2005-08-01

Model Checking for E-Business Control and Assurance

Bonnie B. Anderson
bonnie_anderson@byu.edu

James V. Hansen

Paul B. Lowry

Scott L. Summers

Original Publication Citation

Anderson, B.B., Hansen, J.V., Lowry, P.B., Summers, S. "Model Checking for E-Commerce Control and Assurance," IEEE Transactions: Systems, Men, and Cybernetics, Vol 35, No. 3, August 25.

Model Checking for E-Business Control and Assurance

Bonnie Brinton Anderson, James V. Hansen, Paul Benjamin Lowry, and Scott L. Summers

***Abstract*—Model checking is a promising technique for the verification of complex software systems. As the use of the Internet for conducting e-business extends the reach of many organizations, well-designed software becomes the foundation of reliable implementation of e-business processes. These distributed, electronic methods of conducting transactions place reliance on the control structures embedded in the transaction processes. Deficiencies in control structures of processes that support e-business can lead to loss of physical assets, digital assets, money, and consumer confidence. Yet, assessing the reliability of e-business processes is complex and time-consuming.**

This paper explicates how model-checking technology can aid in the design and assurance of e-business processes in complex digital environments. Specifically, we demonstrate how model checking can be used to verify e-business requirements concerning money atomicity, goods atomicity, valid receipt, and communication-link failure. These requirements are fundamental to many e-business applications.

Model checking can be used to test a broad range of systems requirements—not only for system designers, but also for auditors and security specialists. Systems that are examined by auditors need to have adequate controls built in prior to implementation and will need adequate auditing after implementation to ensure that none of the processes have been corrupted. Model checkers may also provide value in examining the processes of highly integrated applications as found in enterprise resource planning systems.

***Index Terms*—Atomicity, data typing, e-Business, model checking, process and communication protocols.**

I. Introduction

Internet-based business operations offer considerable potential, but they are accompanied by a broad range of often unprecedented risks. An actual or perceived lack of system security and reliability can significantly constrain the growth of the digital economy. While progress is being made in reducing Internet computational risks through a variety of software patches and cryptographic algorithms, these efforts address only a small portion of the larger challenge of establishing the necessary security and reliability of e-business systems. To resolve this challenge, systematic management of the associated operational risks is essential [1].

According to Wang *et al.* [2], management of operational risks requires careful examination of the e-business infrastructure. Distributed Internet computing is changing e-market structures and e-business models in fundamental ways. Although the flexibility of distributed e-operations supports open accessibility and dynamic interactions, flexibility can intensify problems arising from e-market information asymmetry and e-business operational uncertainty. These problems militate against innovative e-commerce developments. Although e-commerce offers the opportunity for businesses to gain efficiency and effectiveness through network-based ad-hoc partnerships, many businesses do not take advantage of these opportunities because of the heightened risks of operational uncertainty and perceived information asymmetry among unfamiliar business partners.

Manuscript received November 18, 2003; revised May 17, 2004. This paper was recommended by Associate Editor S. Lakshmivarahan.

The authors are with the Marriott School of Management and Kevin and Debra Rollins Center for e-Business, Brigham Young University, Provo, UT 84602 USA (e-mail: Bonnie_Anderson@BYU.edu; James_Hansen@BYU.edu, Paul_Lowry@BYU.edu; Scott_Summers@BYU.edu).

Digital Object Identifier 10.1109/TSMCC.2004.843181

These issues take on added importance as new business models and architectures—such as Internet auctions, web services [3] and the semantic web [4]—offer broad support for loosely coupled, e-commerce transactions where buyers and sellers may not have any prior trading experience with one another. For example, the web services [3] platform provides the Universal Description, Discovery and Integration (UDDI) registry for discovery of e-commerce services, WSDL for service description, and SOAP for transaction execution. These facilities require no prior knowledge of buyer and seller by either party.

In such environments, merchants and customers may be reluctant to trust one another and the following situations may arise: A customer is unwilling to pay for a product without being certain the correct product will be sent. A merchant is unwilling to send a product without certainty of receiving payment. If a merchant delivers the product without receiving payment, a fraudulent customer may receive the product and then disappear, with a resulting loss to the merchant. If a customer pays before receiving the product, a merchant may not deliver or may deliver a wrong product. These possibilities underscore the need for carefully designed e-commerce models that are robust under all events.

As Wang *et al.* [5] note, e-system complexity and human limitations make it impossible to imagine all scenarios and guarantee correct processing under all circumstances—even for carefully designed and implemented code. Much of this difficulty is due to interconnectivity, which widens the potential range of error or vulnerability. Variation in execution of concurrent processes in nonstop, nondeterministic systems increases the potential for automation failures. Consequently, minimizing flaws in transaction protocols is crucial for the survival and sustainability of e-business. Stakeholders, such as system designers, users, and auditors need methods to preclude these subtle but potentially critical mistakes—before erroneous processing occurs or an attacker exploits them—to enhance control and assurance to e-commerce users. Model checking offers a promising method for addressing these issues.

II. Model Checking Fundamentals

Automation failures occur when an automated system behaves differently than its stakeholders expect. If the actual system behavior and the stakeholders model are both described as finite state transition systems, then mechanized techniques known as *model checking* can be used to automatically discover any scenarios that cause the behaviors of the two descriptions to diverge from one another. These scenarios identify potential failures and pinpoint areas where design changes or revisions should be considered.

Model checking can trace through all relevant states with respect to any given requirement. Since model checking operates on logic rather than individual execution paths, verification can be more thorough and efficient than test runs and simulation. Some of the most compelling features of model checkers are summarized as follows [6].

1) They help delimit a system's boundary or the interface between the system and its environment.
2) They precisely define a system's desired properties.
3) They characterize a system's behavior more accurately. Most current methods focus on functional behavior only (e.g., "What is the correct answer?") but some can handle real-time behavior as well (e.g., "Is the correct answer delivered on time?").
4) They can aid in proving that a system meets required specifications. By providing counterexamples that show how specifications are not satisfied, model checkers can pinpoint the circumstances under which a system does not meet its specifications. This can also help to correct the system.

These features of model checkers aid stakeholders in two important ways.

1) Through specification, by focusing a system designer's attention to crucial questions, such as: What is the interface? What are the assumptions about the application's environment? What is the system supposed to do under this condition or that condition? What happens if that condition is not met? What are the system's invariant properties?
2) Through verification, by providing additional assurance. Relying on proof that a system meets its security goals is better than relying on opinion—even expert opinion.

It should be emphasized that any proof of correctness is relative to both the formal specification of a system and the formal specification of the desired properties: a system proven correct with respect to an incorrect specification leaves no assurance about the system at all.

The process of proving entails three actions: First, the system of interest must be modeled. A mathematical model is constructed that expresses the semantic structure of an e-business implementation. Second, all properties to be guaranteed in the implementation are formally specified. In an e-business context, one such specification might be that goods must always be received before payment is initiated. Third, a proof is provided. Typically, a proof relies on induction over traces of the e-commerce communication and transaction operations.

In general, verifying that any e-business process is resilient to hidden flaws and errors is a daunting task. Manual methods are slow and error prone. Even theorem provers, which provide a formal structure for verifying standard characteristics, may require human intervention and can be time-consuming. Moreover, even if a failure is found using a theorem prover, it may provide little help in locating the source of the failure [2]. Simulations offer computational power, but they are ad hoc in nature and there is no guarantee they will explore all important contingencies [2].

In contrast, model checking is an evolving technology that can provide effective and efficient evaluation of e-business processes. Model checking was originally developed for validating highly complex integrated circuits and software packages [7], [8], but it has recently been adopted to tackle the complexity of e-commerce transactions [9], [2], [10]. Current model-checking technology is based on automated techniques that are considerably faster and more robust than other approaches, such as simulation or theorem proving. Model checkers can analyze very large state spaces in minutes. Additionally, model checkers aid failure correction by supplying counterexamples when processing failures are discovered [11].

Although model checking is still relatively new [9], [2], [10], it has shown impressive performance in the analysis of complex hardware and software processes [7], [8]. The extended fair-exchange protocol [10] used in this paper incorporates methods fundamental to a broad class of e-business processes, including distributed processing, parallelism, concurrency, communication uncertainties, and continuous operations.

For this study, we use the failures-divergence refinement (FDR) model checker, which is based on the concurrency theory of communicating sequential processes (CSP) [8]. FDR verifies whether a requirement is satisfied in a particular system design by performing tests on a refinement of that system. That is to say, verification is done by comparing the two models to check whether one is a refinement of the other.

A CSP process is represented as a labeled transition system (LTS), which is a set of nodes representing the states of the process that are connected by directed arcs, labeled with an event. LTS describes the evolution of state transition. Starting from the distinguished node n_0, the state is always one of the nodes, and progress is made by performing one of the actions possible for that node. Most actions are visible to the external environment and can only happen with its cooperation. Internal actions cannot be seen from the outside or occur automatically [8].

Such trace refinements are normally used for proving required properties. The refinement-checking algorithm in FDR is abstracted as follows:

```
BEGIN
    Initialize explorer with a pair of initial nodes;
    WHILE
        explorer is not complete
            Examine a fresh pair of nodes from the explorer
            IF
                Nodes are compatible in appropriate model
            THEN
                Add mutually reachable successor pairs to explorer
            ELSE
                Return counterexample with explorer path to this pair
    END
END
```

The objective is to establish that performance requirements are satisfied by the system design.

III. Case Study

Fig. 1 represents the high-level abstraction of a protocol proposed in [10]. The essential processes are summarized in the following paragraphs. We first provide an overview and then provide the necessary details for model checking.

Messages are exchanged between a customer, a merchant, and a trusted third party (TTP). A merchant has several digital products to sell. The merchant places a description of each product in an online catalog service with a TTP, along with a copy of the encrypted product. When a customer finds a product of interest by browsing the catalog, the customer downloads the encrypted product and then sends a purchase order to the merchant. The customer cannot use the product unless it has been decrypted, and the merchant does not send the decrypting key unless the merchant receives a payment token through the purchase order process. The customer does not pay unless the customer is sure that the correct and complete product has been received. The TTP provides anonymous support for purchase order validation, payment token approval, and approval of the overall transaction between the customer and the merchant.

At a detailed level, the customer first browses the product catalog located at the TTP and chooses a product. The customer then downloads the encrypted product along with the product identifier, which is a file that contains information about the product such as its description and checksum. If the identifier of the encrypted product file matches the identifier in the product identifier file, the transaction proceeds. If the identifiers do not match, an advice is send to the TTP, and the customer waits for the correct encrypted product file. This process ensures that the customer receives the product that was requested from the catalog. Next, the customer prepares a purchase order (PO) containing the customer's identity, the merchant identifier, the product identifier, and the product price. A cryptographic checksum is also prepared. The PO, along with the cryptographic checksum, is then sent to the merchant. The combination of the PO and cryptographic checksum allows the merchant to ascertain whether the received PO is complete or whether it was altered in transit. Upon receipt of the PO, the merchant examines its contents. If the merchant is satisfied with the PO, the merchant endorses it and digitally signs the cryptographic checksum of the endorsed PO. This is forwarded to the TTP. The TTP is involved in the process to prevent the merchant from later claiming nonacceptance of the terms and conditions of the transaction. The merchant also sends the TTP a single-use decrypting key for the product. The merchant next sends a copy of the encrypted product to the customer, together

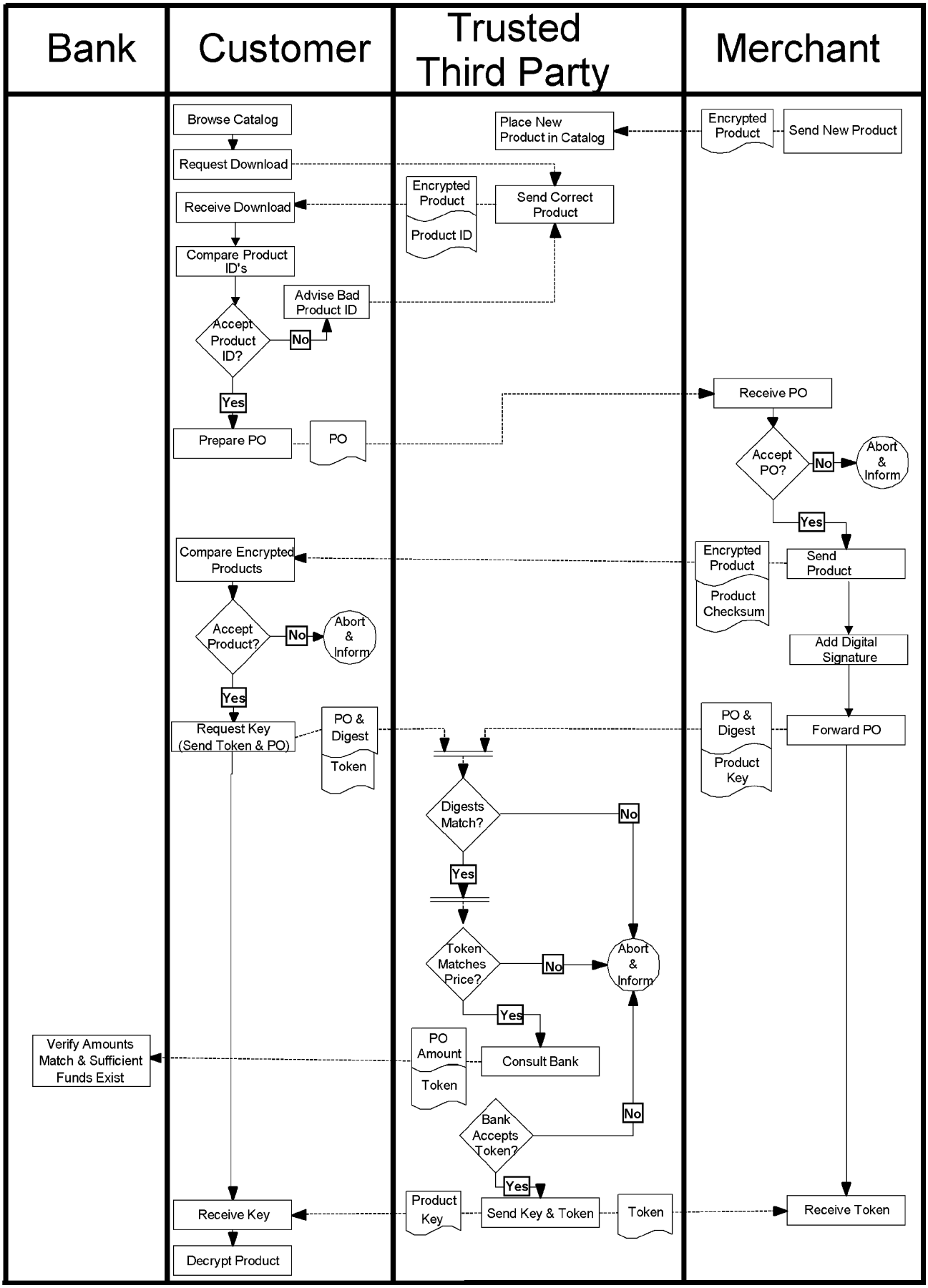

Fig. 1. Sequence diagram for comprehensive e-Business protocol.

with a signed cryptographic checksum. The signed cryptographic checksum establishes the product origin and also provides a check to verify whether the product has been corrupted during transit.

Upon receipt of this second copy of the encrypted product, the customer validates that the first and second copies of the product are identical. Through this process, the customer can be assured of receiving the actual product ordered. The customer then requests the decrypting key from the TTP. To do this, the customer forwards the PO and a signed payment token to the TTP, together with the cryptographic checksum. The payment token contains the customer's identity, the identity of the customer's financial institution, the customer's bank account number with the financial institution, and the amount to be debited from the customer's account.

To verify the transaction, the TTP first compares the digest included in the PO from the customer with the digest of the same PO from the merchant. If the two do not match, the TTP aborts the transaction. Otherwise, the TTP proceeds by validating the payment token with the customer's financial institution by presenting the token and the sale price. If the financial institution does not validate the token, the TTP aborts the transaction and advises the merchant accordingly. If the financial institution does validate the token, the TTP sends the decrypting key to the customer and the payment token to the merchant, both digitally signed with the TTP's private key.

Secure channels guarantee the confidentiality of all messages throughout this protocol. The protocol ensures money atomicity if the payment token generated by the customer contains the amount to be debited from the customer's account and credited to the merchant's account. Consequently, no money is created or destroyed in the system by this protocol.

Goods atomicity is guaranteed if the TTP hands over the payment token only when the customer acknowledges receipt of the product. The process also ensures that the product is actually available to the customer for use only when the customer consents to payment by acknowledging receipt of the product.

Delivery verification is guaranteed if the TTP receives a cryptographic checksum of the product from the merchant. Also, the customer independently generates a checksum of the product received and sends it to the TTP. Using these two copies of the checksum (available at the TTP), both the merchant and the customer demonstrate proof of the contents of the delivered good.

Controls are integrated into the modeling process by specifying limits or boundaries within the model. We offer several examples: Message types in the implementation can be constrained such that only certain types can be sent by the various agents. For example, it can be asserted that a *customer* may send only certain types of messages to the TTP and *merchant*. The e-business model can then be tested to determine if this specification can be violated under any conditions. The content allowable in various messages can be also be stipulated and tested. It also may be important to specify the conditions under which agents can abort a transaction, if at all.

There are also issues of protection from interruption. Systems are often designed under the assumption that transaction links are completely reliable. The reality may be that technical failures or attacks may undermine this expectation. It is safer to specify and verify properties to cope with unreliable communications or communication failures.

IV. Model Checking the System

For purposes of demonstration, we encoded the above-described e-business system [10] in the model checker FDR. FDR has been used in a variety of hardware and software applications, and has recently been applied toward testing for atomicity in two commercial e-commerce payment systems [9]. We illustrate several control features of model checking in the discourse that follows. We intend for the discussion to be nontechnical and use actual code only when it is straightforward; in other instances, we use pseudocode.

A. Data Types and Message Control

We first consider the use of data typing. Data typing allows an analyst to ensure that only specified types of messages are communicated from one trading party to another. Consider the following data-type specifications:

```
--Data types
data type KEY = key
data type PO = po
data type ENCRYPTEDGOODS = eGoodsValid|eGoodsInvalid
data type TOKEN = paymentToken
data type MESSAGE = transAborted
--Channel declarations
channel coutm: {po}
channel moutc: {eGoodsValid, eGoodsInvalid}
channel moutt: {key1}
channel toutc: {key1, eGoodsValid, eGoodsInvalid,
 transAborted}
channel toutm: {paymentToken,
 transAborted }
channel minc: {po}
channel tinc: {paymentToken}
channel cinm: {eGoodsValid, eGoodsInvalid}
channel tinm: {key}
channel cint: {key, eGoodsValid, eGoodsInvalid,
 transAborted}
channel mint: {paymentToken,
 transAborted}
channel coutt: {paymentToken}
```

In this example, all of the data types are simple, except for encrypted goods, which allows two values: *eGoodsValid* (for valid encrypted goods) and *eGoodsInvalid* (for invalid encrypted goods). In general, any finite number of values for a data type can be specified. This useful control is used to specify the messages allowed on various channels as shown above. For instance, the only allowable transmissions on the channel *toutm* (for channel out from TTP to merchant) are *paymentToken* or *transAborted* messages. Any other message by an intruder would be disallowed.

Another example concerns the channel *cint* (for channel in to the customer from the trusted-third party): The only allowable messages are a key, *eGoodsValid*, *eGoodsInvalid*, or *transAborted*. One might ask why *eGoodsInvalid* is allowed since this represents an invalid product. The reason is that it is not known that *eGoodsInvalid* is invalid until the customer checks the product against the description of the ordered product. The remaining channels impose similar restrictions. If the system is expressed in this way, the model checker can later verify whether there are any circumstances under which these constraints can be violated.

B. Ensuring Transaction Completion

Three of the most important requirements of e-business trading are as follows.

1) Money is neither created nor destroyed in the course of an e-commerce transaction. A transaction should ensure the transfer of funds from one party to another without the possibility of the creation or destruction of money. No viable e-business payment method can exist without supporting this property [13], which is commonly known as *money atomicity*.
2) Both the customer and merchant receive evidence that the goods sent (or received) are those to which both parties agreed. This

is of particular importance when dealing with goods that can be transferred electronically (as we do here). This property is commonly known as *valid receipt*. The combination of both 1 and 2 is fundamental [14].

3) The customer must be able to verify that the product about to be received from the merchant is the same as the product that was ordered, before the customer pays for the product [15]. This property is commonly known as *goods atomicity*.

Model checking allows us to verify that our system guarantees these requirements by writing them as specifications in the model checker. FDR returns the results of these large search spaces expeditiously. If a specification is not satisfied, the model checker returns a counterexample that allows an analyst to quickly see how the requirement can be violated. This, in turn, aids in modifying the system to resolve the problem. As an example, we consider a specification for requirement 1 above.

```
Specification requirement_1
    IF NOT transaction_trace = STOP
    OR
        {customer sends paymentToken to TTP;
        merchant receives paymentToken from TTP;
        STOP;}
    OR
        {customer sends paymentToken to TTP;
        customer receives transAborted message from TTP;
        STOP;}
    THEN
        Requirement_1 is violated;
End requirement_1;
```

This specification asserts the following:

1) The customer must send payment to the TTP, and the merchant must subsequently receive that payment from the TTP, or
2) The customer can send payment to the TTP and (in the case where the payment is invalid) the customer then receives a transaction aborted message from the TTP.
3) If neither specificiation 1) nor 2) is satisfied, the requirement is violated.

The following is an example of what the model checker might return as a counterexample in case the requirement fails under some set of conditions.

1) The customer receives the encrypted goods from the TTP and then sends a purchase order to the merchant.
2) The merchant sends the encrypted goods to the customer and sends a key to the TTP, after which the encrypted goods are received by the customer who sends a payment token to the TTP.
3) The TTP then receives the key from the merchant and the payment token from the customer.
4) The TTP sends the payment token to the merchant, after which the customer receives the key from the TTP. *The process then stops*. Since the next step should have been the receipt of the payment token by the merchant (from the TTP), we know where to look to examine the failure.

In the above case, the model checker showed that the failure occurred in the e-process that controls the sending of a decrypting key from the merchant to the TTP. Once this failure was identified, the problem was identified and rectified as shown by a subsequent run of the model checker.

C. Controlling Communication Link Failures

Suppose, however, that unreliability exists on the link from the merchant to the TTP. Such link unreliability could result from technical failure, as well as malicious intrusion by a hacker. Consideration of this contingency can be included in model-checker representation by the following e-process:

```
Procedure comm_mc
    IF product is IN {eGoodsValid, eGoodsInvalid}
    THEN
        CUSTOMER receives product from MERCHANT OR
        PROCEDURE comm_mc;
End comm_mc;
```

This representation means that if all goes well, the transmittal of a payment token from the customer to the TTP will be followed by the receipt of a payment token from the customer. Yet the ***comm_mc*** procedure above allows unspecified delays because each transmission can nondeterministically be followed over and over by the ***comm_mc*** procedure. What happens when we subject our unreliable link to examination by the model checker? It returns the following counterexample.

1) The valid encrypted goods are sent from the TTP and received by the customer.
2) A purchase order is transmitted from the customer and received by the merchant.
3) The merchant sends the valid encrypted goods, which are received by the customer; the merchant also sends a key to the TTP, which is received.
4) The customer sends a payment token to the TTP, which is received.
5) The TTP sends a key to the customer, which is received.
6) The TTP sends a payment to the merchant, *but this payment is not received*.

The model checker has found a failure that violates both requirement 1 and requirement 2. An inspection of the ***requirement_1*** specification repeated below shows that the sending of a payment token from the customer to the TTP must finally be followed by the receipt of that token by the merchant (sent by the TTP) or by the transmittal of a transaction aborted message from the TTP to the customer.

```
Specification requirement_1
    IF NOT transaction_trace = STOP
    OR
        {customer sends paymentToken to TTP;
        merchant receives paymentToken from TTP;
        STOP;}
    OR
        {customer sends paymentToken to TTP;
        customer receives transAborted message from TTP;
        STOP;}
    THEN
        requirement_1 is violated;
End requirement_1;
```

In addition, requirement 2 stipulates that a customer receives the product only after the customer has paid for the product. Since the merchant never received the payment token, no encrypted goods were transmitted from the merchant to the customer. These problems can be resolved by modifying the ***comm_mc*** procedure in the following way:

```
Procedure comm_mc
    IF product is IN {eGoodsValid, eGoodsInvalid}
    THEN
        CUSTOMER receives product from MERCHANT OR
        PROCEDURE time_out;
End comm_mc;
```

The ***time_out*** procedure aborts the transaction if goods are not received within a specified period. Adding this procedure obviates the problem identified above.

There are other areas of possible vulnerability that can be explored in a similar fashion. The point of these examples is to demonstrate that once a requirement is specified, its implementation can readily be verified with a model checker. This is a critical step in designing e-business standards since translating concepts to implementation often leads to error or omission.

V. Discussion

The objective of this paper is to provide an accessible explication and demonstration of the efficiency and effectiveness of using a model checker to evaluate the reliability of transaction protocols within e-business processes. In particular, when many processes are distributed and automated, there are many states within those processes that need to be evaluated. Evidence suggests that such an evaluation is beyond the practical capabilities of manual procedures, theorem provers, and simulation, leaving e-business partners vulnerable to unforeseen failures and corresponding costs and losses. Model checkers do not guarantee discovery of all vulnerabilities; but if designers are able to identify a full range of specifications that a system must satisfy, model checkers provide efficient and effective verification. Notably, model checkers provide counterexamples that define sequences of processes that lead to failure.

We have shown that the key elements of an e-business protocol can be represented using a model checker such as FDR. Since the elements of the e-business protocol that we have incorporated in this study are a subset of the protocols within a large class of e-business applications, there is considerable potential for broader applications. Given current concerns about Internet security, model checkers may have potential in both designing and auditing a variety of system implementations. For example, as web-support services develop the capability for accessing public registries of vendors, both search and subsequent transaction processes may benefit from application of model-checking technology. Model checkers may also prove valuable in examining the processes of highly integrated applications found in enterprise resource-planning (ERP) systems.

A feature of model checking that should be of particular interest to e-business designers is the capability of developing specification statements that must be satisfied by implementation. For demonstration purposes, our presentation has focused on money atomicity, goods atomicity, valid receipt, and communication link failures because these are fundamental to e-business; however, model checking can be used to test a wide variety of protocol features. This approach should help not only systems designers, but also auditors. Systems that are examined by auditors need to have adequate controls built in prior to implementation and will need adequate auditing after implementation to ensure that none of the processes have been corrupted.

Whereas model checking provides a viable formal method for evaluating the security and reliability of e-business processes, some limitations apply. As noted previously, there is little purpose in verifying an e-business system with an incorrect model. This study assumes the model will be correctly defined and coded into a model checker. Model checkers do not build correct models; instead, they help verify the models.

The CSP model-checking language can be complex for practitioners to use, but more accessible approaches are becoming available. For example, a recently developed model-checking language called Casper has overcome CSP's complexity by adding a higher-level language whose interpreter translates code into CSP. Whatever language is used, it is important to emphasize that once a specification is verified it can be continually reused, regardless of the system used to conduct the process. Thus, the up-front investment in process modeling and model checking can yield long-term dividends in terms of system security and reliability as system upgrades and changes are implemented.

Future research could explore issues related to ensuring the accuracy of the model coded into the model checker. Additional studies could also examine ways to educate potential users of e-business systems on features that have been verified and their meaning. Research might also be conducted on comparative performance and efficiency of alternative model checkers. Finally, empirical tests of model-checking principles may help to validate the accuracy and effectiveness of these theories.

VI. Conclusion

E-business is moving forward at a rapid pace across the globe, exposing businesses to e-business processes that rely on interdependencies among customers, merchants, and TTPs. New e-business foci, such as web services and pervasive computing, will extend the reach of e-business, but these may also increase complexity and exposure possibilities. This will necessitate proactive verification of related models. Model checking could become a fundamental tool in this process.

References

[1] W. Janssen, R. Mateescu, S. Mauw, P. Fennema, and P. van der Stappen, "Model checking for managers," in *Proc. 6th Int. SPIN Workshop Practical Aspects Model Checking*, Toulouse, France, Sep. 1999, pp. 92–107.

[2] W. Wang, Z. Hidvegi, A. Bailey, and A. Whinston, "E-process design and assurance using model checking," *IEEE Trans. Comput.*, vol. 33, no. 10, pp. 48–53, Oct. 2000.

[3] F. Coyle, *XML, Web Services, and the Data Revolution*. Reading, MA: Addison-Wesley, 2002.

[4] J. Kopena and W. Regli, "DAMLJessKB: A tool for reasoning with the semantic web," *IEEE Intell. Syst.*, vol. 18, no. 3, pp. 74–77, May/Jun. 2003.

[5] W. Wang, Z. Hidvegi, A. Bailey, and A. Whinston, "Model checking—A rigorous and efficient tool for e-commerce internal control and assurance," in *Working Paper*. Atlanta, GA: Gozuita School Bus., Emory Univ., 2001.

[6] J. Wing, "A symbiotic relationship between formal methods and security," in *Proc. Workshops Computer Security, Fault Tolerance, Software Assurance: From Needs to Solution*, CMU-CS-98-188, 1998, pp. 1–12.

[7] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using FDR," in *Proc. Tools Algorithms Construction Analysis Systems: Second International Workshop*, 1996, pp. 147–166.

[8] A. Roscoe, *The Theory and Practice of Concurrency*. Upper Saddle River, NJ: Prentice-Hall, 1998.

[9] N. Heintze, J. Tygar, J. Wing, and H. Wong, "Model checking electronic commerce protocols," in *Proc. 2nd USENIX Workshop Electronic Commerce*, 1996, pp. 147–165.

[10] I. Ray, I. Ray, and N. Narasimhamurthi, "A fair exchange e-commerce protocol with automated dispute resolution," in *Proc. 14th Annu, IFIP WG 11.3 Working Conf. Database Security*, Schoorl, The Netherlands, Aug. 2000, pp. 27–38.

[11] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. Cambridge, MA: MIT Press, 1999.

[12] P. Ryan and S. Schneider, *The Modeling and Analysis of Security Protocols: The CSP Approach*. Reading, MA: Addison-Wesley, 2001.

[13] H. Schuldt, A. Popovici, and H. J. Schek, "Execution guarantees in electronic commerce payments," in *Proc. 8th Int. Workshop Foundations Models Languages Data Objects*. New York, 1999, pp. 189–198. Lecture Notes in Computer Science.

[14] J. Tygar, "Atomicity versus anonymity: Distributed transactions for electronic commerce," in *Proc. 24th Very Large Data Base Conf.*, New York, 1998, pp. 1–12.

[15] I. Ray and I. Ray, "Fair exchange in e-commerce," in *Proc. Assoc. Computing Machinery SIGcom Exchange*, vol. 3, May 2002, pp. 9–17.