Guarantees: The Basis for Electronic Commerce [1]

Cris Pedregal-Martin[2] and Krithi Ramamritham[3]

Technical Report 00-61

Computer Science Department

University of Massachusetts, Amherst, Mass. 01003-4610, USA

Abstract

It has been recognized that for Electronic Commerce Transactions (ECTs) to work, the entities involved follow specified *protocols*. In this paper, we argue that perhaps even more important than the protocols are the *guarantees* that entities involved in an ECT give each other while they execute their actions in conformance with these protocols. The protocols and guarantees together lead to the fulfillment of high level system-wide properties that an ECT is supposed to possess, both general (e.g., correct exchange of goods for money) and specific (e.g., highest bidder gets the item).

Here we consider two Electronic Commerce scenarios and show how guarantees and protocols can be used both to prove desirable properties for those scenarios and as a specification of the properties that parties to an ECT must satisfy.

1 Introduction

It has been well recognized that for Electronic Commerce Transactions (ECTs) to work, entities involvedin an ECT must follow specified *protocols*. In this paper, we argue that equally important, if not more important, than the protocols are the *guarantees* that entities involved in an ECT give each other while they execute their actions in conformance with these protocols. The protocols and guarantees together lead to the fulfillment of high level system-wide properties that an EC system is supposed to possess. We are interested in the properties that the system as a whole exhibits, such as the invariant of exchanging goods for money, and in the properties which each party in the system either expects or upholds, e.g., a seller expects to get paid for goods shipped, regardless of the implementation details.

To help in the precise description of the protocols, the guarantees, and the high-level properties, we develop a simple formal framework that allow designers to specify and reason about electronic commerce

[1] Research funded by the National Science Foundation under grant IRI-9619588.

[2] Email: cris@cs.umass.edu, vox: +1 413 545-4753, fax: +1 413 545-1249 (Contact Author).

[3] Email: krithi@cs.umass.edu, vox: +1 413 545-0196, fax: +1 413 545-1249.

transactions and the entities that support the enactment of the transactions. We illustrate the capabilities of our framework by applying it to a detailed scenario involving electronic auctions on the web: specificaly, we state the protocols that the bidders, the sellers, and the arbiter must follow and also the guarantees that they depend on each other for. These specifications are then used to demonstate the necessary correctness properties. We also show the step-wise refinement capabilities of our approach by studying the details of credit card authorizations in an electronic retail scenario. These examples illustrate the generality of scope and the analysis capablities of our approach.

In electronic commerce scenarios, where the multiple parties are autonomous and enter into contracts with each other typically in a pairwise manner, the high-level system-wide properties of necessity must emerge as a consequence of the per-party, and pair-wise, properties achieved and maintained by the various parties. Because of their mutual autonomy, one component (party[4]) must state and understand expectations and promises (our guarantees) and their behavior (our protocols) in abstract terms geared toward the needs of the commerce application in question rather than in terms of the specific technology used to implement it.

The framework we use in our work reflects these concerns: it is abstract enough to afford descriptions independent of the technology, yet detailed enough to capture the essential properties of the Electronic Commerce scenarios under study.

Our work sheds light on the building blocks of electronic commerce transactions and the essentials of electronic commerce, as opposed to descriptions wedded to specific technologies. In this sense, the contributions of this paper form a step in the direction of answering the need to systematize the building blocks of electronic commerce transactions so that their properties become more transparent and their construction made more methodical.

1.1 Protocols, Guarantees, and High-level Properties

We postulate that ECTs work not only because of the *protocols* that the entities involved in an ECT agree to follow but, perhaps more importantly, because of the *guarantees* they give each other while they execute their actions in conformance with these protocols.

- Protocols specify correct sequences of operations, and embody the semantics of the applications. For

[4] Throughout the paper we indistinctly refer to an Electronic Commerce *scenario* or *system*, and to the entities involved as its *parties*, or *components*; this reflects the dual usage of commerce and information systems.

example, a protocol specification states the requirement that a merchant should arrange to ship an item to a customer only if a customer has ordered that item. Thus protocols preclude incorrect behaviors on the part of the system.

- Guarantees express the commitment an entity makes to another that the former will fulfill a specified request from the latter in the future, in spite of failures or disruptions. For example, a credit card authorization given by a bank to a merchant, is (tantamount to) a guarantee given by the bank to the merchant; the merchant can later use the guarantee to obtain the money that was authorized when the guarantee was given. Thus guarantees ensure to their requestors that they can count on the guarantor to perform some action at their request. Guarantees are abstractions that make explicit and formalize an aspect of a system – the expectation of future behavior – that is usually described informally or, at worst, silently assumed.

- High level properties of a system are achieved by following the protocols and delivering on the guarantees. A typical example of a high-level property maintained in an EC scenario is that the correct amount of money is exchanged for the correct amount and type of goods.

Protocols are enough to ensure that the executions are not erroneous, but not enough to ensure that given the correct conditions, executions *will* succeed. The latter is the role of guarantees. In part, the role of guarantees is to yield aggregate high-level properties for systems composed of *autonomous* entities, over which no central authority can enforce a discipline, but instead the results must be based on party-to-party expectations (the guarantees).

This paper considers two examples of electronic commerce transactions in light of the commitments (guarantees) between the different parties in an EC transaction and the steps that they execute according to rules (protocols) and how those guarantees and protocols can be used both to prove desirable properties in specific EC scenarios (e.g., the exchange of goods for money), and as a specification of the properties that parties to an EC transaction must satisfy.

1.2 Overview of two Electronic Commerce Scenarios

We consider two scenarios related to each other. The first one is a web auction EC scenario, in which items are sold to the highest bidder in a conventional (English) auction [KF98a, KF98b, LR99]. The second one is a version of a web merchant, wherein retail customers order goods over the Internet for which they pay with credit cards. In this overview we present them as distinct, but we later specialize the retailer scenario to account for the details of the auction concerning the actual transaction between a seller and a buyer.

1. The auction example consists of a Seller, many Buyers, and an Arbiter. In order to bid, a Buyer must offer a guarantee: if it wins the auction, it will buy the item being auctioned. The Buyer offers this guarantee to the Arbiter, who guarantees to the Buyer that it will not bid over the limit specified by the Buyer. In order to offer an item for Auction, the Seller guarantees that it will sell the item for the bid identified as winner by the Arbiter. The Arbiter issues guarantees to the Seller and the Buyers which embody a fair auction: which bids will be accepted, how the winning bid will be chosen, etc. We describe the auction in terms of those parties to it, and we use the Arbiter for the web auction component, because typically that component plays several roles, only one of which is that of auctioneer. A Buyer is usually represented at the Arbiter's systems by a Proxy (who bids on behalf of the buyer, according to predetermined rules and limits); Proxies in turn interact with an Auctioneer, and the Auctioneer with a Seller. For our purposes we consider a Buyer and its proxy as one.

2. The retail example consists of a Merchant, a Customer, a Bank, and a Shipper: the Customer buys goods from the Merchant and offers a credit card to pay for them; the Merchant confirms the sale to the Customer on obtaining a suitable authorization on the Customer's credit card from the Bank, and hands over the goods to the Shipper, who delivers them to the Customer. A sale may not succeed because an item requested is out of stock, or because the Merchant is unable to secure an authorization, or because the Customer backs out before confirming the purchase with a credit card.

   From the point of view of the Customer, its interaction with the Merchant ends as soon as the Merchant confirms that the order has been accepted – by that confirmation, the Merchant guarantees to the Customer that the Customer will receive the goods it ordered. In turn, the Merchant issues that guarantee after receiving a guarantee from the Bank (a credit card authorization) that the Merchant will be able to charge for the shipped goods.

The study of these two scenarios sheds light on the essentials of Electronic Commerce, rather than providing descriptions wedded to specific technologies. We discuss differences and similarities between the examples, and we also find strong analogies between EC and the makeup of traditional Database Transaction Systems.

To achieve this, we formally study these systems by precisely stating the protocols and guarantees that are involved. We also analyze how the components subsystems support or enforce individual guarantees and protocol specifications. For example, a bank may issue a charge authorization by committing a transaction into its database system, ensuring that the authorization and its effects will persist in spite of failures.

1.3 Electronic Commerce Transactions and Advanced Transaction Models

The parties in Electronic Commerce engage in activities that can be described by workflows [CCPP98] or as long-running transactions. In some sense, the specifications of these workflows or long-running transactions are our protocols. Much work has been done in the area of Advanced or Extended Transaction Models (ETMs) (for example, see [CR94, DHL90, BOGM91, GHM96, WH93]), in which some of the assumptions of traditional database transactions are relaxed to accommodate the needs of other types of applications, while preserving some of the desirable (correctness and) semantics of DB transactions.

We believe that although ETMs are useful tools to characterize ECTs, they are not sufficient. We base our argument on the observation that while ETMs specify protocols, they do not deal with guarantees.

1.4 Organization of this paper

The structure of the rest of the paper is as follows. We formally define guarantees and protocols and introduce notation in Section 2. Then we study the electronic auctions scenario in depth, characterizing the properties of each component and of the system as a whole in Section 3. Section 4 follows with a study of the electronic retail scenario, including a discussion of its relationship with the auction scenario. In both examples we state some properties and show how guarantees and protocols contribute to proving their validity. Finally, in Section 5 we place this work in the broader context of Electronic Commerce and Extended Transaction Models, we show analogies between the EC scenarios and a database transactional system, and indicate directions of future work.

2 Definitions and Notation

This section describes the ingredients of the framework we use to state the guarantees, protocols, and system properties.

High-level *system properties* specify the expectations of a system as a whole. For example, an Electronic Commerce system should require that money always be exchanged for goods.

Component-wise properties, *guarantees* and *protocols*, are useful to decompose those expectations into requirements on components (parties) of the system, and symmetrically to show that given those component-wise properties, desirable high-level system properties hold for the whole.

2.1 Predicates on Events in a History

We represent the behavior of a system by considering operations and other important *events* which change the state of the system and the world; these events form the system's *history*. The history H of the system is a partial order on all the events.

We write predicates over the events in the history using standard notation. Symbols with the usual meanings are: $=$ (identity), $\Rightarrow$ (logical implication), $\in$ (element belongs to set), $\forall, \exists$ (quantifiers for all and there exists), : (such that), $\wedge, \vee$ (logical *and* and *or*), etc. The single arrow symbol $\rightarrow$ denotes precedence: $\textsc{e}_0 \rightarrow \textsc{e}_1$ means, should both events occur in the history, event $\textsc{e}_0$ precedes event $\textsc{e}_1$ in the (implied) history H. We denote the events with their name written in small capitals, possibly with a subscript to identify different instances of the same type. For example: $\textsc{close-auction}$, $\textsc{bid}_j$, $\textsc{bid}_k$ are all events. When necessary, we identify an attribute of an event by appending its name with a dot; for example $\textsc{bid}_k.Buyer$ denotes the buyer associated with the k^{th} bid.

We describe the behavior of the system via logical predicates, which specify when and how one event is influenced by the occurrence of other events in the history. Events may be further bound to application-specific values: for example, an event signifying a purchase may have attached to it values that name the item purchased and specify the item's price. These statements are formalized as *logical predicates* on events in history and their values.

We use the combination of statements about where (or in which order) events appear in a history, and the binding of the values for each instance, to capture the behavior of a system, and the rules on that behavior.

2.2 Protocols, Guarantees, High-level Properties

We gave an informal description of protocols, guarantees, and higher-level properties in Section 1.1; here we characterize those concepts in depth, and in we show them in use in Sections 3 and 4.

Protocols. The purpose of a protocol is to dictate correct behavior, by placing conditions on the presence of some event in the system's history or on the precedence relationships between events. Thus a protocol is typically specified using logical expressions involving precedence constraints. For example:

bid-up $\forall j, k \; \textsc{bid}_j \rightarrow \textsc{bid}_k \;\; \Rightarrow \;\; \textsc{bid}_j.price < \textsc{bid}_k.price$

This says that a new bid is always higher than all the preceding bids.

Guarantees. The purpose of a guarantee is to express that, given certain conditions, some desirable effect (operation) *will eventually happen* in spite of intervening failures. This assurance is a promise given by one autonomous system component to another, and, counting on it, the beneficiary can in turn offer other promises, ultimately yielding properties for the system as a whole.

A guarantee is a predicate of a temporal nature: it states that, if a certain operation is done now, from some point in the future on, it will be possible to successfully invoke and execute a specified operation. We state guarantees in the context of understanding how the components of a system interact with each other, and especially to describe the expectations of each component. In this sense, a guarantee is a promise one component makes to another. Thus a component (the *requestor*) typically executes an operation whose success results in the granting of a guarantee by another component (the *guarantor*). The lifetime of a guarantee goes through the events of being granted, triggered, bound, and discharged, in that order, although some events may happen simultaneously.

First, the guarantee is *granted*. This by itself does not imply that the requestor will exercise the guarantee; that is, the guarantor *may or may not* have to deliver on its promise to discharge the guarantee. But, once the guarantee is *triggered*, the guarantor *will* have to deliver on its promise; however, some details about the "variables" associated with the guarantee may be still undefined. Third, the guarantee is *bound*, at which point all the variables associated with it have values assigned to them. Finally, the guarantee is *discharged*, that is, its promise will be fulfilled. (Figure 1 summarizes the features of guarantees.) To be concrete, consider this guarantee from the auction example in Section 3:

will-ship-item$(S, A, \textsc{reg-seller}, \textsc{first-bid}, \textsc{close-auction}(winbid.Buyer), \textsc{ship-item})$

This says that Seller S guarantees to Arbiter A that it will ship item to the Buyer associated with the winning bid; the guarantee is granted at REG-SELLER, triggered by FIRST-BID, bound at CLOSE-AUCTION, when A identifies the winning bid and the corresponding Buyer, and discharged at SHIP-ITEM. The template for a guarantee is then: guarantee-name$(Guarantor, Requestor, \textsc{grant}, \textsc{trigger}, \textsc{bind}, \textsc{discharge})$

In terms of events and histories the meaning of the guarantee can be expressed as follows:

$$(\text{GRANT} \in H \wedge \text{TRIGGER} \in H) \Rightarrow (\text{DISCHARGE} \in H \wedge (\text{TRIGGER} \rightarrow \text{DISCHARGE}))$$

In temporal logic terms, this can be written (where $\Diamond$ is the temporal operator *eventually*):

$$\text{GRANT} \Rightarrow (\text{ TRIGGER} \Rightarrow \Diamond \text{ DISCHARGE })$$

In the above temporal statement, we have abused notation for the sake of brevity: the "predicate" referred to by each event name E should in fact be "$\text{E} \in H$".

Both formulations say that if the guarantee has been granted, once it is triggered it will eventually be discharged. For simplicity we do not specify the event BIND, which does not significantly add to the semantics of whether a guarantee is discharged (BIND specifies when certain *values* associated with the guarantee will be bound, and always occurs between TRIGGER and DISCHARGE).

Although we do not address all aspects of guarantees in this paper, we briefly mention them here.

- Guarantees may include an expiration (for example, credit card authorizations expire after a while) or other clauses for withdrawal by the guarantor.

- Guarantees may allow the requestor to release the guarantor from a guarantee when it is no longer needed.

- A guarantee may involve more than two parties: a system component (requestor) may ask for a guarantee which will be used by another (*beneficiary*). Also, a component may offer a guarantee that is actually supported by another (*honorer*); this generally is a way of hiding or abstracting guarantees between the guarantor and the honorer. For instance, in the auction example, the Arbiter offers the Seller a guarantee that some Buyer will buy if there is a bid, but this is supported by guarantees from Buyers to the Arbiter.

- Typically, the guarantor gives some sort of receipt or key to the requestor. For example, when a merchant obtains a credit card authorization from a bank, the merchant is given an *auth-id*. When the merchant later wants to submit a charge on that credit card, it presents the *auth-id* to the bank in order to validate the charge. Although it is the bank's commitment that it will remember what *auth-id* stands for *in spite of failures*, it is the responsibility of the merchant to safeguard its copy of *auth-id* also in spite of failures.

- Even though for the purposes of our discussion here we assume that guarantees are either discharged at most once, or their discharge is idempotent, this need not be the case in general.

- Quite often, there is an interdependence amongst guarantees. We have observed examples in which a component may rely on another guarantee in order to discharge its guarantee. Also, obtaining one guarantee may preclude requesting another.

parameter	description	example
Requestor	component that requests the guarantee or benefits from it, usually the same	merchant requests an authorization for a credit card for some amount
Guarantor	component that offers the guarantee or honors it, usually the same	bank guarantees it will pay credit card charge it had previously authorized
GRANT	after the grant event, the guarantee has been obtained by the requestor who may or may not use (trigger) it	seller registers for an auction and promises arbiter to pay commission if there is a sale
TRIGGER	after the trigger event, the guarantee *will* be used	merchant presents charge request, with authorization ID
BIND	event after which values of guarantee are bound; happens at or after triggering	auction ends, defining final sale price for item
DISCHARGE	at this event the guarantee is used and its promise fulfilled	seller ships the item to the buyer

Figure 1: Characteristics of a Guarantee

High-level properties. This catch-all "high-level" designation is for properties that characterize a system in the most abstract manner. They are also high-level in that they can be constructed from guarantees and protocols. We distinguish between two groups of high-level properties:

1. properties of a broader class of systems (e.g. electronic commerce in general).

   An example of an electronic commerce property (we term it EC invariant) is that goods be exchanged for money. Figure 2 lists some common invariants, described by Tygar [Tyg96] as forms of "atomicity."

2. Properties specific to the system under consideration (e.g. the web auction scenario).

   An example of a system-specific property is the expectation that the highest bidder in an auction will get the item being auctioned.

property	description	example
Money Atomic	Money is never destroyed	A cash transaction, with physical preservation of cash
Goods Atomic	Goods are exchanged for Money, and Money Atomic	A cash-on-delivery purchase
Certified Delivery	Both parties verify what goods were exchanged, and Goods Atomic	Delivery with exchange of crypto certificates

Figure 2: Electronic Commerce Invariants

3 Auctions on the Web

In this section we study an example based on Web Auction sites such as eBay.com and Amazon.com, which, offer support for individuals to buy and sell through software-controlled auctions. Our focus is on characterizing this EC scenario in terms of the expectations of all the participants, distinguishing its essentials from particulars of the supporting technology.

Consider a simple auction scenario, comprised of a Seller, many Buyers, and an Arbiter; for simplicity, we assume that items are auctioned one at a time, and we do not consider a separate shipper component. Briefly, an auction proceeds as follows: A Seller places an item for auction with the Arbiter; this involves specifying a few auction parameters (base amount at which bidding begins, and optionally, reserve price, take-it price, whether there's a first-bidder discount, auction duration, etc.). Prospective bidders (the Buyers) register themselves with the Arbiter; this involves promising to buy any item whose auction a Buyer wins. The Arbiter opens the auction as specified, and proceeds to accept bids per the terms of the auction; eventually, the Arbiter closes the auction and declares a winning bid (the price to be paid for the item), and a winner (the buyer who placed the winning bid). Then the Seller and winning Buyer communicate and exchange payment for the item.

In the rest of this section we examine in detail this Web Auction scenario. We describe it in terms of the guarantees the different parties request or grant, and the protocols they follow. We also identify the high-level properties that hold in this scenario, both specific to the Web Auction and general to Electronic Commerce. By showing how the former supports the latter, we decompose the system in terms of the properties that it must maintain in order to function correctly. In particular, we identify phases of the electronic commerce transaction that is a Web Auction and how the guarantees ensure that, past a point of no return, a transaction is assured of successful completion.

3.1 Protocols followed in a Web Auction

The following specifies the protocols followed by the different parties to an auction.

- before-open REG-SELLER → OPEN-AUCTION

  There must exist a Seller registered for there to be an auction.

- after-close The following states the events that can only happen after the auction closes:

  CLOSE-AUCTION → PAY-BID

  CLOSE-AUCTION → PAY-COMM

  CLOSE-AUCTION → SHIP-ITEM

  CLOSE-AUCTION → GET-ITEM

  In our scenario, we require no ordering amongst events PAY-BID, PAY-COMM, SHIP-ITEM, and GET-ITEM. But if the Seller and the Buyer settle among themselves, as in Amazon.com, PAY-BID → SHIP-ITEM must hold too, plus other requirements; see Section 4. Also, here we assume perfect, instant shipping in our model (seeSection 3.3). If we considered the shipper we would have to introduce: SHIP-ITEM → GET-ITEM.

- bid-time $\forall k$ OPEN-AUCTION → BID_k → CLOSE-AUCTION

  All bids happen between open and close of the auction.

- bid-base $\forall k\ \mathrm{BID}_k.\mathit{price} \geq \mathit{basebid}$

  All bids are above a predetermined minimum, the $\mathit{basebid}$.

- bid-up $\forall j, k\ \mathrm{BID}_j \rightarrow \mathrm{BID}_k \Rightarrow \mathrm{BID}_j.\mathit{price} < \mathrm{BID}_k.\mathit{price}$

  A new bid is always higher than all the preceding bids.

- bid-max $\forall k : B_{I_k} = \mathrm{BID}_K.\mathit{Buyer} \Rightarrow \mathrm{BID}_K.\mathit{price} \leq B_{I_k}.\mathit{maxbid}$

  A Buyer always bids at most up to its own self-imposed maximum.

3.2 Guarantees for a Web Auction

Here we consider the guarantees made by the parties to each other. Each guarantee represents a promise on the part of its guarantor that it will perform in a certain way, maintaining some property of interest to the grantee. For example, a Buyer promises to buy an item if it is the winning bidder in that item's auction; a Seller promises to pay a commission to the Arbiter if the item is bid on. We summarize the guarantees in Figure 3, and we elaborate on the details of each in the timeline of the next subsection.

guarantee name	guarantee informal description
will-open-auction	Arbiter will open the auction as per agreement with Seller.
will-close-auction	Arbiter promises (to all) to close the auction as per agreement with Seller.
will-id-winner	Arbiter will identify for Seller the winning bid at the close of the auction.
will-ship-item	Seller will ship item to Buyer indicated by Arbiter
will-pay-comm	If Arbiter identifies a Buyer with a winning bid, Seller will pay a commission to Arbiter.
will-get-item	Arbiter promises that if Buyer wins the bidding then Buyer will get the item.
will-pay-bid	If Buyer wins the bidding, Buyer will pay winning bid price to Seller

Figure 3: Web Auction Guarantees

3.3 Timeline of a Web Auction

Here we outline how a Web Auction proceeds in terms of its significant events, highlighting the granting, triggering, binding and discharging of guarantees, and the protocols followed, by all the parties in the auction. The Web auction we consider is the simplest online version of an English Auction, in which buyers bid in front of one another during a certain period, and the auctioneer records each bid, assigning the (single) item auctioned to the winning buyer. The winning buyer is the one who made the highest bid before the closing of the auction, and the price to be paid is the value of that bid; the seller pays a commission to the auctioneer. Online auctions typically last days and close at the announced closing time, with some extension if there is active bidding. Typically, the auction continues past its announced end if there was any bidding in the last ten minutes. Also, the auction website typically conflates several actors of the auction, in that it represents the auctioneer, who takes bids and defines the winner, and the buyers, each of whom has a proxy bidding on their behalf at the auction website, according to predetermined limits. Thus to avoid confusion in the sequel we refer to an Arbiter as the neutral entity which regulates the progress of the auction. See the graphical representation of the timeline of the auction[5] in Figure 4.

1. REG-SELLER Seller registers item with Arbiter for sale, specifying base bid, and optionally reserve price, take-it price, auction period, etc. Success of the Registration generates guarantees from Arbiter A to Seller S and from S to A:

    - will-open-auction $(A, S,$ REG-SELLER, REG-SELLER, REG-SELLER, OPEN-AUCTION$)$

[5] To reduce clutter we omit guarantees will-open-auction and will-close-auction from Figure 4.

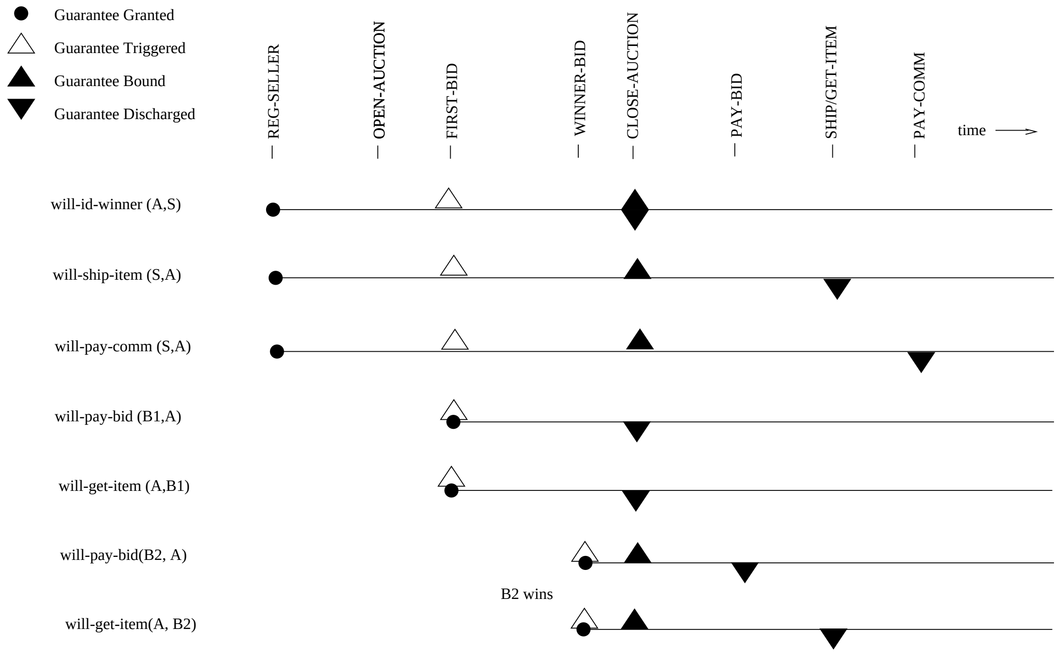

Figure 4: Timeline of a Web Auction

Arbiter will open the auction as per agreement with Seller.

- will-close-auction (A, all, REG-SELLER, OPEN-AUCTION, OPEN-AUCTION, CLOSE-AUCTION)

  Arbiter promises (to all) to close the auction as per agreement with Seller. The duration of the auction is part of this guarantee, but we do not make time explicit in its description.

- will-id-winner(A, S, REG-SELLER, FIRST-BID, CLOSE-AUCTION($winbid$), _)

  Arbiter will identify the winning bid of the auction, i.e., the Buyer who made the winning bid and the bid's amount ($winbid.Buyer$, $winbid.price$). The guarantee is triggered by the FIRST-BID event and is both bound and discharged at the event CLOSE-AUCTION.

  We assume for simplicity that the auction will have at least one bid; $winbid$ is undefined if there are no bidders in the auction.

- will-ship-item(S, A, REG-SELLER, FIRST-BID, CLOSE-AUCTION($winbid.Buyer$), SHIP-ITEM)

  Seller will ship item to the winning bid Buyer; the guarantee is triggered by FIRST-BID but only bound at CLOSE-AUCTION, when Arbiter identifies the winning bid and the corresponding Buyer. It is discharged by the event SHIP-ITEM.

- will-pay-comm(S, A, REG-SELLER, FIRST-BID, CLOSE-AUCTION($winbid.price$), PAY-COMM)

  Seller will pay commission on the sale to the Arbiter; the guarantee is triggered by FIRST-BID but only bound at CLOSE-AUCTION, when Arbiter identifies the winning bid and the corresponding item sale price, from which the commission is calculated. It is discharged at event PAY-COMM.

Remark: Buyers must also register to participate in the auction. For simplicity, we assume (as is often the case) that a Buyer registers the first time the Buyer bids. See event BID, below.

2. OPEN-AUCTION   Arbiter announces it is ready to accept bids. The guarantee will-close-auction is triggered.

   Remark: For simplicity we omit representing time explicitly; if we did, Arbiter would guarantee (to Seller) holding the auction between the times specified by Seller. For the purposes of this example, it is enough to have the open and close events to bound the occurrence of the bid events in the auction.

3. BID$_1 \ldots$ BID$_N$ One or more bids are made. Each bid causes the following guarantees to be granted (BID.$Buyer$ is the buyer who issued the bid):

   - will-pay-bid(BID.$Buyer$, A, BID, CLOSE-AUCTION $\wedge winbid.Buyer =$ BID.$Buyer$,
     CLOSE-AUCTION($winbid.price$), PAY-BID)

     B promises A that it will pay (Seller) *if* it wins the auction; the binding at CLOSE-AUCTION states the amount to be paid. The guarantee is discharged at event PAY-BID.

- will-get-item(A, BID$Buyer$, BID, CLOSE-AUCTION $\wedge winbid.Buyer =$BID.$Buyer$, GET-ITEM)

  Arbiter A promises that *if* BID.$Buyer$ wins the bidding then it will get the item. The guarantee is not just granted and triggered by the event BID. If BID.$Buyer$ wins, the guarantee is bound at CLOSE-AUCTION and discharged at SHIP-ITEM, which by convention we assume is the same as GET-ITEM, in order to abstract away the behavior of the shipper.

FIRST-BID: If this is the first bid of the auction, the event triggers guarantees will-ship-item and will-pay-comm. *Notation:* We abuse notation and write FIRST-BID to identify the first BID event.

Remark: The first bid is a point of no return for the Seller and Arbiter: from here on there will be a sale, and a commission will be paid, even if the buyer and the price are not established yet. We assume that there is at least one bid in each auction to keep the example simple.

4. CLOSE-AUCTION Arbiter closes the auction as agreed with the Seller. This event binds and discharges guarantee will-id-winner, as follows. Arbiter identifies the winning bid WINNER-BID (i.e., the winning Buyer and the price), and notifies the winning Buyer and Seller. Arbiter binds $winbid$ to the highest bid, i.e.,

   Let $winbid.Buyer = \text{BID}_w.Buyer$, and $winbid.price = \text{BID}_w.price$,

   where w is such that $\forall k \neq w$ $\text{BID}_k.price < \text{BID}_w.price$. We write WINNER-BID to denote BID_w.
   Now it is certain that $winbid.Buyer$ will get item, Seller will get paid $winbid.price$.

   CLOSE-AUCTION also binds will-pay-comm, i.e., Seller will pay commission for the sale to Arbiter.

5. The following events all happen after CLOSE-AUCTION, but need not be ordered with respect to each other.

   PAY-BID $winbid.Buyer$ transfers money in amount $winbid.price$ to Seller; this discharges will-pay-bid.

   SHIP-ITEM Seller ships item to $winbid.Buyer$, this discharges guarantee will-ship-winner. To keep the shipper out of the model and the scenario simple, we assume perfect, instant shipping, by making the events for shipping and receiving the item the same.

   GET-ITEM = SHIP-ITEM $winbid.Buyer$ receives item shipped by Seller, this discharges guarantee will-get-item.

   PAY-COMM Seller pays commission to Arbiter, which discharges guarantee will-pay-comm.

   It is interesting to note that all three (four, but for simplicity we consider ship and get to be identical) events are independently assured to happen by earlier guarantees. In the case of Amazon, PAY-ITEM must precede SHIP-ITEM because the seller and buyer must agree on the shipping method and therefore cost, which is added to the purchase price. In general, we can remove this precedence requirement if the shipping cost is parameterized: for example, the buyer knows ahead of time all the shipping options and costs for the item in question, or the shipping may simply be free. In this simple example we ignore the shipping costs.

3.4 Web Auction High-level Properties

As mentioned earlier, there are two kinds of high-level properties: the ones that characterize correct Electronic Commerce systems in general – for example, goods atomicity, and the ones that characterize the correct behavior of the specific Electronic Commerce scenario – for example, the highest bidder gets the item being auctioned. In this subsection we consider the high-level properties specific to the Web Auction example. In the next subsection, we consider the global Electronic Commerce Invariants that hold in this example.

We motivate these high-level properties by describing what each party expects of a fairly conducted Web Auction. After characterizing the web auction properties, we show that they hold as a consequence of the guarantees and protocols described before. One class of properties matter to all parties in the auction: for example, that the auction will end, and that there is a well defined winner and price for the item. The second class of properties are important to specific parties: for example, the Arbiter cares that if the auction is held and there is a bidder, the Arbiter will collect the commission.

Buyer High-level Properties. Informally, a buyer bidding for an item expects to either (i) win the auction and buy the item at a price less than or equal than its self-imposed maximum bid; or (ii) lose the auction and buy nothing and disburse no money. Also, buyers expect fairness in the auction, i.e., that the winner and losers result per the usual auction rules. Here are the high-level properties for buyers:

1. If there is at least one valid bid, there exists a buyer who will win the auction
2. The buyer winner of the auction will get the item at the winning bid price.
3. A buyer will not pay more than its self-imposed maximum bid.
4. The buyer winner of the auction is the highest bidder at the end of the auction.

In the interest of space, we just outline the proofs for the properties here.

Proof outline of Property (1): The set of bids is nonempty and finite; the bids are all distinct; there is thus a unique buyer whose bid is the maximum in the set of bids; that buyer is the winner of the auction. This argument holds for the incremental version of the set – which buyer is the winner changes with each new bid, but there is always a winner. That the set is nonempty comes from the hypothesis - at least one valid bid. That all bids are distinct (in fact, increasing in magnitude) comes from the protocol bid-up; thus, there is exactly one winner. This covers the existence of a “temporary winners;” to see that the there is a winner of the auction, we must require that the auction end, which fixes the set of bids just described for good. The auction ends because each buyer has a maximum bid; let M be the maximum of the set of maximum bids – then the auction cannot go beyond a bid of M, as that would contradict the protocol bid-max that all buyers follow. In fact the auction may not reach that value because it has a time limit specified by the Seller, which the Arbiter guarantees to uphold via its will-close-auction guarantee, but we argue by bounding the bids because in some cases (Amazon) the auction is extended as long as there is a bid in the ten minutes before its announced end, in which case auctions may last beyond the specified closing time as long as there are bidders.

Thus we can formalize the guarantees and protocols as rules on how the set of bids can be modified during the course of the auction.

Proof outline of Property (2). It holds because the Arbiter gives the guarantee will-close-auction, which says that the price quoted is that of the well-defined winning bid, and because the winner has the winning bid by the Arbiter’s guarantee will-id-winner.

Property (3) follows trivially from protocol bid-max. The interesting point here is that in a Web auction, the bidding is conducted by a proxy for each buyer; hence, the enforcement of the protocol bid-max can be cast as a guarantee the Arbiter offers each buyer when they register, and then the generation of bids is left to the Arbiter. Were that the case, the granting event of e.g. will-pay-bid would change to some REG-BUYER event instead of the current BID.

Property (4) follows from the binding and discharging of guarantee will-id-winner.

Seller High-level Properties. Informally, a seller auctioning an item expects that the item will sell at the base bid value or more, and the buyer will pay the winning bid price. Specifically:

1. If there is at least one bid, the item will be sold.
2. The price for the item will be the highest amount bid for it.
3. The item will be sold for at least the base bid amount specified by the seller.

Proof outline of High-level Property (1). We have assumed in this example that there is at least one bid in the auction, to simplify the treatment, but the property is interesting nonetheless. That the item will be sold means that a buyer will pay for it, and that the same buyer will receive it. Guarantee will-id-winner assures the Seller that such a buyer (and price) is well defined: $winner.Buyer$. The binding of guarantee will-ship-item ensures the item goes to $winner.Buyer$. Both guarantees will be eventually triggered and bound because the auction ends (see paragraph above) and guarantee will-close-auction. A similar arguments shows that $winner.Buyer$ will pay its winning bid, thus completing the sale.

Property (2) follows from the semantics of the binding of will-id-winner at event CLOSE-AUCTION. Property (3) follows from the bid-base protocol.

Arbiter High-level Properties. Informally, an Arbiter conducting an auction expects that if there is at least one bid on the item, the item will sell and the Arbiter will collect a commission from the Seller. Proving that the Arbiter will collect commission is analogous to Seller property (1) above.

3.5 General Electronic Commerce Properties

Tygar [Tyg96] identifies system-wide high-level properties that are desirable for all Electronic Commerce situations; we call them invariants. These are three kinds of invariants, as summarized in Figure 2.

Here we briefly indicate how these Electronic Commerce properties relate to the high-level properties specific to the auction. Both Money Atomicity and the stronger Goods Atomicity hold, as follows. We can assume Money Atomicity holds because of the underlying method to exchange money – which we have left unspecified but assume it is done with a correct credit-card system. (In the next section, we elaborate this aspect of Electronic Commerce and justify the assumption that goods atomicity is supported with standard credit-card practices).

That Goods Atomicity holds follows from the observation that at the end of the auction there is exactly one Buyer $(winner.Buyer)$ who will transact with the Seller; the other Buyers will not fulfil their will-pay-bid guarantees because the trigger of that guarantee is bound to $winner.Buyer$. This is from Buyer Property (1) above. This settles the identification of the buyer. That the money gets exchanged for goods follows from Seller Property (1). Thus we have Goods atomicity in this example.

Our auction example does not provide for Certified Delivery. Amazon.com offers a redress mechanism (for items “materially different” from what was advertised), which coupled with shipping records is a step towards Certified Delivery, albeit using compensation after the fact.

In the next section we discuss another Electronic Commerce scenario.

4 Credit Card Purchase on the Web

Here we will analyze a scenario of an online Merchant selling to a Customer who pays with a credit card; a Bank functions as a trusted third party for the transfer of money, and a Shipper does the transfer of goods. That scenario proceeds thus: Customer browses, places an order for goods with a credit card, and Merchant checks stock and obtains credit card authorization from Bank (if either fails the sale is cancelled). With item in stock and a charge authorization from the Bank (this is the guarantee the Bank gives the Merchant), Merchant can guarantee Customer that the goods ordered will reach the Customer; the Customer confirmation of purchase triggers both guarantees (delivery, and payment). Merchant uses its own state (that goods are in stock), and a guarantee from the Shipper to support its guarantee to the Customer, and can ship assured of collecting because the Bank guarantees that the future credit card charge will go through. Merchant ships good to Customer via Shipper, and then presents its charge authorization to the Bank to collect the money owed. Later the Shipper verifies the delivery; if the delivery were to fail, compensation must be used; for example Shipper would refund the Merchant who will refund the Bank who will refund the Customer. The atomicity of the exchange of goods for money that is obvious to verify in a conventional, face-to-face cash purchase is obtained here via a multi-phase protocol: Customer supplies credit card, Bank promises to pay, Merchant Ships, then collects, then Customer receives goods. (In this section, M stands for the Merchant, C for the Customer, B for the Bank, and S for the Shipper.) For simplicity, we assume that a single Bank takes care of the Customer and the Merchant's side; in the real world, there would be at least two banks involved, with mutual guarantees for the transfer of charges and payments.

Notice that this scenario subsumes a possible implementation for the exchange of money for goods in the case of the auction. The Merchant is the Seller in that case, and the Customer, the Buyer who won the auction. (In particular, in the Web Auction scenario as described in Section 3, in principle one need not get authorization – Buyer gave a guarantee to Arbiter – or to check Stock –it is in a guarantee that Seller gave Arbiter to be able to put the item on auction.)

Generally, the mechanism involves credit cards and banks. In Amazon.com, for example, one of the options for participants in the auction is to settle their transactions on their own, and that is typically done by credit card. Thus this example shows the composability of our approach, in that it can be used a component of the web auction example to show a particular implementation of one of its portions.

Here it is interesting to consider the fate of guarantees in the face of failures. For example, the credit card authorization issued by the Bank, which guarantees that a later charge will be accepted and the money transferred, involves a “receipt” (an authorization-id) that the Bank gives the Merchant. Obviously, the Bank must keep its accounts in spite of failures, and must remember the authorization and honor the charge in spite of intervening failures. But it is the responsibility of the Merchant to save that receipt (i.e., to remember, in spite of failures, the guarantee given, so as to use it later if necessary). This suggests that guarantees at the Bank and the Merchant may be implemented via the commitment of some internal database transactions.

4.1 Protocols followed in Credit Card Purchase

- auth-checked The following states the precedence ordering of events that connected with the authorization been *requested*. Note that the outcome of the request may be yes or no; which path is taken is not the concern of the protocol, as long as it is one of the legitimate paths.

  GET-AUTH → ACCEPT-ORDER

  GET-AUTH → RESTOCK

- stock The following states the precedence ordering of events related to the reservation of stock, successful or not, and the release of stock in case of cancellation of the order.

  ENTER-ORDER → ESCROW-STOCK

  ESCROW-STOCK → RESTOCK

  GET-AUTH → REJECT-ORDER

  RESTOCK → REJECT-ORDER

  ESCROW-STOCK → ACCEPT-ORDER

- closing The following states the precedence ordering on events related to closing of the transaction.

  ACCEPT-ORDER → SHIP-ORDER

  SHIP-ORDER → CHARGE-ORDER

  SHIP-ORDER → DELIVER-ORDER

4.2 Guarantees for Credit Card Purchase

We list in Figure 5 the guarantees that concern the pairwise expectations amongst all the parties. We describe them in detail as they appear in the timeline subsection, below. However, the complete lifecycle of guarantees will-auth-charge and will-settle-charge, which correspond to the relationship between the

guarantee	guarantee informal description
will-accept-order	Merchant will accept order from Customer (conditional on auth. and stock)
will-ship-order	Merchant will ship order to Customer once order accepted
will-deliver-order	Shipper promises Merchant Shipper will deliver to Customer once it is given order
will-charge-order	Bank promises Merchant it will process a charge that had been authorized
will-auth-charge	Bank promises Customer it will authorize charge requests per their contract
will-settle-charge	Customer promises Bank it will settle credit card charges per their contract

Figure 5: Credit Card Purchase Guarantees

Customer and the Bank, will not appear in the timeline below. Because that relationship is ongoing and lasts much longer than a specific credit card purchase, it would be artifical to present all its relevant events on the purchase timeline. We strike a balance between completeness and simplicity by describing their role even though we do not show their granting.

4.3 Timeline of a Credit Card Purchase

The following is a timeline of events during a Credit Card Purchase, but it can also be seen as another level of detail of the Web Auction example, because a Seller and a Buyer might close their transaction in this manner. Notice that in the Web Auction the guarantees between Seller and Arbiter ensure the availability of the item, and the guarantees between Buyer and Arbiter ensure the payment of the credit card charges.

The timeline below is one possible outcome of following the protocols above, which are somewhat less restrictive (e.g., GET-AUTH and ESCROW-STOCK could happen concurrently).

1. ENTER-ORDER Buyer supplies contact, shipping, and credit card information, and confirms it wants to buy the item. The guarantees granted at this event are:

    - will-accept-order$(M, C,$ ENTER-ORDER, (GET-AUTH.$OK \wedge$ ESCROW-AUTH.OK), _, SHIP-ORDER$)$
    - will-ship-order$(M, C,$ ENTER-ORDER, ACCEPT-ORDER, _, SHIP-ORDER$)$

2. ESCROW-STOCK Merchant verifies that there is stock on hand to fulfill the order. If OK, this is part of the triggering of will-accept-order.

    If stock is not available, Merchant notifies Customer and the transaction ends without a sale or a charge. In this timeline we assume that GET-AUTH happens only after successfully obtaining stock.

3. GET-AUTH Merchant contacts Bank with Customer's credit card data and total order amount (item price + shipping costs). If the credit authorization is denied, Merchant notifies Customer, releases the item back to the available stock, and the transaction ends without a sale or a charge; that is accomplished by skipping to event RESTOCK in the timeline.

   If the credit authorization is granted, Bank provides Merchant with *auth-id*, which embodies Bank's promise to transfer funds up to the authorized amount to Merchant upon Merchant's later request. The guarantee granted is:

   - will-charge-order(B, M, GET-AUTH.OK, SHIP-ORDER, _, CHARGE-ORDER)

   The success of the authorization (coupled with that of escrowing stock, which happened before, see above) also triggers will-accept-order. In the successful case (stock and authorization OK), this is the point of no return: the order will be filled and sent, and charged, in spite of intervening failures.

4. RESTOCK Merchant releases the stock it had reserved, because for some reason (e.g. no credit card authorization) order cannot be completed. The stock inventory is updated.

5. REJECT-ORDER Merchant notifies Customer that the order has been rejected.

6. ACCEPT-ORDER Merchant notifies Customer that the order has been accepted; Merchant provides Customer with the ORDER-ID which embodies the Merchant's promise to ship the item. Customer can now "go away," i.e., disconnect from Merchant, and be assured that the item will arrive within some time. This triggers guarantee will-ship-order. The Merchant secures a guarantee from the Shipper:

   - will-deliver-order(M, S, ACCEPT-ORDER, SHIP-ORDER, SHIP-ORDER, DELIVER-ORDER)

7. SHIP-ORDER Merchant ships item to Customer via Shipper; this triggers will-deliver-order. It also discharges guarantees will-accept-order and will-ship-order.

8. DELIVER-ORDER Customer receives order from Shipper. This discharges will-deliver-order.

9. CHARGE-ORDER Merchant presents *auth-id* and charge amount to Bank; Bank verifies it and transfers money to Merchant. This discharges guarantee will-ship-order

4.4 High-level Property

In this example we want to prove Goods Atomicity, i.e., that goods get exchanged for money. We only outline the proof here: the Bank will get charged exactly if items have been delivered because (i) will-charge-order is triggered by SHIP-ORDER; and so is will-deliver-order. Likewise, the Merchant will get paid for what was ordered, because will-charge-order is triggered by SHIP-ORDER. The transfer of money between the Customer and the Bank is ensured by the guarantee will-settle-charge, whose granting is outside the timeline.

The usual real world implementation of this scenario by Web merchant is strictly not “goods atomic” in that the goods might never be delivered by the shipper, so there is an ad-hoc a posteriori compensation (refund via insurance, shipper, et al.) Thus goods received implies money disbursed (via the Bank guarantee and Merchant’s own protocol of charging at Shipping or later, and Shipper’s guarantee); and money disbursed implies goods received in a relaxed manner: a Bank may accept a charge today for goods the Customer will receive later. Some package shippers provide real-time on-line verification of delivery, so Merchants and Banks could implement strict goods atomicity by synchronizing the package delivery with the money transfer.

5 Conclusions

The work reported in this paper has been motivated by a need to develop a framework that can deal with EC components and their interactions precisely, elucidating: (i) their behavioral properties, (ii) their dependencies, (iii) the promises they must keep, and (iv) the properties these in turn must satisfy. We wanted to focus on the essentials of electronic commerce transactions, as opposed to descriptions tied to specific technologies.

To illustrate the capabilities of our framework, we took a detailed example of an EC scenario involving electronic auctions on the web, specified the protocols that the bidders, the sellers, and the arbiter must follow and also the guarantees that they depend on each other for. The specifications of these guarantees and protocols were used to demonstarte the correctness of higher level auction-specific properties as well as EC in general.

We also elaborated upon the monetary and good exchange aspects of the auction scenario [Tyg96] by examining the details of the protocols and guarantees involved in credit card authorzations in an electronic retail scenario.

These two examples together indicate that our framework is fairly general in scope, leads to intuitive specifications without reference to underlying implementation/technology, and lends itself to analysis. Our prior work on applying a preliminary form of this framework to mobile transactions also attests to this generality [PR99].

Finally, such a formalization reveals some of the interesting characteristics of electronic commerce interactions in general. For example, any commercial transaction seems to have two phases: “matching” and “closing.” Matching is the process of identifying partners for the transaction, generally a buyer and a seller, establishing guarantees to possible transaction partners (possibly through third parties, such as credit-card companies or auction houses) to make the closing possible. Closing begins with the closing of a deal, followed by the actual exchange of goods for money. Of interest to us is the fact that there is a point-of-no-return, the “closing point” (a commit point, in traditional transaction terms), at which the rest of the transaction is ensured to finish.

From a transactional viewpoint, an EC system is characterized by the conditions necessary for each party to reach the closing point, what triggers the closing point, and what guarantees and protocols must be followed throughout in order to ensure that the commercial transaction committed at the closing point finishes successfully. The conditions, represented as protocols and guarantees, cover both safety properties – no wrong behavior will happen – and liveness properties – the system will make progress.

Interestingly, but perhaps not surprisingly, the foundations for database transactions lie in similar guarantees, protocols, and properties. For example, a transaction processing system follows the protocol that it will not write the result of an operation to disk unless the log record for that operation has been written to disk; the disk subsystem offers the guarantee that if a log record for an operation has been written to disk, that record will be available later for retrieval (to (re)do the operation), and such protocols and guarantees help in achieving high level properties such as failure atomicity.

Motivated by these parallels with database transactions, the fact that a component bases its granting of a guarantee to another upon securing a guarantee from a third, and the relationship of refinement between the auction and the credit card scenario, we will pursue the formal aspect our work by investigating the compositional properties of guarantees.

In addition to investigating the use of our framework to embellish specifications of workflows [KR98, CCPP98] and extended transaction models [CR94, Elm91, BK91] with the guarantees needed to satisfy higher-level requirements, we plan to pursue the above insight to not only precisely describe transaction system components and the protocols that they follow [MHL^{+}92, SAS99] but also bring out the guarantees they offer each other, the latter often not explicitly stated. Since these guarantees underlie the recoverability of transactional systems, and recovery is one of the most difficult aspects of transaction processing systems, we believe there is a lot to be gained by this endeavour.

References

[BK91] Naser S. Barghouti and Gail E. Kaiser. Concurrency control in advanced database applications. *Computing Surveys*, 23(3):269–317, 1991.

[BOGM91] Alejandro P. Buchmann, M. Tamer Özsu, Dimitrios Georgakopoulos, and Frank Manola. *A Transaction Model for Active Distributed Object Systems*, chapter 5. In Elmagarmid [Elm91], 1991.

[CCPP98] Fabio Casati, Stefano Ceri, S. Pernici, and Giuseppe Pozzi. Workflow evolution. *Data Knowledge Engineering*, 24(3):211–238, 1998.

[CR94] Panos K. Chrysantis and Krithi Ramamritham. Synthesis of Extended Transaction Models using ACTA. *ACM Trans. on Database Systems*, pages 450–191, September 1994.

[DHL90] Umeshwar Dayal, Meichun Hsu, and Rivka Ladin. Organizing long-running activities with triggers and transactions. In *Proc. of the ACM SIGMOD Conference*, pages 204–214, May 1990.

[Elm91] A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan-Kaufmann, San Mateo, Calif., 1991.

[GHM96] Dimitrios Georgakopoulos, Mark F. Hornick, and Frank Manola. Customizing transaction models and mechanisms in a programmable environment supporting reliable workflow automation. *IEEE Trans. on Knowledge and Data Engineering*, 8(4):630–649, 1996.

[KF98a] Manoj Kumar and Stuart I. Feldman. Business negotiations on the internet. In *Proc. of inet'98*, Geneva, Switzerland, July 1998. Available at http://www.ibm.com/iac/reports-technical/reports-bus-neg-internet.html.

[KF98b] Manoj Kumar and Stuart I. Feldman. Internet auctions. In *Proc. of the 3rd USENIX Workshop on Electronic Commerce*, Boston, Mass., August 1998. Available at http://www.usenix.org/events/ec98/kumar_auctions.html.

[KR98] Mohan U. Kamath and Krithi Ramamritham. Failure handling and coordinated execution of concurrent workflows. In *Proc. of the 14th IEEE Int'l Conference on Data Engineering*, Orlando, Fla., February 1998. Available at http://www-ccs.cs.umass.edu/db/publications/.

[LR99] David H. Lucking-Reiley. Auctions on the internet: What's being auctioned, and how? Technical report, Vanderbilt University, Nashville, Tenn., August 1999. Available at http://www.vanderbilt.edu/Econ/reiley/papers/InternetAuctions.html.

[MHL$^+$92] C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks using Write-Ahead Logging. *ACM Trans. on Database Systems*, 17(1):94–162, March 1992.

[PR99] Cris Pedregal-Martin and Krithi Ramamritham. Recovery guarantees in mobile systems. In *Proc. of the ACM Int'l Workshop on Data Engineering for Wireless and Mobile Access*, Seattle, Wash., August 1999. Available at http://www-ccs.cs.umass.edu/db/publications/.

[SAS99] Heiko Schuldt, Gustavo Alonso, and Hans-Jörg Schek. Concurrency control and recovery in transactional process management. In *Proc. of the ACM Symposium on Principles of Database Systems (PODS'99)*, Philadelphia, Penn., USA, May 1999.

[Tyg96] J. D. Tygar. Atomicity in electronic commerce. In *Proc. of the 15th Annual ACM Symposium on Principle of Distributed Computing*, pages 8–26, May 1996.

[WH93] Gerhard Weikum and Christof Hasse. Multi-level transaction management for complex objects: Implementation, performance, parallelism. *VLDB Journal*, 2(4):407–453, 1993.