SIMULATION AND VERIFICATION OF ATOMICITY PROPERTIES FOR AN ELECTRONIC CASH SYSTEM

Panagiotis Katsaros
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
e-mail: katsaros@csd.auth.gr

Vasilis Odontidis
Department of Mathematics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece

Maria Gousidou-Koutita
Department of Mathematics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
e-mail: gousidou@math.auth.gr

KEYWORDS
Electronic cash, Coloured Petri Nets, Model checking

ABSTRACT

An electronic cash system as any distributed system is subject to site and communication failures, as well as potential security attacks. This work focuses on simulation and verification of three important correctness properties, for ensuring failure resilience or detecting potential property violation scenarios. We refer to the NetBill electronic cash system and the properties to be checked are money atomicity, goods atomicity and certified delivery. We introduce a Colored Petri Net that models NetBill, in the presence of site or communication failures and all possible transaction abort cases. The presented model has been implemented in CPN Tools, a graphical ML-based tool for analyzing Colored Petri Nets. This allows us to combine the provided state space exploration functions and the supported Computation Tree like temporal logic (CTL), for model checking the forenamed atomicity properties. At the same time, it is possible to exploit the provided interactive simulation facilities to explore potential property violation scenarios and correct the protocol's design.

INTRODUCTION

NetBill (Cox et al. 1995) is an electronic cash protocol for selling and delivering low-priced network goods. As in all electronic cash systems, NetBill is required: (i) not to create money in the presence of site and/or communication failures, (ii) the merchant to receive payment if and only if the consumer receives the ordered goods and (iii) both the merchant and the consumer to be able to give non-repudiable proof of the contents of the delivered goods. The first mentioned property is called money atomicity, the second is known as goods atomicity and the last one is called certified delivery.

The already published model checking approaches (Heintze et al. 1996, Ray and Ray 2000, Ray et al. 2005) use a Communicating Sequential Processes (CSP) based system representation (`SYSTEM`) and verification is performed by the Failure Divergence Refinement (FDR) model checker. The properties to be checked are expressed as CSP processes (`SPEC`) and FDR checks if the set of behaviors generated by `SYSTEM` is a subset of those generated by `SPEC`. An interesting result is the fact that the Digicash electronic cash system (Chaum et al. 1990) does not satisfy all forenamed properties in the presence of failures, as opossed to NetBill.

In this work, we propose a Colored Petri Net (CP-net) that models NetBill, in the presence of site or communication failures and all possible transaction abort cases. The properties of interest are expressed as CTL-based temporal logic formulae (Clarke et al. 1986), which are model checked over the generated model state space.

Compared to the already published modeling approaches, CP-nets provide an explicit representation of both model states and actions. Our model has been implemented in CPN Tools (Jensen 1998), a graphical ML-based tool for editing and analyzing CP-nets. This makes feasible to exploit the tool's interactive simulation facilities in order to explore potential property violation scenarios and correct the protocol's design.

CP-nets have been also used in the formal analysis of the Internet Open Trading Protocol (Ouyang and Billington 2004). In that work the authors verify valid transaction termination (absence of deadlocks), absence of livelocks and absence of unexpected dead transitions. Their analysis approach is based on inspection of the model's terminal states (markings) and the corresponding strongly connected components (SCC) graph[1].

The properties considered in our work are not related to structural properties of the generated state space graph, as for example the absence of self-loops. To the best of our knowledge, our work is a first attempt to apply CTL-based model checking, in order to ensure a set of functional properties (money atomicity, goods atomicity and certified delivery) over a CP-net representation of an e-commerce problem.

Given the advanced tool support of CPN Tools, the results published in (Ouyang and Billington 2004) and the ones provided in this article, we believe that CP-nets is an attractive alternative over the CSP-based modeling approach used in e-commerce problems: it possesses the comparative advantage of interactively simulating the actions executed in a property violation scenario, in order to correct the protocol's design.

Next section describes the implemented CP-net for the considered electronic cash system. Then, we proceed to the proposed model checking of the required functional properties. The paper concludes with a summary of the achieved results and an eye on interesting future research prospects.

A COLORED PETRI NET MODEL FOR NETBILL

CPN Tools

In CPN Tools, CP-nets are developed in a modern graphical environment that provides interactive feedback for the model's behavior through simulation. Colors, variables, function declarations and net inscriptions are written in CPN ML, which is an extension of Standard ML and for this reason

[1] A strongly connected component (SCC) of the state space is a maximal sub-graph whose nodes are mutually reachable from each other.

employs a functional programming style. In CPN Tools it is possible to use simple as well as compound color sets such as products, records, lists and union color sets.
The toolset provides the necessary functionality for the analysis of simple and timed CP-nets specified in a number of hierarchically related pages. Typical models consist of 10-100 pages with varying complexity and programming requirements. The companion state space tool allows the generation of the entire or a portion of the model's state space (occurrence graph) and the performance of standard as well as non-standard analysis queries.
The standard properties that characterize the behavior of a CP-net in terms of the model's structure include:

- bounds-related properties that characterize the model in terms of the number of tokens we may have at the places of interest,
- home properties that provide information about markings or sets of markings to which it is always possible to return,
- liveness properties that are used to examine whether a set of binding elements *X* remains active and
- fairness properties that provide information about how often the different binding elements occur.

The NetBill electronic cash system

The NetBill transaction model involves three parties: the consumer (*C*), the merchant (*M*) and the bank server (*B*). We use the notation "*X* ⇒ *Y message*" to indicate that *X* sends the specified message to *Y*. The basic protocol consists of the following messages:

1. $C \Rightarrow M$ Price request
2. $M \Rightarrow C$ Price quote
3. $C \Rightarrow M$ Goods request
4. $M \Rightarrow C$ Goods, encrypted with a key *K*
5. $C \Rightarrow M$ Signed Electronic Payment Order (*epo*)
6. $M \Rightarrow B$ Endorsed Electronic Payment Order (including the key *K*)
7. $B \Rightarrow M$ Signed result (including *K* in case of successful payment)
8. $M \Rightarrow C$ Signed result (including *K* in case of successful payment)

A transaction involves three phases: price negotiation, goods delivery and payment. We consider the selling of information goods, in which case the NetBill protocol, links goods delivery and payment into a single atomic transaction. The consumer and the merchant interact with each other in the first two phases:

- *C* issues a price request for a particular product (1) and *M* replies with the requested price (2),
- *C* either aborts the transaction or issues a goods request to the merchant (3),
- in the second case, *M* delivers the requested goods encrypted with a key *K* (4).

The goods are cryptographically checksummed so that *C* or *M* can prove that the requested goods are not affected by a potential transmission error and are not altered by a protocol participant. The bank (*B*) is not involved until the payment phase:

- *C* sends to the merchant a signed electronic payment order (5) including all necessary payment details and the received product checksum,
- *M* validates the received electronic payment order and checksum information and either aborts the transaction or endorses it by sending the received payment order and the associated decryption key *K* to *B* (6),
- *B* responds to *M* (7) with the payment result and the decryption key *K* (in case of successful payment) and the transmitted information is finally forwarded to *C* (8) to end the transaction.

NetBill provides protection to *C* against fraud by *M* in the following ways:

- the key *K*, which is needed to decrypt the goods is registered with *B* and if *M* does not respond in a successful payment as expected (possibly because of a site or a communication failure), the consumer asks the key from *B*, that in fact acts as a trusted third party,
- if there is a discrepancy between what *C* ordered and what *M* delivered, *C* can easily demonstrate this discrepancy to the trusted third party, since the payment order received by *B* includes all details about what exactly was ordered, the amount charged, the key *K* reported by *M* and the checksum of the delivered encrypted goods.

Thus, if the goods are faulty it is easy to demonstrate that the problem lies with the goods as sent and not with any subsequent alteration. When as in our case the protocol assures that both *C* and *M* can give non-repudiable proof of the contents of the delivered goods, we say that the protocol possess the property of certified delivery.

The proposed CP-net

In this section, we introduce a CP-net for the NetBill electronic cash system. The adopted modeling assumptions take into account consumer and merchant site failures and non-reliable communication between the protocol's participants, including potential message losses. We assume that both, consumer's account debit and merchant's account credit take place at the same site (bank server) that provides trivial transaction atomicity guarantees. Thus, we omit modeling bank site failures, since this would burden the CP-net with details that are not part of the NetBill protocol, but concern the provided transaction processing mechanism. This implies the property of money atomicity, but we show how to express it by exploiting the provided state space exploration functions and the supported Computation Tree like temporal logic (CTL).
The proposed CP-net consists of a number of hierarchically related pages, which model the protocol's behavior in different levels of abstraction.
In the highest level of abstraction (Figure 1), we model the protocol's participants and message exchanges. We omit the protocol steps 1 and 2, since they are not significant for checking payment atomicity. Figure 2 summarizes the color and variable declarations used for the transition and arc

inscriptions of the described CP-net. Our model is important to reflect all possible protocol execution scenarios. We adopt a compact representation of all distinct transaction abort cases by specifying them as `request` typed tokens that encode the following execution scenarios:

1. *C* sends to *M* a valid goods request (`gReq=v`)
2. *C* sends to *M* an invalid goods request (`gReq=v`)
3. the encrypted goods received by *C* are the requested ones (`enGoods=v`)
4. the encrypted goods received by *C* are affected by an occurred transmission error (`enGoods=i`)
5. *C* sends to *M* a valid electronic payment order (`epoReq=v`)
6. *C* sends to *M* an invalid electronic payment order (`epoReq=i`)

An electronic payment order (*epo*) is not valid, when it is not signed or includes invalid payment details, like for example a product checksum that does not coincide to the appropriate diagnostic string at the output place `stop`. Unilateral transaction aborts are also represented by transitions like the one called `errorEnGoods`, which correspond to the validation actions performed by the protocol participants. Dispatch of the signed *epo* by *C* signifies the commitment of *C* to the executed transaction request.

The diagnostic strings placed at the place `stop` indicate protocol termination, but only `"No Funds"` and `"Success"` are reported to the end user.

In case of site or communication failure, *C* is informed for the transaction result by querying *B*.

Due to space limitations, we omit the details of the `MerchantProcess` transition of the high-level model. The main point considered in the design of the `BankProcess` transition (Figure 4) is the money transfer transaction to not be interleaved with queries that are potentially made by *C*.

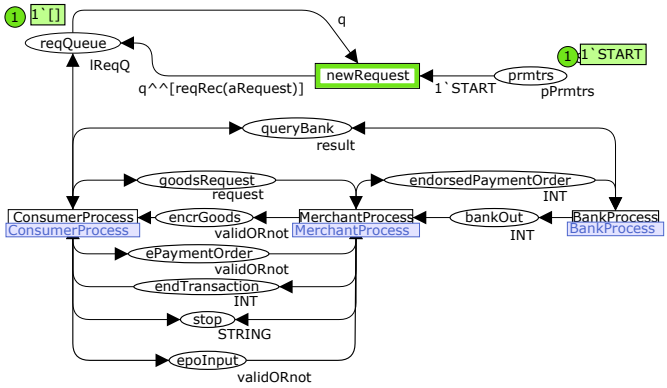

Figure 1: Top-level structure of the NetBill CP-net

```
colset pPrmtrs          =with START;
colset validORnot       =with v | i;
colset request          =record gReq:validORnot*enGoods:validORnot*epoReq:validORnot;
colset sRequest         =union reqRec:request;
colset lReqQ            =list sRequest;
colset result           =with noFunds | paymentReceipt | noRecord;
var aRequest            :request;
var q                   :lReqQ;
var valCode             :validORnot;
var intVar              :INT;
var res                 :result;
```

Figure 2: Color sets and variables used in the NetBill CP-net

one assigned to the encrypted goods sent to *C*.

Figure 3 introduces the consumer process page that corresponds to the `ConsumerProcess` substitution transition of Figure 1. Input and output places were assigned to the synonyms shown in the top-level CP-net. Firing of transition `gRequest` places the `result` typed token `noRecord` at the place `queryBank`.

This models the possibility of *C* querying *B* (trusted third party) for the result of the ongoing transaction. The `request` typed token `aRequest` is passed to the place `goodsRequest` and it is then used non-deterministically to fire either the `commErrCtoM1` transition (communication error: C to M) or the one corresponding to its reception by the merchant process (merchant process page).

We note that *C* can abort the executed transaction any time up to the submission of *epo*. Potential unilateral abort decisions and consumer site failures are modeled by transitions named as `abort#` and terminate the protocol by placing an

MODEL CHECKING THE REQUIRED ATOMICITY PROPERTIES

The results for the properties of the standard report of the CPN Tools state space analysis constitute a necessary input source, for correctly expressing the CTL formulae of the required correctness properties. The standard report for the described NetBill CP-net are based on a state space graph with 59 nodes and 103 arcs with no home markings and no dead and live transitions insrances. There are no infinite occurrence sequences and the protocol terminates in one of the 13 dead markings, with node numbers that are easily found by the provided state space exploration functions.

The ML-functions used in model checking the required atomicity properties are summarized in Table 1.

Function `SearchNodes` is used to detect the marking(s) right after the occurrence of a particular event, like for example a money transfer from *C*'s account to *M*'s account.

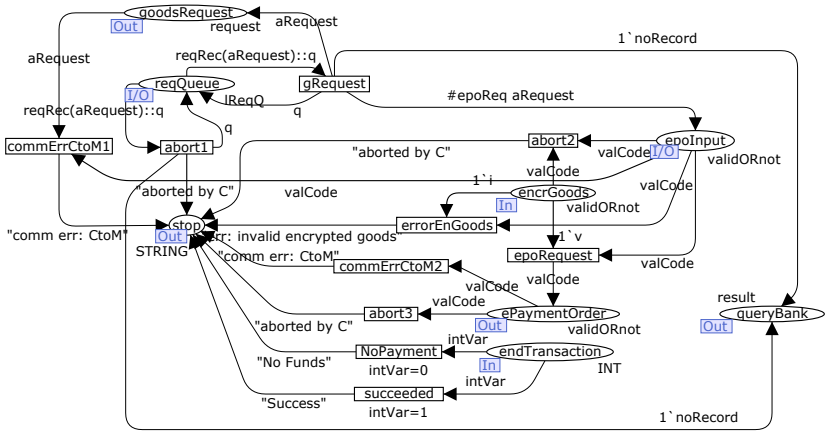

Figure 3: The Consumer (*C*) process page of the NetBill CP-net

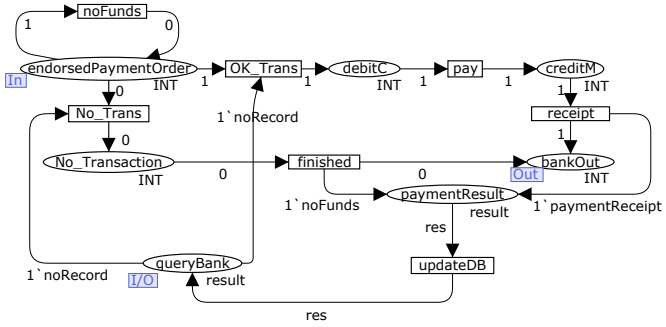

Figure 4: The Bank (*B*) process page of the NetBill CP-net

Table 2 summarizes the CTL formulae used to express the required properties in terms of paths over the generated state space graph. A path is a sequence of states and transition occurrences, that is, a walk-through of the state space constrained by the direction of existing arcs. A path may be infinite, but for the described NetBill CP-net, due to the absence of infinite occurrence sequences (state space report), there are no infinite paths. A CTL expression that corresponds to the required property is model checked by the `eval_node` function, starting from the node number that is passed as second argument.

Table 1: State space querying functions

function description	use
`Mark.<PageName>'<PlaceName> N M`	Returns the set of tokens positioned on place <PlaceName> on the Nth instance of page <PageName> in the marking M
`SearchNodes ( <search area>, <predicate function>, <search limit>, <evaluation function>, <start value>, <combination function>)`	Traverses the nodes of the part of the occurrence graph specified in <search area>. At each node the calculation specified by <evaluation function> is performed and the results of these calculations are combined as specified by <combination function> to form the final result. The <predicate function> maps each node into a boolean value and selects only those nodes, which evaluate to true. We use the value EntireGraph for <search area> to denote the set of all nodes in the occurrence graph and the value NoLimit for <search limit> to continue searching for all nodes, for which the predicate function evaluates to true.
`List.nth(l,n)`	Returns the nth element in list l, where 0 <= n < length l.

Table 2: CTL state formulae operators and model checking functions

state formulae syntax	meaning
`NOT(A)`	Boolean value that corresponds to the negation of A, where A is a CTL formula.
`AND(`A_1`,`A_2`)`	This formula is true if both A_1 and A_2 are true.
`NF(<message>,<node function>)`	A function that is typically used for identifying single states or a subset of the state space. Its arguments are a string and a function, which takes a state space node and returns a boolean. The string is used when a CTL formula evaluates to false in the model checker.
`EV(A)≡FORALL_UNTIL(TT,A)`	This formula is true if the argument A becomes true eventually (within a finite number of steps) starting from the state we are now. TT denotes the true constant value.
`ALONG(A)≡NOT(EV(NOT(A)))`	This formula is true if there exists a path for which the argument A holds for every state. The path is either infinite or ends in a dead state.
`POS(A)≡EXIST_UNTIL(TT,A)`	This formula is true if possible from the state we are now, to reach a state where the argument A is true.
`EXIST_NEXT(A)`	This formula is true iff there exists an immediate successor state, from where we are now, in which the argument A is true.
`FORALL_NEXT(A)`	This formula is true iff for all immediate successor states from where we are now the argument A is true.
`eval_node <formula> <node>`	The standard model checking function that takes two arguments: the CTL formula to be checked and a state from where the model checking should start.

Figure 5 shows the model checking of the money atomicity property. We are interested to verify that money transfer takes place atomically that is, for all paths starting from the occurrence of the consumer's debit, the protocol performs the corresponding credit to the merchant's account irrespective of the considered failure possibilities.

Value `firstdebitState` corresponds to the marking that signifies consumer's debit. This is the marking from where the model checking starts. Value `noDebit` is used to detect redundant debits before the occurrence of the expected credit. Value `moneyAtomicity` (`true`) ensures that for all immediate successors, `noDebit` is true for each state along the path, until the last state on the path, where `creditState` becomes `true`.

Figure 6 shows the model checking of non-atomic goods delivery. We are interested to verify that irrespective of the considered failures and unilateral abort decisions (i) when *C* signs a valid *epo* it is not possible to eventually perform *C*'s debit, without a subsequent protocol termination with a registered payment receipt (including the required decryption key) and (ii) when *C* sends a not necessarily valid *epo* it is not possible to eventually register a payment receipt, without having previously debited *C*'s account. The first mentioned guarantee ensures goods atomicity from the consumer's perspective and the second mentioned guarantee ensures goods atomicity from the merchant perspective.

Value `dispatchedEPOState` corresponds to the marking that signifies the dispatch of a valid, signed *epo*. This is the marking from where the model checking of (i) starts. Value `debitState` is used to detect *C*'s debit. Value `notRegisteredDecrKey` is used to check the absence of a payment receipt. Value `noGoodsAtomicityA` (`false`) ensures that there is no path, for which it is possible to eventually occur *C*'s debit and at the same time in every state, to not register the expected payment receipt. Note that because of the absence of infinite paths (state space report), all paths quantified by `ALONG` end in a dead marking (protocol termination).

Value `dispatchedEPOState1` corresponds to the marking that signifies the dispatch of a valid signed *epo*. On the other hand, value `dispatchedEPOState2` corresponds to the marking that signifies the dispatch of an invalid *epo*. These are the markings from where the model checking of (ii) starts. Value `noDebitFound` is used to check the absence of *C*'s debit. Value `registeredDecrKey` is used to detect registration of an unexpected payment receipt. Value `noGoodsAtomicityB` (`false` in both model checking cases) ensures that there is no path, for which it is possible to eventually register a payment receipt and at the same time in every state, to not have performed *C*'s debit.

Figure 7 shows the model checking of a non-certified delivery. We are interested to verify that irrespective of the considered failures and unilateral abort decisions, the protocol does not fall in a state, where it is possible to end with a payment receipt, without *C* having previously obtained an encrypted version of the requested goods and the corresponding checksum number. The obtained checksum number can be used to prove potential discrepancy between what *C* ordered and what *M* delivered (if the number coincides with the checksum number written on the registered payment receipt).

```
fun debitDone n = (Mark.BankProcess'debitC 1 n = [1]);
val firstDebitState = List.nth(SearchNodes (
                        EntireGraph,
                        fn n => (debitDone n),
                        NoLimit,
                        fn n => n,
                        [],
                        op ::),0);
fun creditDone n = (Mark.BankProcess'queryBank 1 n = [paymentReceipt]);
val noDebit = NOT(NF("Double debit!",debitDone));
val creditState = NF("No credit!",creditDone);
val moneyAtomicity = FORALL_NEXT(FORALL_UNTIL(noDebit,creditState));
eval_node moneyAtomicity firstDebitState;
```

Figure 5: Model checking money atomicity: true

```
fun signedEPO n = (Mark.Protocol'ePaymentOrder 1 n = [v]);
val dispatchedEPOState = List.nth(SearchNodes (
        EntireGraph,
        fn n => (signedEPO n),
        NoLimit,
        fn n => n,
        [],
        op ::),0);
fun debitDone n = (Mark.BankProcess'debitC 1 n = [1]);
fun noTrans n = (Mark.Protocol'queryBank 1 n <>
                 [paymentReceipt]);
val debitState = NF("No debit!",debitDone);
val notRegisteredDecrKey = NF("Found decryption key!",
                           noTrans);
val noGoodsAtomicityA = ALONG(AND(EV(debitState),
                        notRegisteredDecrKey));
eval_node noGoodsAtomicityA dispatchedEPOState;
```

```
fun sendEPO n = (Mark.Protocol'ePaymentOrder 1 n <> []);
val dispatchedEPOStates = SearchNodes (
        EntireGraph,
        fn n => (sendEPO n),
        NoLimit,
        fn n => n,
        [],
        op ::);
val dispatchedEPOState1 = List.nth(dispatchedEPOStates,0);
val dispatchedEPOState2 = List.nth(dispatchedEPOStates,1);
fun noDebitDone n = (Mark.BankProcess'debitC 1 n <> [1]);
fun succeedTrans n = (Mark.Protocol'queryBank 1 n =
                      [paymentReceipt]);
val noDebitFound = NF("Debit found!",noDebitDone);
val registeredDecrKey = NF("Failed transaction!",
                        succeedTrans);
val noGoodsAtomicityB = ALONG(AND(EV(registeredDecrKey),
                        noDebitFound));
eval_node noGoodsAtomicityB dispatchedEPOState1;
eval_node noGoodsAtomicityB dispatchedEPOState2;
```

Figure 6: Model checking the two parts of the non-atomic goods delivery: false

```
fun registerKeyState n = (Mark.Protocol'queryBank 1 n = [paymentReceipt]);
val registerKey = POS(EV(NF("No paymentReceipt!",registerKeyState)));
fun enGoodsTransferedState n = (Mark.Protocol'encrGoods 1 n = [v]);
val noGoods = NOT(POS(EV(NF("Encr goods sent!",enGoodsTransferedState))));
val nonCertifiedDelivery = EXIST_NEXT(AND(noGoods,registerKey));
eval_node nonCertifiedDelivery InitNode;
```

Figure 7: Model checking non-certified delivery: false

Table 3: Protocol termination inspection

marking (N)	Mark.Protocol'stop 1 N	Mark.Protocol'queryBank 1 N	interpretation
59	[“No Funds”]	[noFunds]	No failures.
58	[“comm err: MtoC”]	[noFunds]	Communication failure: M fails to report the transaction result to C. C is informed for the result of the submitted transaction by querying B.
57	[“Success”]	[paymentReceipt]	No failures.
56	[“comm err: MtoC”]	[paymentReceipt]	Communication failure: M fails to report the transaction result to C. C obtains the product decryption key by querying B.
55	[“comm err: BtoM or M site failure”]	[noFunds]	M is not informed for the result of the submitted transaction due to a potential site/ communication failure. C is informed for the result of the submitted transaction by querying B.
51	[“comm err: BtoM or M site failure”]	[paymentReceipt]	M is not informed for the result of the submitted transaction due to a potential site or communication failure. C obtains the product decryption key by querying B.
35	[“comm err: MtoB”]	[noRecord]	Communication failure: the signed payment order is not transmitted to B. C is informed that there is no transaction by querying B.
33	[“err: invalid EPO”]	[noRecord]	M aborts the transaction due to an invalid *epo*. C is informed that there is no transaction by querying B.
30	[“err: invalid encrypted goods”]	[noRecord]	C aborts the transaction due to reception of encrypted goods that are possibly affected by an occurred transmission error.
21	[“comm err: CtoM”]	[noRecord]	Communication failure: the goods request or the signed payment order is not transmitted to M. C is informed that there is no transaction by querying B.
20	[“err: invalid goods reque”]	[noRecord]	M aborts the transaction due to an invalid goods request.
19	[“aborted by M”]	[noRecord]	M aborts the transaction due to a potential site failure or due to a unilateral abort decision. C is informed that there is no transaction by querying B.
10	[“aborted by C”]	[noRecord]	C aborts the transaction due to a potential site failure or due to a unilateral abort decision before being committed to it, by the dispatch of a signed payment order.

Value `registerKey` is used to detect registration of the expected payment receipt. Value `noGoods` is used to check the absence of an encrypted goods delivery. Value `nonCertiefiedDelivery` (`false`), when it is model checked starting from the initial node, ensures that there is no path for which it is possible to not have delivered the assumed encrypted goods and the protocol to terminate with having registered a payment receipt.

Model checking certified delivery from the merchant perspective is performed in a similar way.

Protocol failure analysis

Protocol failure analysis aims in exploring all property violation scenarios and pinpoints areas where design changes or revisions should be considered. Having shown that CP-net based model checking of payment atomicity is feasible, we can then exploit the CPN Tools advanced graphical environment, to interactively simulating the actions performed in possible property violation scenarios. In what is concerned with NetBill, protocol failure analysis is not applicable, since we did not detect atomicity violation cases. However, we proceed to the inspection of the CP-net’s terminal markings (Table 4) by interactively simulating the presented CP-net.

CONCLUSION

In this paper, we have illustrated how Colored Petri Net model checking can be used to prove that an e-commerce protocol satisfies the properties of money atomicity, goods atomicity and certified delivery. The proposed protocol failure analysis exploits the advanced interactive simulation features of CPN Tools to explore potential property violation scenarios and to pinpoint areas, where design changes or revisions should be considered.

The described approach can also be applied for model checking other properties relevant to the design of e-commerce protocols, like for example anonymity, security (Panti et al. 2002), personalization potentiality (Georgiadis and Manitsaris 2005), information flow security (Katsaros 2005) etc.

ACKNOWLEDGMENTS

We acknowledge the CPN Tools team at Aarhus University, Denmark for kindly providing us the license of use of the valuable CP-net toolset.

REFERENCES

Chaum, D., Fiat, A. and M. Naor. 1990. “Untraceable electronic cash”, In *Proc. CRYPTO’88*, Springer-Verlag, 200-212

Clarke, E. M., Emerson, E. A. and A. P. Sistle. 1986. “Automatic verification of finite state concurrent system using temporal logic”, *ACM Transactions on Programming Languages and Systems* 8, No. 2, 244-263

Cox, B., Tygar, J. D. and M. Sirbu. 1995. “NetBill security and transaction protocol”, In: *Proc. 1st USENIX Workshop in Electronic Commerce*, New York, California, 77-88

Georgiadis, C. K. and A. Manitsaris. 2005. “Personalization in Mobile Commerce Environments: Multimedia Challenges”, *Cutter IT Journal* 18, No. 8, 36-43

Heintze, N., Tygar, J., Wing, J. and H. Wong. 1996. “Model checking electronic commerce protocols”, In: *Proc. 2nd USENIX Workshop in Electronic Commerce*, Oakland, CA, 146-164

Jensen, K. 1998. “An introduction to the practical use of colored Petri Nets”, In: *Lectures on Petri Nets II: Applications*, LNCS 1492, Springer-Verlag, 237-292

Katsaros, P. 2005. “On the design of access control to prevent sensitive information leakage in distributed object systems: a Colored Petri Net model”, In: *Proc. CoopIS/DOA/ODBASE*, LNCS 3761, Springer-Verlag, 945-962

Ouyang, C. and J. Billington. 2004. “Formal analysis of the Internet Open Trading Proocol”, In: Proc. FORTE 2004 Workshops, LNCS 3236, Springer-Verlag, 1-15

Panti, M., Spalazzi, L. and S. Tacconi. 2002. “Verification of security properties in electronic payment protocols”, In: *Proc. ACM SIGPLAN Workshop on Issues in the Theory of Security*, Oregon

Ray, I. and I. Ray. 2000. “Failure Analysis of an E-Commerce Protocol using Model Checking”, In: *Proc. 2nd Int. Workshop on Advanced Issues of E-Commerce and Web-based Information Systems*, San Jose, CA, 176-183

Ray, I., Ray, I. and N. Natarajan. 2005. “An anonymous and failure resilient fair-exchange e-commerce protocol”, *Decision Support Systems* 39, 267-292