmation sent to the customer creates C_6.

Figure 16(c) shows the quoting interaction between the distributor and the shipper. The quote sent by the shipper to the distributor creates C_7, the quote acceptance sent by the distributor to the shipper creates C_8, and the confirm ship request message sent by the shipper to the customer creates C_9.

In Figure 16(d), the shipper ships the goods to the customer. This sequence is guarded by the shipper's confirm ship request message to the customer. By shipping the goods to the customer, the shipper satisfies C_9, which in turn satisfies the pending commitments C_6, and C_3. For brevity, we use the name of each task (e.g., shipGoodsE) in the business model as the name of the message (e.g., *shipGoodsE*) that completes the task. This is not necessary in our approach. It would be a simple matter to use different names but we would have to map each message to the corresponding task.

Similarly, Figures 22 and 23 (in the online appendix) show the remaining sequence diagrams for QTC. Figures 22(e), (f), (g), and (h) show product quoting, ship quoting, shipping, and service quoting, respectively. Figure 23(i) shows the customer's request for service, and Figures 23(j) and (k) show various payment and install interactions. These diagrams show the commitments that the messages create. In some sequence diagrams, commitments are satisfied as their consequent is brought about.

The patterns that constitute the QTC business model of Figure 13 yield a total of 200 CTL specifications. The QTC sequence diagrams of Figures 16, 22, and 23 yield an FSM that has 12,600 states and 12,600 transitions. On a computer with 2.66 GHz Intel Core 2 Duo processor, and 4 GB memory, NuSMV verifies the QTC model in 0.1 seconds. The sequence diagrams from this section satisfy the QTC business model from Figure 13. Figure 17

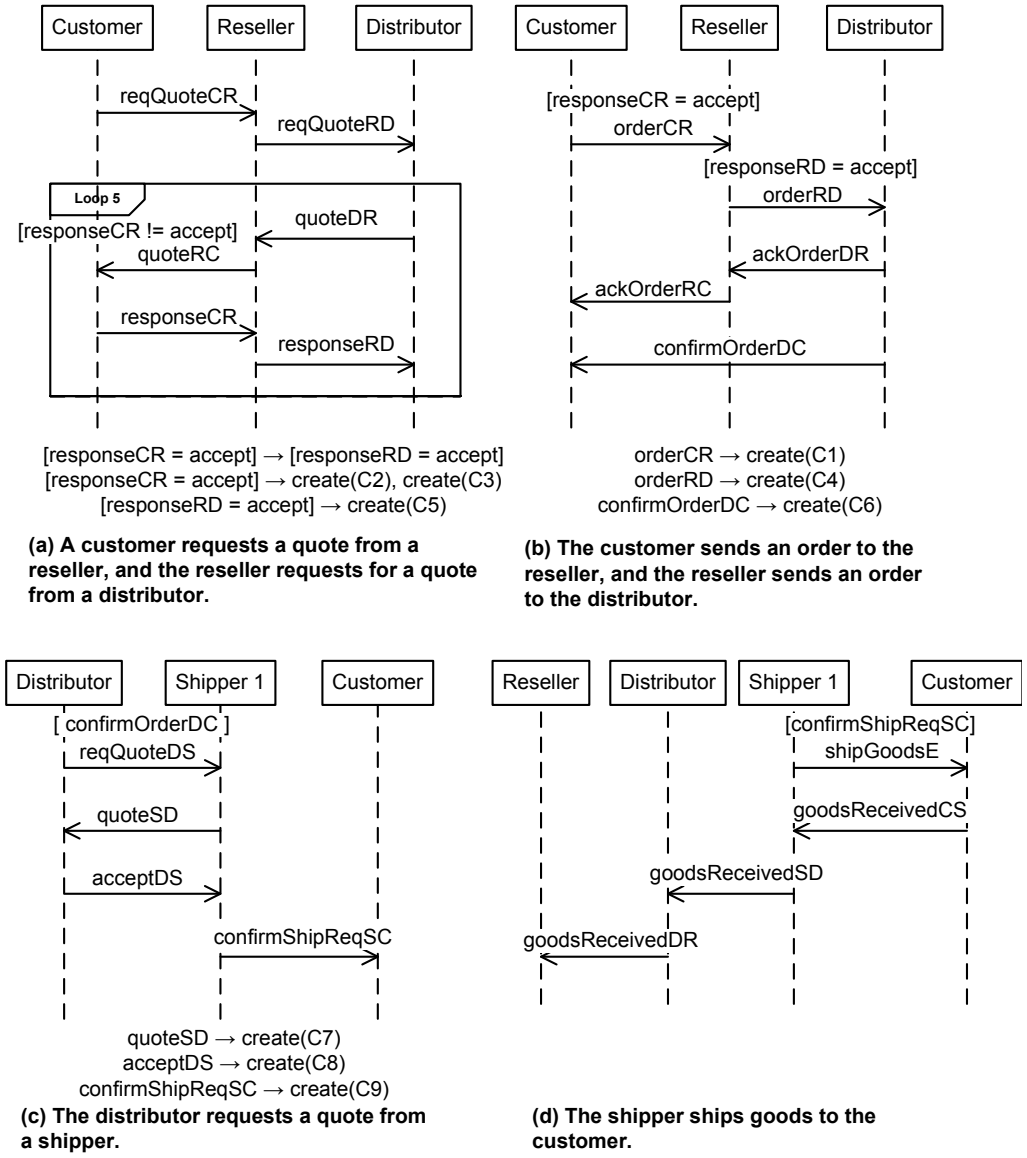


Fig. 16. Operational interactions in a QTC implementation.

shows a partial screen shot of NuSMV output. It shows several specifications that the model satisfies. For example, the highlighted specification is NuSMV equivalent to a specification from the unilateral commitment pattern: $\mathsf{AG} = (\text{NULL} \rightarrow \mathsf{AX}(\text{NULL} \vee \text{CONDITIONAL} \vee \text{DETACHED}))$. The output shows that the model satisfies this specification for commitment C_1. We now present a few variations of these sequence diagrams that fail to satisfy the specification.

Potential Violations

Because the above model is verified successfully, we consider some (imaginary) potential violation scenarios to further demonstrate our approach.

Scenario 1: Failure to confirm the order

If we remove the confirmOrderDC message from the sequence diagram of Figure 16(b), the distributor fails to confirm the order, thus violating these specifications:

1) $\mathsf{AG}(\text{DETACHED} \rightarrow \mathsf{AF}(\text{SATISFIED}))$ for C_5: The NuSMV counterexample shows an execution in which C_5 detaches since the reseller pays the distributor. However, C_5 never satisfies since the distributor never confirms the order, that is, never creates C_6. Therefore, the distributor violates C_5. At a business level, this situation is undesirable since the reseller pays the distributor to create a commitment to ship the goods to the customer, but the distributor fails to create that commitment.
2) $\mathsf{AG}(\text{VIOLATED}_5 \rightarrow \mathsf{AF}(\neg \text{PENDING}_3))$: In the operational model, the (original) commitment C_3 becomes pending when reseller accepts the distributor's quote (responseRD = accept). By not sending the confirmOrderDC message, the distributor violates the (outsourced) commitment C_5. But C_3 stays pending, and it is not reactivated. At a business

level, this is undesirable since the reseller who is responsible for satisfying C_3 fails to reactivate it after the outsourced commitment is violated.

Scenario 2: Failure to ship the goods

Here we remove the shipGoodsE message from the sequence diagram of Figure 16(d), that is, the shipper fails to ship the goods to the customer. Figure 18 shows a partial screen shot of the NuSMV output, highlighting that the model violates the specification $\mathsf{AG}(\text{DETACHED} \rightarrow \mathsf{AF}(\text{SATISFIED}))$ for commitment C_6. The model also violates the same specification for commitment C_9. In the counterexample produced by NuSMV, the commitments are detached, but are never satisfied since their consequent shipGoodsE is not brought about. Note that the specification $\mathsf{AG}(\text{DETACHED} \rightarrow \mathsf{AF}(\text{SATISFIED}))$ is not violated for C_3 although it has the consequent shipGoodsE. This is because, as Figure 23(j) shows, the customer does not pay the reseller until the goods are received. Since shipGoodsE is removed, the customer never receives the goods and, therefore, C_3 never detaches.

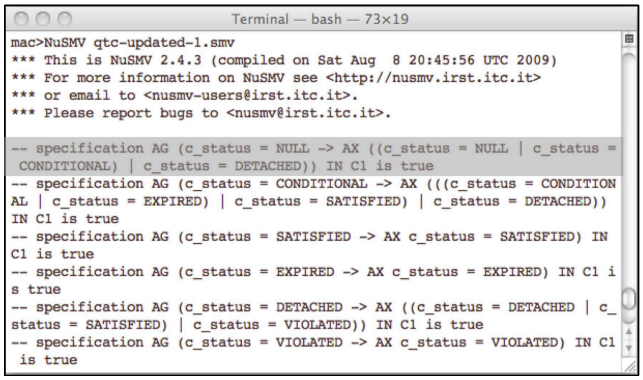


Fig. 17. Screen shot of NuSMV output indicating satisfaction.

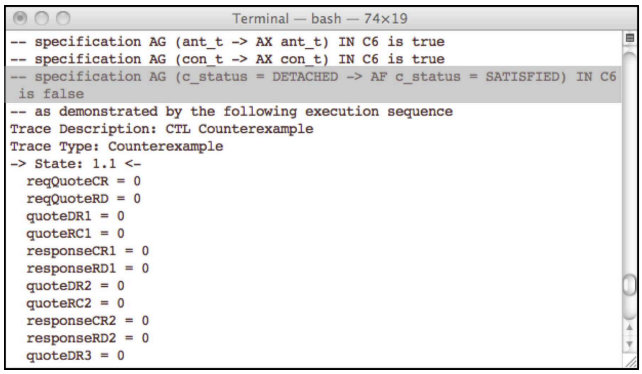


Fig. 18. NuSMV output showing violation: failure to ship goods.

Runtime Violations

The above examples demonstrate how our approach detects commitment violations in an operational model at design time. However, a participant may violate a commitment at runtime. To handle such cases, the business model may employ the penalty pattern [11]. For example, if the reseller ships the goods but the customer fails to pay $10 within 15 days, that is, the customer violates a commitment. Then a penalty applies, committing the customer to pay $15 within 30 days to the reseller.

6 Related Work

We relate our work to three research areas: business process modeling, software engineering methodologies for open systems, and formal approaches of modeling and verifying agent interactions.

Business Process Modeling

Hofreiter et al. [15] present UN/CEFACT's methodology UMM for modeling interorganizational business processes as global choreographies. Similar to our approach, UMM specifies a choreography at a business level, independently of the underlying implementation technology. However, unlike UN/CEFACT's UMM model, our metamodel naturally captures business aspects by giving primacy to the commitments among the participants. Further, the well-defined semantics of a commitment and its operations provide a basis for formally verifying operational executions.

Bodenstaff et al. [16] present a methodology to check consistency between different models of an interorganizational scenario. Their methodology is model-independent, but their notion of consistency fails to consider the temporal progression of a model. In contrast, our method verifies the temporal progression of an operational model, with respect to the business model.

Rosenberg et al. [17] highlight the lack of a modeling method that derives a cross-organizational business process starting from service level agreements (SLA) among the participants. They propose a methodology that extends the web services choreography description language (WS-CDL) to capture SLAs specified in the web service level agreement (WSLA) language. Unlike our metamodel, this approach fails to capture the agreements (relationships) among the participants at a business level, and lacks the flexibility that our metamodel offers.

Milosevic et al. [18] propose a method that derives a cross-organizational business process starting from a contract between the participants. This is similar in spirit to our approach, which starts from a business model to derive an operational model. However, Milosevic et al.'s approach models a contract using the deontic notion of an obligation. Although obligations are similar to commitments, they are limited since they are not directed from one participant to another, and cannot be readily manipulated. Further, unlike our approach, Milosevic et al.'s method lacks a formal approach to verify if the derived business process satisfies the original contract.

Engineering Methodology

Gordijn and Wieringa [19] propose the e^3-value approach, which captures a business organization as an actor, similar to our notion of an agent. Actors execute value activities, similar to our tasks. In e^3, a value interface aggregates related in and out value ports of an actor to represent economic reciprocity. This concept is close to our concept of commitment, but it lacks formal semantics and doesn't yield flexibility. For example, unlike value interfaces, commitments can be readily delegated. Due to this, during execution, the exchange and interaction may take place among actors different from those included in an e^3 model.

Our approach relates to Tropos [20] in spirit, but differs significantly in that we use commitments. The debtor, creditor, and consequent of a commitment are respectively similar to the dependee, depender, and dependum of a Tropos dependency [21]. However, unlike a dependency, a commitment has an antecedent that brings it into full force. This enables modeling of reciprocal relationships between economic entities, which is lacking in the concept of dependency.

El Menshawy et al. [22] extend our approach from [9] with argumentation to develop a methodology to model communities of web services. Unlike El Menshawy et al.'s methodology, our approach is founded upon patterns. We formalize the patterns, and show how business executions are verified with respect to a business model.

Amoeba [1] is a process modeling methodology based on commitment protocols. This methodology creates a model in terms of fine-grained messages and commitments. Our approach lies at a higher level of abstraction containing business goals, tasks, and commitments.

Opera is a framework for modeling multiagent societies [23], though from the perspective of a single designer or economic entity. In contrast, we model interactions among multiple entities. Opera's concepts of landmark, scene, and contract are close to our concepts of task, protocol, and commitment, respectively. However, Opera uses traditional obligations, which lack the flexibility of commitments, as explained above.

Formal Approaches

Fornara et al. [6] model an open interaction system in terms of social commitments, events, agents, roles, and norms. They specify a model using an ontology in OWL, and use SWRL and a custom Java program to monitor the temporal progression of social commitments. We specify the business model using CTL, and model check operational interactions specified as UML sequence diagrams. Unlike the patterns in our approach that model a business scenario, Fornara et al.'s approach lacks principled means of creating a model.

Winikoff [5] argues that a message-centric agent interaction model limits flexibility and autonomy of the agents. He proposes an approach using commitments for modeling agent interactions. We indeed share the same motivation, and base our business model on commitments. Winikoff further outlines a multistep process for designing an interaction model, which he applies on a simple meeting scheduler application. However, unlike our approach, his process does not identify reusable patterns. Further, we evaluate our approach on a real-world Quote To Cash business process scenario.

The Logic for Contract Representation (LCR) [24] specifies interactions in Opera [23]. Avali and Huhns [25] share our notion of commitments. They apply BDI_{CTL*} to relate commitments to an agent's beliefs, desires, and intentions. LCR and Avali and Huhns' approach support reasoning about obligations and commitments, respectively. Such reasoning would complement our work. By contrast, our approach focuses on interactions among real-world organizations and enables us to develop high-level business models, and to verify those with respect to low-level operational models. Further, we develop a set of real-world business patterns.

Fuxman et al. [26] present an approach to check a Tropos specification for consistency and other properties that capture stakeholder expectations. Our approach checks an operational model with respect to a business model, wherein the semantics of a commitment and its operations yield the desired temporal specifications.

Pijpers and Gordijn [27] present a semiformal approach for checking the consistency of a process model (UML activity diagram) with respect to an e^3-value model. In their approach, a process model needs to contain all value exchanges that an e^3-value model specifies. For consistency, our approach does not mandate an execution to create and satisfy all of the model commitments. Instead, it only requires detached commitments to be satisfied prior to the detached timeout. This offers flexibility that real-world processes demand.

7 Conclusions

The main contributions of this paper are a business metamodel, a set of modeling patterns, and an approach for formalizing business models and verifying operations with respect to models. Using a real-life business scenario, we evaluate both our model and patterns (Section 3), and our verification approach (Section 5). Our set of business model patterns is clearly not exhaustive nor do we expect any set of patterns to be exhaustive—hundreds of patterns exist for programming and for software architecture, and the domain of business models appears no less complex than those domains. However, our core set of patterns shows how we may construct additional patterns. Our approach helps a business modeler concentrate on high-level commitments, and helps detect flaws in business process implementations. This paper establishes the practical usability of the approach by applying it to a real-world business process.

In future work, we hope to study business models from different domains to identify additional patterns. Along these lines, we have begun abstracting high-level business patterns [28] from the so-called Partner

Interface Processes of RosettaNet, a leading industry standard for business-to-business interactions [29]. We will extend the approach to check if business executions satisfy model goals, and develop properties that a model should satisfy for consistency. We also expect to develop a methodology for business modeling, and graphical tools for creating business models.

Acknowledgments

Our business metamodel and patterns (but not their formalization or verification) appeared in our previous work [9], as did parts of our second case study in the appendix. We thank the anonymous reviewers for helpful comments and Scott Gerard and Amit Chopra for useful discussions.

References

[1] N. Desai, A. K. Chopra, and M. P. Singh, "Amoeba: A methodology for modeling and evolution of cross-organizational business processes," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 19, no. 2, pp. 6:1–6:45, Oct. 2009.

[2] M. P. Singh and M. N. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*. Chichester, UK: John Wiley & Sons, 2005.

[3] BPEL, "Web services business process execution language, version 2.0," Jul. 2007, http://docs.oasis-open.org/wsbpel/2.0/.

[4] WS-CDL, "Web services choreography description language version 1.0," Nov. 2005, www.w3.org/TR/ws-cdl-10/.

[5] M. Winikoff, "Designing commitment-based agent interactions," in *Proc. IEEE/WIC/ACM Int'l Conf. Intelligent Agent Technology*, 2006, pp. 363–370.

[6] N. Fornara and M. Colombetti, "Ontology and time evolution of obligations and prohibitions using semantic web technology," in *Proc. 7th AAMAS Workshop Declarative Agent Languages and Technologies*, 2009, pp. 101–118.

[7] M. P. Singh, "An ontology for commitments in multiagent systems," *Artificial Intelligence and Law*, vol. 7, pp. 97–113, 1999.

[8] *UML 2.0 Superstructure Specification*, Object Management Group, Framingham, Massachusetts, Oct. 2004.

[9] P. R. Telang and M. P. Singh, "Business modeling via commitments," in *Proc. 7th AAMAS Workshop Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE)*, ser. LNCS, vol. 5907. Springer, 2009.

[10] M. B. van Riemsdijk, M. Dastani, and M. Winikoff, "Goals in agent systems: a unifying framework," in *Proc. 7th Int'l Conf. Autonomous Agents and Multiagent Systems (AAMAS)*, 2008, pp. 713–720.

[11] M. P. Singh, A. K. Chopra, and N. Desai, "Commitment-based service-oriented architecture," *IEEE Computer*, vol. 42, no. 11, pp. 72–79, Nov. 2009.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, ser. Professional Computing Series. Reading, MA: Addison-Wesley, 1995.

[13] P. Yolum and M. P. Singh, "Enacting protocols by commitment concession," in *Proc. 6th Int'l Conf. Autonomous Agents and Multi-Agent Systems*, May 2007, pp. 116–123.

[14] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, Massachusetts: The MIT Press, 1999.

[15] B. Hofreiter, C. Huemer, P. Liegl, R. Schuster, and M. Zapletal, "UN/CEFACT's Modeling Methodology (UMM): A UML profile for B2B e-commerce," in *2nd Int'l Workshop Best Practices of UML (ER)*, 2006, pp. 19–31.

[16] L. Bodenstaff, A. Wombacher, M. Reichert, and R. Wieringa, "MaDe4IC: An abstract method for managing model dependencies in inter-organizational cooperations," *Service Oriented Computing and Applications*, vol. 4, pp. 203–228, 2010.

[17] F. Rosenberg, A. Michlmayr, and S. Dustdar, "Top-down business process development and execution using quality of service aspects," *Enterprise Information Systems*, vol. 2, pp. 459–475, 2008.

[18] Z. Milosevic, S. Sadiq, and M. Orlowska, "Translating business contract into compliant business processes," in *Proc. 10th IEEE Int'l Enterprise Distributed Object Computing Conf. (EDOC)*, 2006, pp. 211–220.

[19] J. Gordijn and R. Wieringa, "A value-oriented approach to E-business process design," in *Proc. 15th Int'l Conf. Advanced Information Systems Engineering*, ser. LNCS, vol. 2681. Springer, 2003, pp. 390–403.

[20] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[21] P. R. Telang and M. P. Singh, "Enhancing Tropos with commitments," in *Conceptual Modeling: Foundations and Applications*, ser. LNCS, vol. 5600. Springer, 2009, pp. 417–435.

[22] M. El-Menshawy, J. Bentahar, and R. Dssouli, "Enhancing engineering methodology for communities of web services," in *Second Multi-Agent Logics, Languages, and Organisations Federated Workshop*, vol. 494, 2009.

[23] H. Weigand, V. Dignum, J.-J. C. Meyer, and F. Dignum, "Specification by refinement and agreement: Designing agent interaction using landmarks and contracts," in *Engineering Societies in the Agents World III*, ser. LNCS, vol. 2577. 2002, pp. 257–269.

[24] V. Dignum, J.-J. C. Meyer, F. Dignum, and H. Weigand, "Formal specification of interaction in agent societies," in *Formal Approaches to Agent-Based Systems*, ser. LNCS, vol. 2699. Springer, 2002, pp. 37–52.

[25] V. R. Avali and M. N. Huhns, "Commitment-based multiagent decision making," in *Proc. 12th Int'l Workshop Cooperative Information Agents*, ser. LNCS, vol. 5180. Springer, 2008, pp. 249–263.

[26] A. Fuxman, J. Mylopoulos, M. Pistore, and P. Traverso, "Model checking early requirements specifications in Tropos," in *Proc. 5th IEEE Int'l Symp. Requirements Engineering (RE)*. 2001, pp. 174–181.

[27] V. Pijpers and J. Gordijn, "Consistency checking between value models and process models: A best-of-breed approach," in *Proc. 3rd Int'l Workshop Business/IT Alignment and Interoperability (BUSITAL)*, 2008, pp. 58–72.

[28] P. R. Telang and M. P. Singh, "Abstracting and applying business modeling patterns from RosettaNet," in *Proc. 8th Int'l Conf. Service-Oriented Computing*, 2010, pp. 426–440.

[29] RosettaNet, Overview: Clusters, segments, and PIPs, 2008, www.rosettanet.org.

[30] S. Browne and M. Kellett, "Insurance (motor damage claims) scenario," CrossFlow Consortium, Document Identifier D1.a, 1999.

Pankaj R. Telang is an IT Architect at Cisco Systems, and a PhD student in the Department of Computer Science, North Carolina State University, Raleigh. His research interests include business modeling, service oriented architecture, and agent oriented software engineering.

Munindar P. Singh is a Professor in the Department of Computer Science, North Carolina State University, Raleigh. His research interests include multiagent systems and service-oriented computing with a special interest in the challenges of trust, service selection, and business processes in large-scale open environments. Dr. Singh was the Editor-in-Chief of the *IEEE Internet Computing* from 1999 to 2002. He is a member of the editorial boards of *IEEE Internet Computing*, *Autonomous Agents and Multiagent Systems*, *Web Semantics*, and *Service-Oriented Computing and Applications*.