Colored Petri Net based model checking and failure analysis for E-commerce protocols

Panagiotis Katsaros[1] Vasilis Odontidis[2] Maria Gousidou-Koutita[2]

[1] Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
katsaros@csd.auth.gr
http://delab.csd.auth.gr/~katsaros/index.html

[2] Department of Mathematics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
gousidou@math.auth.gr

Abstract. We present a Colored Petri Net approach to model check three atomicity properties for the NetBill electronic cash system. We verify that the protocol satisfies money atomicity, goods atomicity and certified delivery in the presence of potential site or communication failures and all possible unilateral transaction abort cases. Model checking is performed in CPN Tools, a graphical ML-based tool for editing and analyzing Colored Petri Nets (CP-nets). In case of property violation, protocol failure analysis aims in exploring all property violation scenarios, in order to correct the protocol's design. Model checking exploits the provided state space exploration functions and the supported Computation Tree like temporal logic (CTL). On the other hand, protocol failure analysis is performed by inspection of appropriately selected markings and if necessary, by interactively simulating certain property violation scenarios. In e-commerce, Colored Petri Net model checking has been used in verifying absence of deadlocks, absence of livelocks and absence of unexpected dead transitions, as well as in verifying a protocol against its service. To the best of our knowledge, our work is the first attempt to employ CP-nets for model checking atomicity properties. We believe that the described approach can also be applied in model checking other functional properties that are not directly related to the structural properties of the generated state space graph.

1 Introduction

E-commerce systems offer considerable potential, but they are accompanied by a broad range of often unprecedented risks. A lack of system security or reliability can cause the system to behave differently than its stakeholders expect and can lead to loss of physical assets, digital assets, money, and consumer confidence. Model checking refers to a set of mechanized techniques, which are used to automatically discover any scenarios, in which the actual system behavior and the behavior of the stakeholders' model diverge from one another. These scenarios identify potential failures and pinpoint areas where design changes or revisions should be considered.

In e-commerce, most published model checking approaches ([10], [16], [23], [24]) use a Communicating Sequential Processes (CSP) system description (`SYSTEM`) and verification is performed by the Failure Divergence Refinement (FDR) model checker. The properties to be checked are also expressed as CSP processes (`SPEC`) and FDR checks if the set of behaviors generated by `SYSTEM` is a subset of those generated by `SPEC`.

In the field of e-commerce, CP-nets ([12]) and CPN Tools ([7]) have been used to model check the absence of deadlocks and livelocks and the absence of unexpected dead transitions and inconsistent terminal states, for the Internet Open Trading Protocol ([18], [20], [21]). Also, CP-nets have been used in verifying a protocol against its service ([19]). The correctness properties considered in [21] depend on certain structural properties of the state space graph, like for example the absence of self-loops.

However, in e-commerce we are also interested to model check functional properties that are not directly related to the structural properties of the state space graph. CSP-based model checking is broadly used due to its success in verifying security properties, like confidentiality ([22], [26]), message authentication ([26]), anonymity ([26]), integrity ([22], [26]) and information flow security ([14]). CSP-based model checking can also be applied in verifying validated receipt ([23], [24]), accountability ([13]), personalization potentiality ([9]) and atomicity properties, like withdrawal atomicity, payment atomicity and deposit atomicity ([15], [27], [29]), in the presence of site or communication failures and all possible unilateral transaction abort cases.

In this work, we use CPN Tools to model check payment atomicity for the NetBill ([6]) electronic cash system. We show how to exploit the provided state space exploration functions and the supported Computation Tree like temporal logic (CTL), in order to verify ([4], [5]) three different levels of payment atomicity (money atomicity, goods atomicity and certified delivery). In case of property violation, we propose protocol failure analysis to be based on inspection of appropriately selected markings and if necessary, to exploit the CPN Tools advanced graphical environment to interactively simulate the actions performed in one or more property violation scenarios.

Although it was already known that NetBill possesses the three levels of payment atomicity ([10]), we are not aware of published results that model check certified delivery. Moreover, the applied CP-net approach is characterized by the comparative advantage of interactively simulating the developed model and makes CPN Tools an attractive alternative (over CSP-based model checking), for model checking the forenamed security and reliability properties. Finally, the proposed protocol failure analysis is possible to be applied to other electronic cash systems, like for example Digicash ([2], [3]), Payword ([25]), MicroMint ([25]) and MiniPay ([11]), which fail to provide a certified delivery mechanism, for selling and delivering digital goods and services.

Section 2 provides a compact description of the NetBill electronic cash system and the three levels of payment atomicity. Section 3 introduces the adopted modeling assumptions and the proposed CP-net. Section 4 focuses on model checking the three levels of payment atomicity (money atomicity, goods atomicity and certified delivery). Section 5 introduces the proposed protocol failure analysis and the paper concludes with a summary of our CP-net model checking experience and its potential impact.

2 The NetBill electronic cash system

The NetBill transaction model involves three participants: the consumer (*C*), the merchant (*M*) and the bank server (*B*). 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. We use the notation "$X \Rightarrow Y$ *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)

C and *M* interact with each other in the following way:

- *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 in order to be able to confirm that received goods are not affected by a potential transmission error and that they are not subsequently altered. The bank (*B*) is not involved until the payment phase:

- *C* sends to *M* a signed electronic payment order (5) including all necessary payment details and the received product checksum,
- *M* validates the received electronic payment order (*epo*) 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), which are finally forwarded to *C* (8) to terminate the transaction.

NetBill provides protection for *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 valid payment as expected, 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 (that would produce different checksum information).

NetBill is one of the first electronic cash systems that provides the three levels of payment atomicity, in the presence of potential site or communication failures and all possible unilateral transaction abort cases.

Money atomicity is the basic level that should be ensured by all transactions exchanging electronic cash. It means that the payment transaction ensures that there is no possibility of creation or destruction of money, while electronic cash is being transferred.

Goods atomicity ensures money atomicity and also ensures that there is no possibility of paying without receiving goods or vice versa.

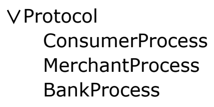
Most electronic payment systems ensure money and goods atomicity, but fail to ensure certified delivery, which is the highest level of atomicity. Certified delivery includes both money and goods atomicity and also allows both participants to prove the details of the transaction. In the related bibliography, certified delivery is also mentioned as non-repudiation of transactions ([28]) or strongly fair exchange ([1]).

3 A Colored Petri Net model for NetBill

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).

Compared to the CP-net proposed for the Internet Open Trading Protocol ([17]) our model differs in how the protocol's participants are represented. In our model, the participant processes correspond to substitution transitions that include potential site and communication failures and all possible unilateral transaction abort cases. In [17], the participants are represented by places, whose color sets include token colors that correspond to all possible participant states. Also, we do not use places that implement reliable FIFO communication channels between the protocol participants. In our case, a protocol message loss terminates the corresponding protocol session. We aim in model checking the forenamed payment atomicity properties that are not directly related to the structural properties of the generated state space graph. On the other hand, the CP-net of [17] has been used in verifying the protocol against its service and in model checking correctness properties that depend on certain structural properties of the model's state space graph.

The proposed CP-net consists of a number of hierarchically related pages (Figure 1), which model the protocol's behavior in different levels of abstraction.



```
∨Protocol
    ConsumerProcess
    MerchantProcess
    BankProcess
```

Fig. 1. The hierarchy page of the NetBill CP-net

In the highest level of abstraction (Figure 2), 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 3 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=i`)
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 data 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`)

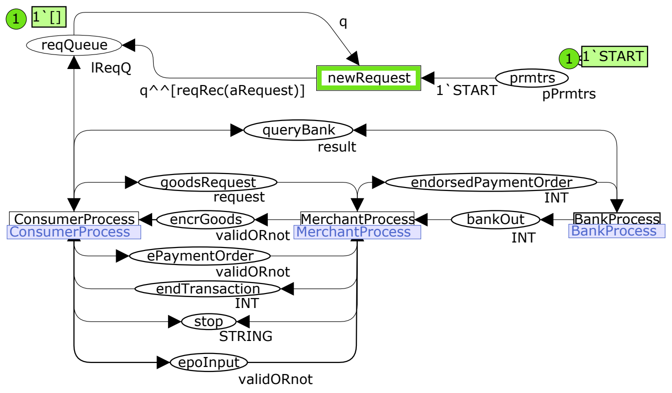


Fig. 2. Top-level structure of the NetBill CP-net

```
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;
```

Fig. 3. Color sets and variables used in the NetBill CP-net

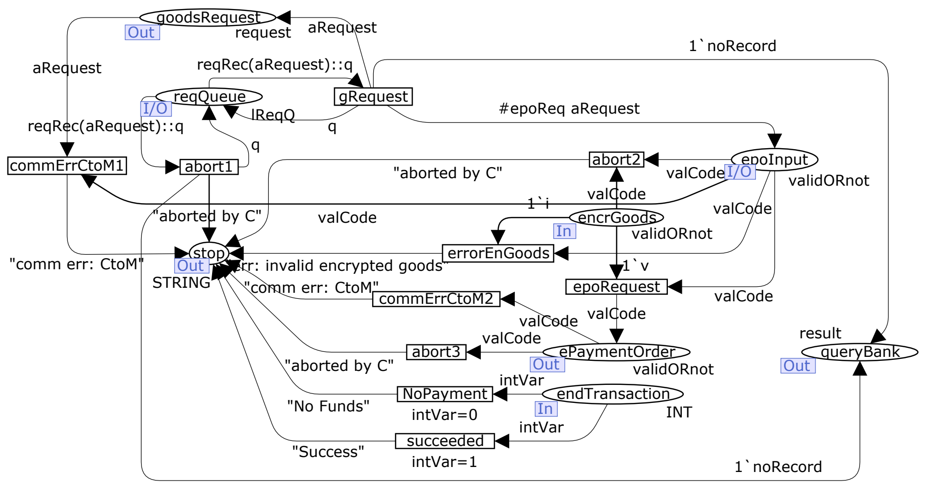


Fig. 4. The Consumer (*C*) process page of the NetBill CP-net

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

Figure 4 introduces the consumer process page that corresponds to the `Consumer-Process` substitution transition of Figure 2. 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 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*.

Figure 5 introduces the merchant process page that corresponds to the `Merchant-Process` substitution transition of Figure 2. On reception of a `request` typed token at the place `goodsRequest` the merchant process either aborts the transaction (site failure), terminates the protocol (because of an invalid goods request) or proceeds to the processing of the received request (transition `receiveGoodsReq`).

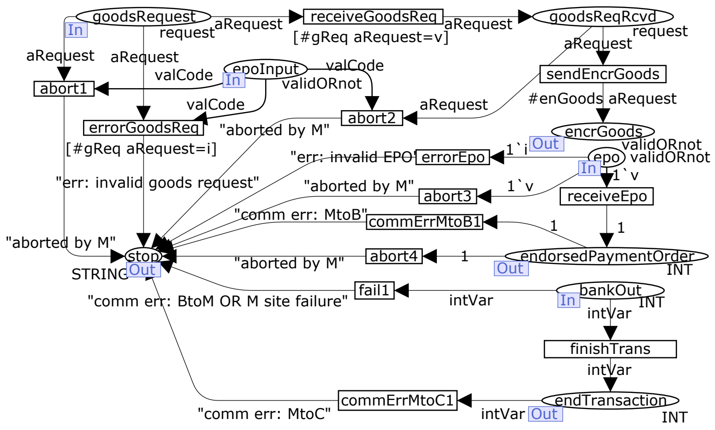


Fig. 5. The Merchant (*M*) process page of the NetBill CP-net

The merchant process responds to the consumer process by dispatching an encrypted version of the ordered goods (transition `sendEncrGoods`). On reception of the signed *epo* at the place `epo`, the merchant process either terminates the transaction in case of invalid *epo*, aborts the transaction due to a site failure (transition `abort3`) or endorses the received *epo* (attaches the required decryption key) and forwards it to the bank server process through the place `endorsedPaymentOrder`.

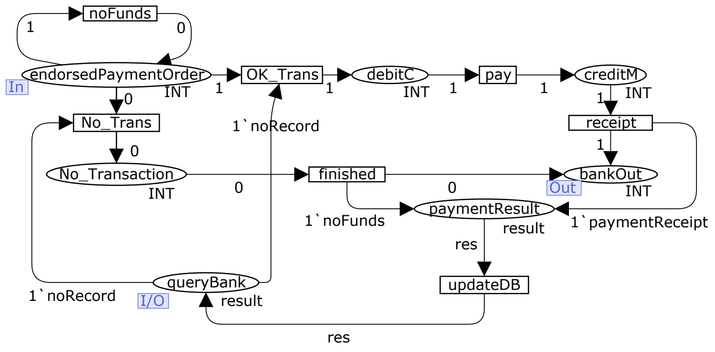


Fig. 6. The Bank (*B*) process page of the NetBill CP-net

Figure 6 introduces the bank process page that corresponds to the `BankProcess` substitution transition of Figure 2. On reception of an endorsed payment order at the place `endorsedPaymentOrder` the bank server either proceeds to the debit of the consumer's account or discovers a low balance account (because of the transition `noFunds`) and does not charge *C*. Both transaction cases are (assumed to be) executed atomically and this is modeled, by initially removing the `result` typed token found in place `queryBank` and finally placing an appropriate token value in it. Although this makes money atomicity self-proved, we are still interested to demonstrate the potentiality of the Computation Tree like temporal logic (CTL) supported in CPN

Tools to express and prove all payment atomicity properties, including money atomicity.

4 State space analysis and model checking

Figure 7 shows the standard report generated for the state space analysis of the described NetBill CP-net.

```
 Statistics
------------------------------------------------------------------------
  State Space
    Nodes:  59
    Arcs:   103
    Secs:   0
    Status: Full

  Scc Graph
    Nodes:  59
    Arcs:   103
    Secs:   0

Boundedness Properties
------------------------------------------------------------------------
  Best Integers Bounds               Upper          Lower
  BankProcess'No_Transaction 1       1              0
  BankProcess'creditM 1              1              0
  BankProcess'debitC 1               1              0
  BankProcess'paymentResult 1        1              0
  ConsumerProcess'epoInput 1         1              0
  MerchantProcess'goodsReqRcvd 1     1              0
  Protocol'bankOut 1                 1              0
  Protocol'ePaymentOrder 1           1              0
  Protocol'encrGoods 1               1              0
  Protocol'endTransaction 1          1              0
  Protocol'endorsedPaymentOrder 1    1              0
  Protocol'goodsRequest 1            1              0
  Protocol'prmtrs 1                  1              0
  Protocol'queryBank 1               1              0
  Protocol'reqQueue 1                1              1
  Protocol'stop 1                    1              0

Home Properties
------------------------------------------------------------------------
  Home Markings:  None

Liveness Properties
------------------------------------------------------------------------
  Dead Markings:  13 [59,658,57,56,55,...]
  Dead Transitions Instances: None
  Live Transitions Instances: None

Fairness Properties
------------------------------------------------------------------------
  No infinite occurrence sequences.
```

Fig. 7. The standard report generated for the state space analysis of the NetBill CP-net

The results for the standard properties checked in the shown report constitute a necessary input source, for correctly expressing the CTL formulae of the required correctness properties. We observe the absence of home markings and the absence of dead and live transitions instances. 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 payment atomicity are summarized in Table 1:

Table 1. State space querying functions

function description	use
`Mark.<PageName>'<PlaceName> N M`	Returns the set of tokens positioned in place <PlaceName> of 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.

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.

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.

Table 2 summarizes the CTL formulae used to express the required properties in terms of paths over the generated state space graph. 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. The ML statements need to activate the provided Computation Tree like temporal logic (CTL) are shown in Figure 8.

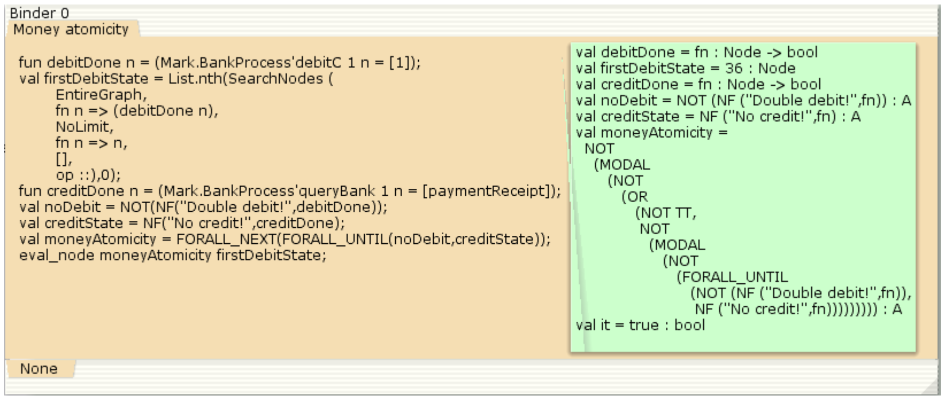
```
use (ogpath^"ASKCTL/BitArray.sml");
use (ogpath^"ASKCTL/ASKCTL.sml");
open ASKCTL;
```

Fig. 8. ML statements used to activate the provided CTL support

4.1 Model checking money atomicity

Figure 9 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`.



```
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;
```

```
val debitDone = fn : Node -> bool
val firstDebitState = 36 : Node
val creditDone = fn : Node -> bool
val noDebit = NOT (NF ("Double debit!",fn)) : A
val creditState = NF ("No credit!",fn) : A
val moneyAtomicity =
  NOT
    (MODAL
      (NOT
        (OR
          (NOT TT,
           NOT
             (MODAL
               (NOT
                 (FORALL_UNTIL
                   (NOT (NF ("Double debit!",fn)),
                    NF ("No credit!",fn))))))))) : A
val it = true : bool
```

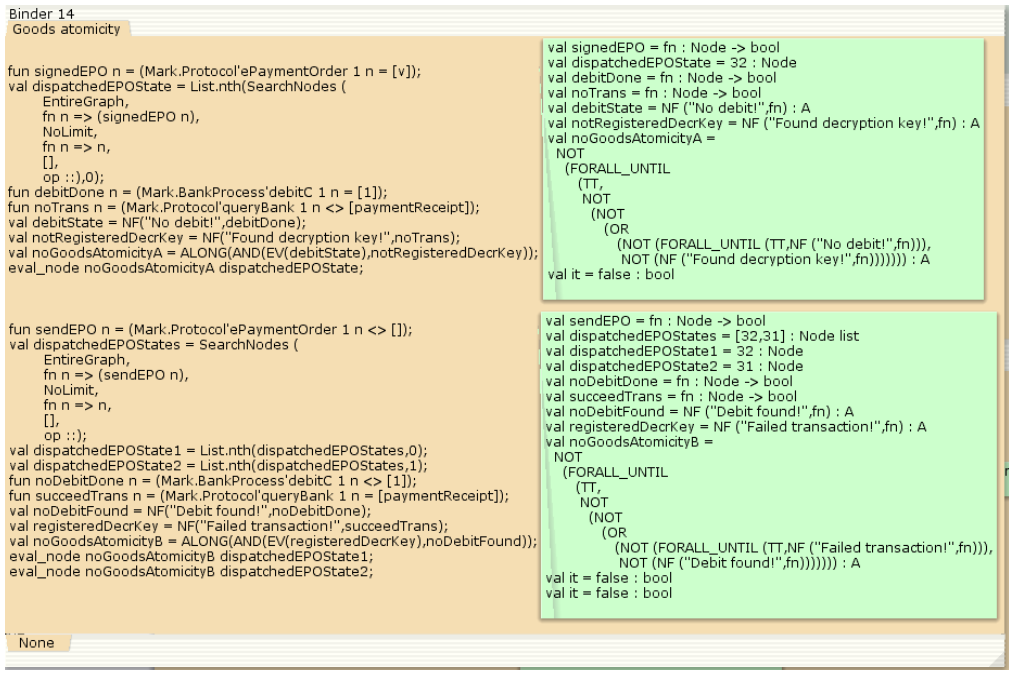
Fig. 9. Model checking money atomicity: true

4.2 Model checking goods atomicity

Figure 10 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.



```
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;
```

```
val signedEPO = fn : Node -> bool
val dispatchedEPOState = 32 : Node
val debitDone = fn : Node -> bool
val noTrans = fn : Node -> bool
val debitState = NF ("No debit!",fn) : A
val notRegisteredDecrKey = NF ("Found decryption key!",fn) : A
val noGoodsAtomicityA =
  NOT
   (FORALL_UNTIL
     (TT,
      NOT
       (NOT
         (OR
           (NOT (FORALL_UNTIL (TT,NF ("No debit!",fn))),
            NOT (NF ("Found decryption key!",fn))))))) : A
val it = false : bool
```

```
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;
```

```
val sendEPO = fn : Node -> bool
val dispatchedEPOStates = [32,31] : Node list
val dispatchedEPOState1 = 32 : Node
val dispatchedEPOState2 = 31 : Node
val noDebitDone = fn : Node -> bool
val succeedTrans = fn : Node -> bool
val noDebitFound = NF ("Debit found!",fn) : A
val registeredDecrKey = NF ("Failed transaction!",fn) : A
val noGoodsAtomicityB =
  NOT
   (FORALL_UNTIL
     (TT,
      NOT
       (NOT
         (OR
           (NOT (FORALL_UNTIL (TT,NF ("Failed transaction!",fn))),
            NOT (NF ("Debit found!",fn))))))) : A
val it = false : bool
val it = false : bool
```

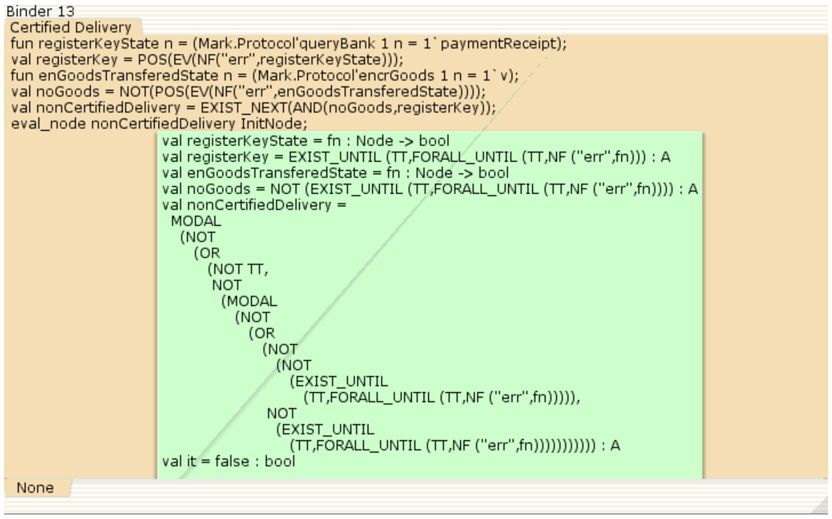
Fig. 10. Model checking the two parts of the non-atomic goods delivery: false

4.3 Model checking certified delivery

Figure 11 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).

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.



```
fun registerKeyState n = (Mark.Protocol'queryBank 1 n = 1`paymentReceipt);
val registerKey = POS(EV(NF("err",registerKeyState)));
fun enGoodsTransferedState n = (Mark.Protocol'encrGoods 1 n = 1`v);
val noGoods = NOT(POS(EV(NF("err",enGoodsTransferedState))));
val nonCertifiedDelivery = EXIST_NEXT(AND(noGoods,registerKey));
eval_node nonCertifiedDelivery InitNode;
```

```
val registerKeyState = fn : Node -> bool
val registerKey = EXIST_UNTIL (TT,FORALL_UNTIL (TT,NF ("err",fn))) : A
val enGoodsTransferedState = fn : Node -> bool
val noGoods = NOT (EXIST_UNTIL (TT,FORALL_UNTIL (TT,NF ("err",fn)))) : A
val nonCertifiedDelivery =
  MODAL
    (NOT
       (OR
          (NOT TT,
           NOT
             (MODAL
                (NOT
                   (OR
                      (NOT
                         (NOT
                            (EXIST_UNTIL
                               (TT,FORALL_UNTIL (TT,NF ("err",fn))))),
                       NOT
                         (EXIST_UNTIL
                            (TT,FORALL_UNTIL (TT,NF ("err",fn))))))))))) : A
val it = false : bool
```

Fig. 11. Model checking non-certified delivery: false

5 Protocol failure analysis

In general, protocol failure analysis aims in exploring all property violation scenarios (if any) 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.

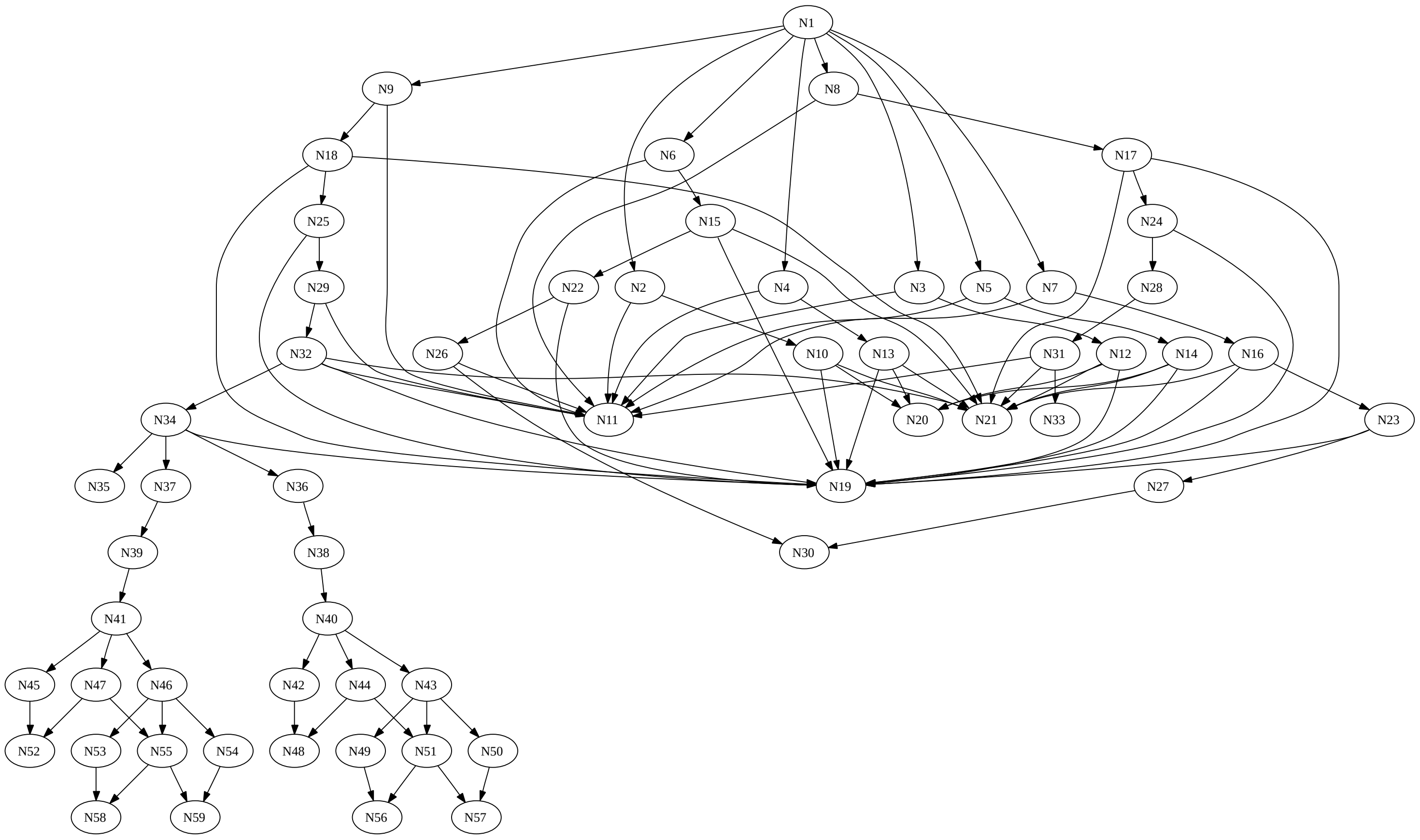


Fig. 12. NetBill's CP-net state space graph

Protocol failure analysis is based on inspection of the terminal markings (dead markings) in all property violation paths. The diagnostic strings positioned at places, like the place `stop` and the place `queryBank` in the top-level CP-net of NetBill (Figure 2), provide details for interactively simulating the corresponding protocol execution scenario. The simulation control functionality found in the latest version of CPN Tools allows firing transitions with an interactively chosen binding. Thus, the actions included in the scenario of interest are easily reproduced and the analyst explores all possible protocol revision prospects to repair the detected property violation. A necessary prerequisite is the selection of informative diagnostic strings in the phase of model development.

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 and the visualization of the generated state space graph.

CPN Tools exports the model's state space graph in a DOT language based text file that is then automatically visualized by an appropriate program, which implements well-tuned layout algorithms ([8]), for placing graph nodes and arcs. Figure 12 shows the generated state space graph for the described NetBill CP-net. Leaf nodes correspond to the dead markings to be inspected (Figure 13).

```
ListDeadMarkigs() -> val it =[59,58,57,56,55,
                              51,35,33,30,21,
                              20,19,10]:Node list
```

Fig. 13. Dead markings for the described NetBill CP-net

Finally, Table 3 provides a concise view of the performed protocol termination inspection.

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 or 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.

marking (N)	Mark.Protocol' stop 1 N	Mark.Protocol' queryBank 1 N	interpretation
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 request”]`	`[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.

5 Conclusion

This paper introduces the use of CP-nets and CPN Tools to model check the three levels of payment atomicity, for an electronic cash system. The combined use of appropriate state space exploration functions and CTL formulae allowed us to express and model check money atomicity, goods atomicity and certified delivery.

Although it was already known that NetBill possesses these three properties, we are not aware of published works in e-commerce, where CP-nets are used to model check correctness properties that are not directly related to the structural properties of the generated state space graph. We believe that the described approach is also applicable in more complex system models and is also possible to be extended for model checking other reliability and security properties.

We also proposed the performance of protocol failure analysis, in order to explore potential property violation scenarios and pinpoint areas, where design changes or revisions should be considered. In protocol failure analysis, it is possible to exploit the advanced graphical environment of CPN Tools to interactively simulate the actions included in a protocol execution scenario.

Our model checking experience suggests that CPN Tools is an attractive alternative over CSP-based model checking in e-commerce problems.

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. We also acknowledge the anonymous referees for their helpful comments.

References

1. Asokan, N., Fairness in electronic commerce, PhD thesis, University of Waterloo, Ontario, Canada, 1998
2. Chaum, D., Fiat, A., Naor, M., Untraceable electronic cash, In: Proceedings of CRYPTO'88, Springer-Verlag, 1990, 200-212
3. Chaum, D., Security without identification: Transaction systems to make big brother obsolete, Communications of the ACM, Vol. 28, No. 10, 1985, 1030-1044
4. Cheng, A., Christensen, S., Mortensen, K. H., Model checking Coloured Petri Nets exploiting strongly connected components, In: Proceedings of the International Workshop on Discrete Event Systems, Edinburg, Scotland, UK, 1996, 169-177
5. Clarke, E. M., Emerson, E. A., Sistle, A. P., Automatic verification of finite state concurrent system using temporal logic, ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2, 1986, 244-263
6. Cox, B., Tygar, J. D., Sirbu, M., NetBill security and transaction protocol, In: Proceedings of the 1st USENIX Workshop in Electronic Commerce, New York, NY, USENIX Association, California, 1995, 77-88
7. CPN Tools, http://wiki.daimi.au.dk/cpntools/cpntools.wiki
8. Gansner, E. R., Koutsofios, E., North, S. C., Vo, K.-P., A technique for drawing directed graphs, IEEE Transactions on Software Engineering, Vol. 19, No. 3, 1993, 214-230
9. Georgiadis, C. K., Manitsaris, A., Personalization in Mobile Commerce Environments: Multimedia Challenges, Cutter IT Journal, Vol. 18, No. 8, 2005, 36-43
10. Heintze, N., Tygar, J., Wing, J., Wong, H., Model checking electronic commerce protocols, In: Proceedings of the 2nd USENIX Workshop in Electronic Commerce, Oakland, CA, USENIX Association, California, 1996, 146-164
11. Herzberg, A. and Yochai, H., Minipay: Charging per click on the web, Computer Networks, Vol. 29, 1997, 939-951
12. Jensen, K., Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use, Volumes 1-3, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag, 1997
13. Kailar, R., Accountability in electronic commerce protocols, IEEE Transactions on Software Engineering, Vol. 22, No. 5, 1996, 313-328
14. Katsaros, P., On the design of access control to prevent sensitive information leakage in distributed object systems: a Colored Petri Net model, In: Proceedings of CoopIS/DOA/ODBASE, Lecture Notes in Computer Science 3761, Springer-Verlag, 2005, 945-962
15. Kempster, T. and Stirling, C., Modeling and model checking mobile phone payment systems, Proceedings of the FORTE 2003 Workshops, Lecture Notes in Computer Science 2767, Springer-Verlag, 2003, 95-110
16. Lu, S. and Smolka, S. A., Model checking the Secure Electronic Transaction (SET) protocol, In: Proceedings of the 7th International Symposium on Modeling, Analysis and Simulatio of Computer and Telecommunication Systems, 1999, 358-365
17. Ouyang, C., Kristensen, L. M. and Billington, J., A formal service specification for the Internet Open Trading Protocol, In: Proceedings of the 23rd International Conference on Applications and Theory of Petri Nets, Lecture Notes in Computer Science 2360, Springer-Verlag, 2002, 352-373
18. Ouyang, C., Kristensen, L. M. and Billington, J., A formal and executable specification of the Internet Open Trading Protocol, In: Proceedings of EC-Web 2002, Lecture Notes in Computer Science 2455, Springer-Verlag, 2002, 377-387

19. Ouyang, C. and Billington, J., On verifying the Internet Open Trading Protocol, In: Proceedings of EC-Web 2003, Lecture Notes in Computer Science 2738, Springer-Verlag, 2003, 292-302
20. Ouyang, C. and Billington, J., An improved formal specification of the Internet Open Trading Protocol, In: Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus, 2004, 779-783
21. Ouyang, C. and Billington, J., Formal analysis of the Internet Open Trading Protocol, In: Proceedings of the FORTE 2004 Workshops, Lecture Notes in Computer Science 3236, Springer-Verlag, 2004, 1-15
22. Panti, M., Spalazzi, L. and Tacconi, S., Verification of security properties in electronic payment protocols, In: Proceedings of the ACM SIGPLAN and IFIP WG 1.7 Workshop on Issues in the Theory of Security, Oregon, 2002
23. Ray, I., Ray, I., Failure Analysis of an E-Commerce Protocol using Model Checking, In: Proceedings of the Second International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems, San Jose, CA, June 2000, 176-183
24. Ray, I., Ray, I., Natarajan, N., An anonymous and failure resilient fair-exchange e-commerce protocol, Decision Support Systems, Vol. 39, 2005, 267-292
25. Rivest, R. L. and Shamir, A., Payword and MicroMint: Two simple micropayment schemes, In: Security Protocols Workshop, 1996, 69-87
26. Schneider, S., Modelling security properties with CSP, Tech. Report CSD-TR-96-04, Dept. of Computer Science, Royal Holloway, University of London, 1996
27. Schuldt, H., Popovici, A. and Schek, H.-J., Execution guarantees in electronic commerce payments, In: Proceedings of the 8th International Workshop on Foundations of Models and Languages for Data and Objects, Lecture Notes in Computer Science 1773, Springer-Verlag, 2000, 193-202
28. Shyamasundar, R. K. and Deshmukh, B., MicroBill: An efficient secure system for subscription based services, In: Proceedings of ASIAN 2002, Lecture Notes in Computer Science 2550, Springer-Verlag, 2002, 220-232
29. Xu, S., Yung, M., Zhang, G. and Zhu, H., Money conservation via atomicity in fair off-line e-cash, In: Proceedings of the 2nd Int. Workshop of Information Security, Lecture Notes in Computer Science 1729, Springer-Verlag, 1999, 14-31