TECHNISCHE UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Transitions into Black Box Views – The NetBill Protocol Revisited –

Max Breitling and Jan Philipps

TUM-I0013
SFB-Bericht Nr. 342/09/00 A
September 00

TUM–INFO–09-I0013-80/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Transitions into Black Box Views *

— The NetBill Protocol Revisited —

Max Breitling   Jan Philipps

Institut für Informatik
Technische Universität München
D-80290 München

{max.breitling|jan.philipps}@in.tum.de

Abstract

System specification by state machines together with property specification and verification by temporal logics are by now standard techniques to reason about the control flow of hardware components and embedded systems. The techniques to reason about the dataflow within loosely coupled systems, however, are less well developed.

In this contribution, we propose a formalism for the verification of systems with asynchronously communicating components. The components themselves are specified as state machines, while the dataflow between components is described as a relation over the input and output histories of a system. Communication history properties are derived from temporal logic properties of the component state machines. The history properties can then be used to deduce global properties of a complete system.

To demonstrate our approach, we model the NetBill protocol for micro-payments in the Internet and prove some correctness properties.

1 Introduction

State machines have become a popular technique to specify software and hardware systems. They are often described by various incarnations of state transition diagrams, which are a suggestive notation for component design

*This work is supported by the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen".

or implementation documents. Both in their graphical and in their non-graphical —such as B, VDM or Z— variants, state-based specification techniques have a precise semantics and clear operational models. Effects of state transitions can be analyzed by Hoare-like triples with pre- and post-conditions.

More abstract properties of state machines can be formulated with temporal logics to express invariance or liveness properties. Proofs in temporal logic often follow the operational intuition behind state machines: Invariance properties, for example, are typically shown using induction over the machine transitions.

Temporal logics are less well suited, however, to express properties of the data flow between loosely coupled components that communicate asynchronously via buffered communication channels. For such systems, black box views relating input and output communication histories of data flow components and systems are better suited. Such relations can be concisely formulated in the style of Focus [9, 10, 1]; they are inherently modular and allow easy reasoning about the global system behavior. In [2, 3, 5], we introduced a formalism for the verification of black box properties of systems with asynchronously communicating state machine components. The formalism builds on work by Manfred Broy [8]. In [6] tool support on the basis of Isabelle/HOL [12] is described.

In this paper, we demonstrate our approach with a model of the NetBill protocol. We specify the protocol in an operational way that is easy to implement but nevertheless abstract enough for verification purposes. We formulate essential properties at different abstraction levels, and sketch their formal proofs.

Section 2 contains a brief summary of black box and state machine specification techniques. Section 3 contains a state machine specification of the NetBill protocol for electronic payments in the Internet. In Section 4 we formalize corrrectness properties of the protocol as history relations, and show that the state machine specification indeed satisfies these properties. Section 5 contains a short discussion of our specification and verification approach. The conclusion in Section 6 summarizes the results and contains an outlook on future work.

2 Component and System Specifications

Our system model is a variant of the system model of Focus [9, 10, 1]. It is described in detail in [2]. We model a system by describing its components, its interface with respect to the system's environment, and its behavior. The components are connected via directed channels. The system's *interface* is described by the communication channels with the types of the message that are sent on them. The communication along all channels is modeled by finite or infinite message streams. The *behavior* of a system is characterized by a relation between the input and output streams, that we described in either of two different abstraction levels: black-box specifications and state machines.

2.1 Streams

The communication history between components is modeled by *streams*. A stream is a finite or infinite sequences of messages. Finite streams can be enumerated, for example: $\langle 1, 2, 3, \ldots 10\rangle$; the empty stream is denoted by $\langle\,\rangle$. For a set of messages Msg, the set of finite streams over Msg is denoted by Msg^*, that of infinite streams by Msg^∞. By Msg^ω we denote $\mathsf{Msg}^* \cup \mathsf{Msg}^\infty$.

Given two streams s, t and $j \in \mathbb{N}$, $\#s$ denotes the length of s. If s is finite, $\#s$ is the number of elements in s; if s is infinite, $\#s = \infty$. We write $s \frown t$ for the concatenation of s and t. If s is infinite, $s \frown t = s$. We write $s \sqsubseteq t$, if s is a prefix of t, i.e. if $\exists\, u \in \mathsf{Msg}^\omega \bullet s \frown u = t$. The j-th element of s is denoted by $s.j$, if $1 \leq j \leq \#s$; it is undefined otherwise. $\mathsf{ft}.s$ denotes the first element of a stream, i.e. $\mathsf{ft}.s = s.1$, if $s \neq \langle\,\rangle$. For $A \subseteq \mathsf{Msg}$ we denote by $A \circledS s$ the subsequence that results from s by removing all elements not in A. For singleton sets we often just write $a \circledS s$ instead of $\{a\} \circledS s$.

2.2 Black-Box Specifications

A black-box specification is an abstract description in the sense that it does not relate to any internals of the system, but just describes the external, visible behavior.

The behavior relation is defined by formulas Φ where the free variables range over the input and output streams. The streams fulfilling these predicates describe the allowed black-box-behavior of our system. We can use all the operators on streams to formulate the predicates.

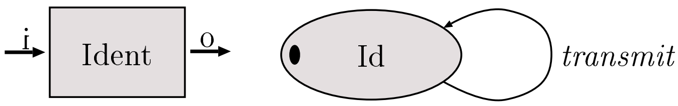
As a very simple example, consider a component *Identity* that just copies messages from one input channel i to one output channel o. Its black-box behavior specification is defined by the formula $o = i$.

2.3 State Machines

The behavior of a system can also be specified by a *state transition system* (STS), formalized by the tuple $\mathcal{S} = (I, O, A, \mathcal{I}, \mathcal{T})$. The names of the input and output channels are contained in I and O, respectively. The set A

contains for each $i \in I$ a variable i° (a prefix of i) denoting the sequence of messages already consumed by $\mathcal{S}$. Additionally, A may contain variables to represent local data, as e.g. a variable σ for the control state. A *state* of the system consists of a variable valuation that assigns values of the appropriate type to all variables. Channel variables are evaluated to streams containing the history of messages sent. The system starts in a state fulfilling the predicate $\mathcal{I}$, and $\mathcal{T}$ is the set of transitions.

State transition systems can be described in various ways, for example by state transition diagrams [2], by tables [10], or by the notation as used in this paper. All techniques have a common technique to describe a transition by four parts: A *precondition*, a set of *input statements*, a set of *output statements* and a *postcondition*. The informal meaning of a transition is as follows: If the available messages in the input channels can be matched with *Inputs*, the precondition is and the postcondition can be made true by assigning proper values to the primed variables, the transition is enabled. If it is chosen, the inputs are read, the outputs are written and the postcondition is made true.



transmit

pre	true
input	$i?x$
output	$o!x$
post	true

(a) Component (b) State Machine (c) Transition *transmit*

Figure 1: Identity Component

The component *Identity* (Figure 1) just needs one transition, called *transmit*, that is always enabled, reads some value x from i, and immediately sends it on o, without causing other changes in the components state.

Transitions can be schematically translated into logical formulas; see [2] for details.

2.4 Composition

Systems can be composed of several components by identifying channels with the same names. The composition can be graphically illustrated by *structure diagrams*, as used in Figure 4. The behavior of a composite system is completely defined by the behaviors of its components. Two components $\mathcal{S}_1$

and $\mathcal{S}_2$ can only be composed to $\mathcal{S} = \mathcal{S}_1 \otimes \mathcal{S}_2$ if they are *compatible* (defined in [2]), meaning essentially that they do not control the same variables.

Using black-box specifications, the behavior of the composed system is defined as the conjunction of the component behavior predicates. For state machines, a transition of the composed system consist of a transition of one component together with an environment transition of the others.

2.5 From State Machines to Black-Box Views

While the state machines represent an operational view on system's behavior, the black-box specifications can be seen as properties of the system. Therefore, it is crucial to be able to make a formal connection between both abstraction layers, since this allows us to prove that a (implemented) system has certain black-box properties, e.g. show that the component *Identity* with one transition *transmit* indeed fulfills the property $o = i$.

In [2], we used temporal logic to establish the connection between both abstraction levels. Properties can be split into a safety and progress part, that read in our simple example as

$$\Box\, o = i^{\circ}$$
$$\Box((\#o = k \wedge \#o < \#i) \Rightarrow \Diamond \#o \geq \#i)$$

The properties express that the output is correct in all reachable states (i.e. equal to the consumed messages on i) and that the output will be eventually extended as long as there is still buffered input left. Invariance is proved as usual by showing that the invariant is valid initially, and stays valid for all transitions. Output extension can be shown be finding helpful transitions that extend the output, and that are enabled, and therefore will be taken due to some fairness properties that are assumed in the execution model of the state machines. In [6] we suggest how verification diagrams and mechanized proof support assist the verification of properties of the above format.

Invariance properties form the basis of safety properties on the black box level. Invariants are also black box safety properties, if their free variables refer only to history variables ($I \cup I^{\circ} \cup O$) and if they are admissible [16] with respect to these variables.

Progress properties are derived from schemata similar to the property above; in general, the output channel length is compared with an arbitrary continous expression over the length of the input channels. More details can be found in [2].

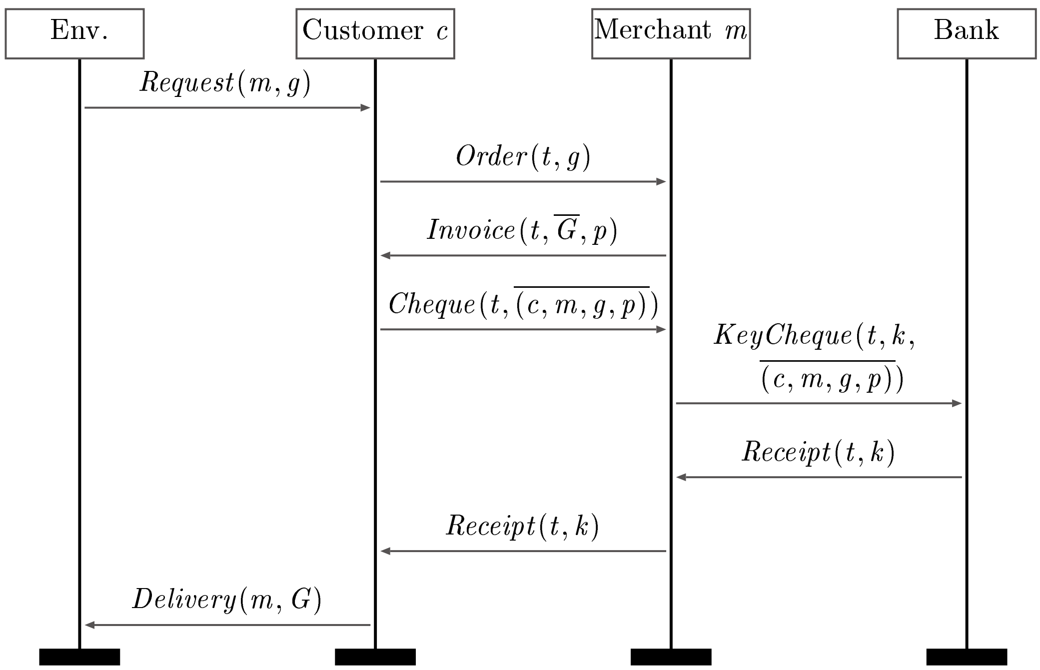


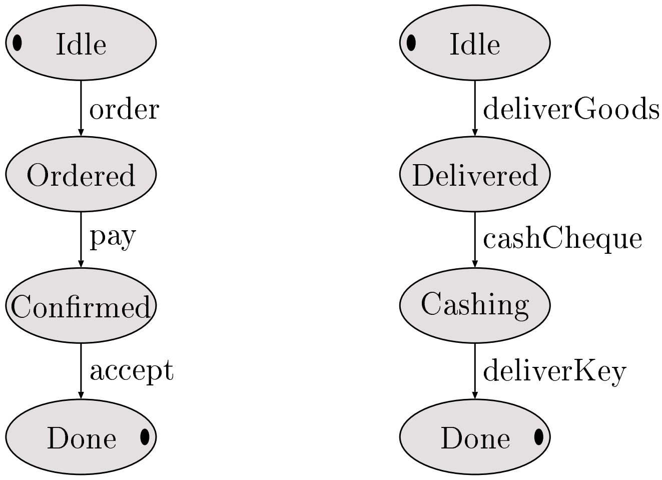
Figure 2: NetBill Transaction

3 NetBill Specification

The NetBill protocol [11, 19] supports low-cost transactions of electronic goods in the internet. Transactions occur between a customer process, a merchant process, and a centralized bank server. All money-related activities occur at the bank server.

Figure 2 show a sample transaction of the NetBill protocol. The customer process receives an order for electronic goods g at a merchant m from the environment. It generates a unique transaction number t which is used to identify the transaction in the subsequent message exchanges, and forwards the order to the merchant m. The merchant returns an invoice, which consists of a price statement and the encrypted goods. The customer process then issues a cheque to the merchant, which states that it is willing to pay the price for the goods. This cheque is digitally clearsigned: Every participant in the protocol can read it, but it is impossible for anyone to change the information in it. This cheque, together with the key for decrypting the goods, is forwarded to the bank. The bank returns a receipt and the key to the merchant, which forwards it to the customer. With this key, the customer process decrypts the goods received earlier and delivers them to the user.

Figures 3(a) and 3(b) show a transaction from the point of view of the



(a) Customer view (b) Merchant view

Figure 3: Customer and Merchant View of a Transaction

customer and merchant, respectively. The state structures are explained in more detail in Sections 3.2 and 3.3.

In this section, we give a formal specification of a simplified NetBill system. In Section 3.1 we define the state and message types used in the transaction protocol; Sections 3.2 to 3.4 contain component specifications for the customers, merchants and the bank.

3.1 General Definitions

Basic types. Figure 4 shows the architecture of a NetBill system. It consists of an arbitrary number of customers, an arbitrary number of merchants and the centralized bank server. Customers are identified by elements from a set CID of customer identifiers; similarly, we assume a set MID of merchant identifiers.

In contrast to other NetBill formalizations, we allow an arbitrary number of overlapping transactions between each customer and each merchant, i.e. a customer may order goods even if another transaction is not yet finished.. To identify the various transactions, we assume a set TID of transaction IDs that consist of a pair of the concerned customer and a unique serial number, i.e. $\mathrm{TID} \subseteq \mathrm{CID} \times \mathbb{N}$.

The electronic goods handled by the protocol are taken from a set GOODS; for each good there is a unique identifier in the set GID. We use a bijective

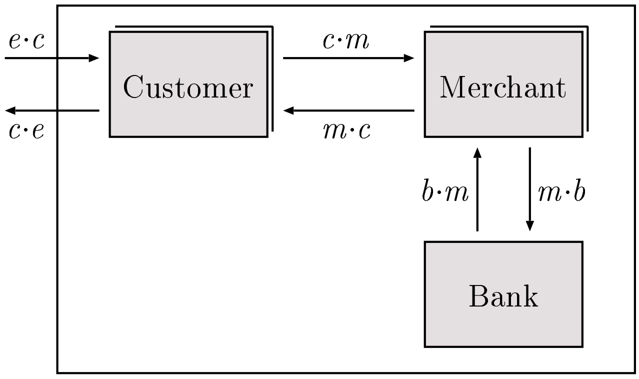


Figure 4: NetBill architecture

function

$$shelf : \mathrm{GID} \rightarrow \mathrm{GOODS}$$

to map IDs to the corresponding goods. Monetary values for prices are modeled by elements from a set $\mathbb{M}$. The prices of the goods are yielded by the function

$$price : \mathrm{GID} \rightarrow \mathbb{M}$$

Encryption. An essential part of any e-commerce protocol is encryption of messages. The NetBill transaction protocol uses both symmetric and public key cryptography. We abstract from the underlying algorithms, and just assume that for each message set M there exists a set of encrypted messages $\overline{M}$ and two functions

$$\textsc{Encrypt} : Key \rightarrow M \rightarrow \overline{M} \quad \text{and} \quad \textsc{Decrypt} : Key \rightarrow \overline{M} \rightarrow M$$

For symmetric encryption, we demand that

$$\textsc{Decrypt}\ k\ (\textsc{Encrypt}\ k\ m) = m$$

For public key encryption, we demand that the public keys of the customers are freely accessible by a function

$$\textsc{Pubkey} : \mathrm{CID} \rightarrow Key$$

and a message that is signed by customer c (with private key k_c) can be decrypted with the public key:

$$\textsc{Decrypt}\ \textsc{Pubkey}.c\ (\textsc{Encrypt}\ k_c\ m) = m$$

This latter requirement is satisfied, for example, by the well known RSA algorithm.

Message types. We define a number of complex data types to be used for messages on the communication channels.

- *Environment/Customer:* The only messages from the environment to the customer process are goods requests parameterized by the ID of the goods ordered, and the ID of the merchant from which the good is ordered:

  $$T_{ec} \ ::= \ \mathit{Request}(\mathrm{MID} \times \mathrm{GID})$$

  The only messages returned to the environment are the goods:

  $$T_{ce} \ ::= \ \mathit{Delivery}(\mathrm{MID} \times \mathrm{GOODS})$$

- *Customer/Merchant:* The customer sends two kinds of messages to the merchant: Orders of a certain good, and signed cheques which the merchant then forwards to the bank for further processing.

  $$\begin{array}{rcl} T_{cm} & ::= & \mathit{Order}(\mathrm{TID} \times \mathrm{GID}) \\ & \mid & \mathit{Cheque}(\mathrm{TID} \times \overline{(\mathrm{CID} \times \mathrm{MID} \times \mathrm{GID} \times \mathbb{M})}) \end{array}$$

  The merchant sends two kinds of messages to the customer:

  $$\begin{array}{rcl} T_{mc} & ::= & \mathit{Invoice}(\mathrm{TID} \times \overline{\mathrm{GOODS}} \times \mathbb{M}) \\ & \mid & \mathit{Receipt}(\mathrm{TID} \times \mathrm{KEY}) \end{array}$$

- *Merchant/Bank:* The merchant forwards the cheques to the bank together with the key to decrypt the involved goods.

  $$T_{mb} \ ::= \ \mathit{KeyCheque}(\mathrm{TID} \times \mathrm{KEY} \times \overline{(\mathrm{CID} \times \mathrm{MID} \times \mathrm{GID} \times \mathbb{M})})$$

  The bank sends a receipt to the merchant (to be forwarded to the customer) as a signal that the money transfer has succeeded.

  $$T_{bm} \ ::= \ \mathit{Receipt}(\mathrm{TID} \times \mathrm{KEY})$$

We sometimes form message sets by replacing parameters of a message constructor with the placeholder "$\cdot$". For example, we write

$$\mathit{Receipt}(t, \cdot) \quad \text{for} \quad \bigcup_{k \in \mathrm{KEY}} \{\mathit{Receipt}(t, k)\}$$

3.2 Customer

A customer is identified by its ID. The state space of each customer consists of its private key, and a mapping from transaction IDs to a *CustTrans* record. This record holds the control state of the transition according to Figure 3(a) as well as the goods ID, the merchant ID, the encrypted goods, the price of the goods, and the decryption key:

$$
\begin{array}{lrcl}
\mathit{CustTrans} ::= & \multicolumn{3}{l}{\textbf{record}} \\
 & \mathit{phase} & : & \{\textsc{Idle}, \textsc{Ordered}, \textsc{Confirmed}, \textsc{Done}\} \\
 & \mathit{gid} & : & \mathrm{GID} \\
 & \mathit{mid} & : & \mathrm{MID} \\
 & \mathit{goods} & : & \overline{\mathrm{GOODS}} \\
 & \mathit{price} & : & \mathbb{M} \\
 & \mathit{key} & : & \mathrm{KEY} \\
 & \multicolumn{3}{l}{\textbf{end}}
\end{array}
$$

Transactions IDs are determined by the customer; our specification uses a rather simple allocation scheme based on a variable *nexttid*, which holds the next free ID.

Thus, a customer $c \in \mathrm{CID}$ is specified in Figure 5. Initially, all transactions are idle. Each transaction is processed on the customer side as shown in Figure 3(a). A transaction gets activated by the transition *order* that is always enabled. The customer process receives an order consisting of a goods ID and a merchant ID to describe what should be bought from which merchant. A new transaction number is generated, all required data are stored in $st.t$, and the order is forwarded to the merchant.

The transition *pay* accepts the encrypted goods, and generates a signed cheque that is sent to the merchant. This transition is only enabled if the price the merchant offers is less than the price the customer expects. Note that the customer cannot be sure if he got the correct goods, since they are encrypted.

Finally, the customer process gets the decryption key from the merchant (transition *accept*). If the goods are the goods it expected, they are sent to the system environment, and the transaction status is set to Done.

3.3 Merchant

On the merchant side, each transaction is processed as shown in Figure 3(b). Each merchant must store for each transition the control state from Figure 3(b) as well as goods ID, customer ID, price and the decryption key:

$Customer[\mathrm{c}]$

inputs

$$\begin{array}{lcl} e{\cdot}c & : & T_{ec} \\ m{\cdot}c & : & T_{mc} \text{ for all } m \in MID \end{array}$$

outputs

$$\begin{array}{lcl} c{\cdot}e & : & T_{ce} \\ c{\cdot}m & : & T_{cm} \text{ for all } m \in MID \end{array}$$

state

$$\begin{array}{lcl} pk & : & \mathrm{KEY} \\ nexttid & : & \mathbb{N} \\ st & : & \mathrm{TID} \to CustTrans \end{array}$$

initialization

$$nexttid = 0 \wedge$$
$$\forall\, t \in \mathrm{TID} \bullet st.t.phase = \textsc{Idle}$$

transition relation

$$order[c, m, t, g], pay[c, m, t],$$
$$accept[c, m, t, k]$$

$order[c, m, t, g]$

pre

$$t = (c, nexttid)$$

input

$$e{\cdot}c?Request(m, g)$$

output

$$c{\cdot}m!Order(t, g)$$

post

$$\begin{array}{l} st.t.phase := \textsc{Ordered} \\ st.t.gid := g \\ st.t.price := price(g) \\ nexttid := nexttid + 1 \end{array}$$

$pay[c, m, y]$

pre

$$st.t.phase = \textsc{Ordered}$$
$$\wedge\; p \leq price(st.t.gid)$$

input

$$m{\cdot}c?Invoice(t, \overline{g}, p)$$

output

$$c{\cdot}m!Cheque(t, \textsc{Encrypt}\ pk$$
$$(c, m, st.t.gid, p))$$

post

$$\begin{array}{l} st.t.phase := \textsc{Confirmed} \\ st.t.goods := \overline{g} \\ st.t.price := p \end{array}$$

$accept[c, m, t, k]$

pre

$$st.t.phase = \textsc{Confirmed}$$
$$\wedge\; shelf.st.t.gid =$$
$$\textsc{Decrypt}\ k\ st.t.goods$$

input

$$m{\cdot}c?Receipt(t, k)$$

output

$$c{\cdot}e!Delivery(m,$$
$$\textsc{Decrypt}\ k\ st.t.goods)$$

post

$$\begin{array}{l} st.t.phase := \textsc{Done} \\ st.t.key := k \end{array}$$

Figure 5: Customer Specification

$Merchant$[m]

inputs

$c{\cdot}m \; : \; T_{cm}$ for all $c \in \mathrm{CID}$
$b{\cdot}m \; : \; T_{bm}$

outputs

$m{\cdot}c \; : \; T_{mc}$ for all $c \in \mathrm{CID}$
$m{\cdot}b \; : \; T_{mb}$

state

$st \; : \; \mathrm{TID} \rightarrow \mathit{MerchTrans}$

initialization

$\forall\, t \in \mathrm{TID} \bullet st.t.phase = \textsc{Idle}$

transition relation

$deliverGoods[c, m, t]$,
$cashCheque[c, m, t]$,
$deliverKey[c, m, t]$

$deliverGoods[c, m, t]$

pre

$st.t.phase = \textsc{Idle}$

input

$c{\cdot}m?Order(t, g)$

output

$m{\cdot}c!Invoice(t,$
$\textsc{Encrypt}(k, shelf.g),$
$price.g)$

post

$st.t.phase := \textsc{Delivered}$
$st.t.gid := g$
$st.t.price := price.g$
$st.t.key := k$

$cashCheque[c, m, t]$

pre

$st.t.phase = \textsc{Delivered} \wedge$
let $(cid, mid, gid, pr) =$
$\textsc{Decrypt}(\textsc{Pubkey}.c, ch)$
in
$cid = c \wedge mid = m$
$gid = st.t.gid \wedge pr = st.t.price$
end

input

$c{\cdot}m?Cheque(t, ch)$

output

$m{\cdot}b!KeyCheque(t, st.t.key, ch)$

post

$st.t.phase := \textsc{Cashing}$

$deliverKey[c, m, t]$

pre

$st.t.phase = \textsc{Cashing}$

input

$b{\cdot}m?Receipt(t, k)$

output

$m{\cdot}c!Receipt(t, k)$

post

$st.t.phase := \textsc{Done}$

Figure 6: Merchant Specification

$$
\begin{array}{lll}
\textit{MerchTrans} ::= & \textbf{record} & \\
& \quad \textit{phase} : & \{\textsc{Idle}, \textsc{Delivered}, \textsc{Cashing}, \textsc{Done}\} \\
& \quad \textit{gid} : & \text{GID} \\
& \quad \textit{cid} : & \text{CID} \\
& \quad \textit{price} : & \mathbb{M} \\
& \quad \textit{key} : & \text{KEY} \\
& \textbf{end} &
\end{array}
$$

A merchant can recognize a new order since the corresponding transaction t is in the phase IDLE. We do not model malicious behaviors, so we do not check if the transaction number really corresponds to the customer who sent the order, encoded by the name of the channel on which the order was received. The transition *deliverGoods* immediately sends the encrypted goods to the customer, and remembers relevant information in $st.t$. The transition *cashCheque* examines the cheque of the customer, and forwards it to the bank. If the bank confirms the receipt, the key will be sent to the customer and the transaction is completed. Merchants are specified formally in Figure 6.

3.4 Bank

The bank state consists of an account for each customer and each merchant, and a store of transaction descriptions that is modeled as a partial function from transaction IDs to description tuples.

The bank, specified in Figure 7, has only one transition that is enabled if a received cheque is correctly signed from the customer. The bank then transfers the money from the customers account to the merchants account, and sends a receipt to the merchant. It stores the information for eventual later requests.

4 NetBill Verification

In this section we show that our NetBill specification satisfies the following properties:

- *Guaranteed Delivery:* All goods ordered by the customer are delivered. Conversely, goods are only delivered if they were ordered. This is a typical black box property of the system; we use the verification techniques of [2, 5] to derive it.

Bank

inputs

$$m{\cdot}b \;:\; T_{mb} \text{ for all } m \in \mathrm{MID}$$

outputs

$$b{\cdot}m \;:\; T_{bm} \text{ for all } m \in \mathrm{MID}$$

state

$$\begin{array}{lcl} \textsc{Maccount} & : & \mathrm{MID} \rightarrow \mathbb{M} \\ \textsc{Caccount} & : & \mathrm{CID} \rightarrow \mathbb{M} \\ \mathit{keystore} & : & \mathrm{TID} \nrightarrow (\mathrm{KEY} \times \mathrm{CID} \times \mathrm{MID} \\ & & \qquad \times \mathrm{GID} \times \mathbb{M}) \end{array}$$

initialization

$$\mathit{keystore} = \varnothing$$

transition relation

$$\mathit{transfer}[m, t]$$

transfer$[m, t]$

pre

$$\begin{array}{l} \textbf{let } (\mathit{cid}, \mathit{mid}, \mathit{gid}, \mathit{pr}) = \textsc{Decrypt}(\textsc{Pubkey}.\mathit{fst}.t, \mathit{ch}) \\ \textbf{in } \mathit{cid} = \mathit{fst}.t \\ \textbf{end} \end{array}$$

input

$$m{\cdot}b?\mathit{KeyCheque}(t, k, \mathit{ch})$$

output

$$b{\cdot}m!\mathit{Receipt}(t, k)$$

post

$$\begin{array}{l} \textbf{let } (\mathit{cid}, \mathit{mid}, \mathit{gid}, \mathit{pr}) = \textsc{Decrypt}(\textsc{Pubkey}.\mathit{fst}.t, \mathit{ch}) \\ \textbf{in } \; \textsc{Maccount}.\mathit{mid} := \textsc{Maccount}.\mathit{mid} + \mathit{pr} \\ \qquad \textsc{Caccount}.\mathit{cid} := \textsc{Caccount}.\mathit{cid} - \mathit{pr} \\ \textbf{end} \\ \mathit{keystore} := \mathit{keystore} \cup \{t \mapsto (k, \mathit{cid}, \mathit{mid}, \mathit{gid}, \mathit{pr})\} \end{array}$$

Figure 7: Bank Specification

- *Guaranteed Payment:* For all goods ordered by a customer, the money amount corresponding to the good's price is subtracted from the customer account.
- *Money Atomicity:* The sum of the customer and merchant accounts remains invariant. Because of the centralized NetBill bank server, this property is quite obvious, it follows immediately from a simple formula in predicate logic.

In Section 4.1, we state and prove some invariants of the NetBill protocol. They are used in Section 4.2 to prove a number of basic safety and progress properties of the system components. In Sections 4.3 and 4.4 we show the correctness statements mentioned above.

4.1 Invariants

For the property proofs, we make use of two invariants. One invariant describes the NetBill behavior from the point of view of a single transaction, the other describes it from the point of view of a given customer.

In this section, we frequently refer to a given customer c and merchant m. For readability, we sometimes abbreviate $\mathit{Customer}[c]$ and $\mathit{Merchant}[m]$ by $\mathcal{C}$ and $\mathcal{M}$, respectively.

4.1.1 Transaction View

The transaction view describes the system state for all eight phases of a single NetBill transaction t between a customer c and a merchant m (see Figure 2). It is visualized in Figure 8. For example, the first row represents the state in which the transaction is still inactive: Customer and merchant are still idle, and no messages have been sent along the channels. Two subsequent rows describe a transition of either the customer, merchant or bank. For example, the second state reflects the system state after an order transition of the customer: The control state of the customer changes to ORDERED, and a new message *Order* is produced on channel $c{\cdot}m$. The other channels remain unchanged. In the table, newly generated messages are highlighted by boxes. A difference of the number of messages on a channel x and x° means that there are unread messages available on that channel. Note that the changes in the control state directly follow the state machine structures in Figures 3(a) and 3(b).

In addition to the control state and data flow information of Figure 8, the system accumulates information about the ordered goods, keys, and price in the customer and merchant data state variables. For each of the eight phases

	$\#\mathit{Order}(t,\cdot)\textcircled{s}c{\cdot}m$	$\#\mathit{Order}(t,\cdot)\textcircled{s}c{\cdot}m^{\circ}$	$\#\mathit{Cheque}(t,\cdot)\textcircled{s}c{\cdot}m$	$\#\mathit{Cheque}(t,\cdot)\textcircled{s}c{\cdot}m^{\circ}$	$\#\mathit{Invoice}(t,\cdot,\cdot)\textcircled{s}m{\cdot}c$	$\#\mathit{Invoice}(t,\cdot,\cdot)\textcircled{s}m{\cdot}c^{\circ}$	$\#\mathit{Receipt}(t,\cdot)\textcircled{s}m{\cdot}c$	$\#\mathit{Receipt}(t,\cdot)\textcircled{s}m{\cdot}c^{\circ}$	$\#\mathit{KeyCheque}(t,\cdot,\cdot)\textcircled{s}m{\cdot}b$	$\#\mathit{KeyCheque}(t,\cdot,\cdot)\textcircled{s}m{\cdot}b^{\circ}$	$\#\mathit{Receipt}(t,\cdot)\textcircled{s}b{\cdot}m$	$\#\mathit{Receipt}(t,\cdot)\textcircled{s}b{\cdot}m^{\circ}$	C.st.t.phase	M.st.t.phase
Γ_1	0	0	0	0	0	0	0	0	0	0	0	0	IDLE	IDLE
Γ_2	$\boxed{1}$	0	0	0	0	0	0	0	0	0	0	0	ORDERED	IDLE
Γ_3	1	$\boxed{1}$	0	0	$\boxed{1}$	0	0	0	0	0	0	0	ORDERED	DELIVERED
Γ_4	1	1	$\boxed{1}$	0	1	$\boxed{1}$	0	0	0	0	0	0	CONFIRMED	DELIVERED
Γ_5	1	1	1	$\boxed{1}$	1	1	0	0	$\boxed{1}$	0	0	0	CONFIRMED	CASHING
Γ_6	1	1	1	1	1	1	0	0	1	$\boxed{1}$	$\boxed{1}$	0	CONFIRMED	CASHING
Γ_7	1	1	1	1	1	1	$\boxed{1}$	0	1	1	1	$\boxed{1}$	CONFIRMED	DONE
Γ_8	1	1	1	1	1	1	1	$\boxed{1}$	1	1	1	1	DONE	DONE

Figure 8: Transaction View: Control Invariant

$$
\begin{array}{lcll}
\Delta_1 & \stackrel{\text{df}}{=} & \mathsf{true} & \\
\Delta_2 & \stackrel{\text{df}}{=} & \Delta_1 \wedge & \#\mathit{Order}(t,g)\text{Ⓢ}c{\cdot}m = 1 \quad \wedge \quad \mathcal{C}.st.t.gid = g \\
\Delta_3 & \stackrel{\text{df}}{=} & \Delta_2 \wedge & \#\mathit{Invoice}(t, \textsc{Encrypt}(k, \mathit{shelf}.g), \mathit{price}.g)\text{Ⓢ}m{\cdot}c = 1 \\
& & & \wedge\ \mathcal{M}.st.t.gid = g \wedge \mathcal{M}.st.t.key = k \\
\Delta_4 & \stackrel{\text{df}}{=} & \Delta_3 \wedge & \#\mathit{Cheque}(t, \textsc{Encrypt}\ \mathcal{C}.pk\ (c, m, \mathcal{C}.st.t.gid, \mathit{price}.g))\text{Ⓢ}c{\cdot}m \\
& & & = 1 \wedge \mathcal{C}.st.t.goods = \textsc{Encrypt}(k, \mathit{shelf}.g) \\
\Delta_5 & \stackrel{\text{df}}{=} & \Delta_4 \wedge & \#\mathit{KeyCheque}(t, \mathcal{M}.st.t.key, \\
& & & \qquad \textsc{Encrypt}\ \mathcal{C}.pk\ (c, m, \mathcal{C}.st.t.gid, \mathit{price}.g))\text{Ⓢ}m{\cdot}b = 1 \\
\Delta_6 & \stackrel{\text{df}}{=} & \Delta_5 \wedge & \#\mathit{Receipt}(t, \mathcal{M}.st.t.key)\text{Ⓢ}b{\cdot}m = 1 \\
\Delta_7 & \stackrel{\text{df}}{=} & \Delta_6 \wedge & \#\mathit{Receipt}(t, \mathcal{M}.st.t.key)\text{Ⓢ}m{\cdot}c = 1 \\
\Delta_8 & \stackrel{\text{df}}{=} & \Delta_7 &
\end{array}
$$

Figure 9: Transaction View: Data Invariant

in the protocol, we formulate a property Δ_i that relates the components' data state and message contents. The data invariant is shown in Figure 9. Note that in our example, the set of properties becomes stronger in every phase, since all transitions just set internal values that were undefined before. We only state a subset of the valid properties that is sufficient for the correctness proofs later; it is straightforward to extend them with pricing information.

Together, Figures 8 and 9 illustrate the following invariant for the NetBill system:

$$\forall\, c, m, t \bullet \exists\, g, k \bullet \bigvee_{i=1}^{8} (\Gamma_i \wedge \Delta_i)$$

The proof of the invariant proceeds along the typical inductive proof structure for invariants: It holds initially, and is not violated by any system transition.

4.1.2 Customer View

While the transaction invariant describes the system states for a given customer c, merchant m and transaction t, we now formalize an invariant that hides the transactions and just refers to the goods a customer ordered.

The customer issues orders to the NetBill system and accepts the delivered goods. Each order is handled by a single transaction. Transactions that did not yet lead to delivery of a good are called *pending*. The set of pending transactions for an order of a good g by the customer c at the merchant m

is denoted by $\mathcal{P}_{c,m,g}$ and defined as follows:

$$\mathcal{P}_{c,m,g} \stackrel{\text{df}}{=} \{\ t \in \text{TID} \ \mid \ \begin{array}{l} \mathit{Customer}[c].\mathit{st}.t.\mathit{phase} \notin \{\text{Idle}, \text{Done}\} \wedge \\ \mathit{Customer}[c].\mathit{st}.t.\mathit{gid} = g\ \} \end{array}$$

From the customer point of view, the number of orders for a good g should equal the number of deliveries of this good plus the number of pending transactions for g. This invariant is formally expressed as

$$\forall\, c, m, g \bullet \#\mathit{Delivery}(m, \mathit{shelf}.g)Ⓢc{\cdot}e + |\mathcal{P}_{c,m,g}| = \#\mathit{Request}(m, g)Ⓢe{\cdot}c^{\circ}$$

It is easy, though tedious, to prove this invariant. The proof structure is visualized by the verification diagram in Figure 10, which splits the invariant into the two cases $|\mathcal{P}_{c,m,g}| = 0$ and $|\mathcal{P}_{c,m,g}| > 0$. Each of the diagram nodes represents one case. The free variables c, m, g are fixed by skolemizing the quantifiers of the invariant above.

The opaque dot in the upper diagram node means that initially the upper case of the invariant holds; this is easy to see from the customer specification. The arrows show the verification obligations for the inductive step: Each new order increases $|\mathcal{P}_{c,m,g}|$ and the number of processed *Request* messages; each delivery of a good identified by g decreases $|\mathcal{P}_{c,m,g}|$, but increases the number of *Delivery* messages at the same time. Orders and deliveries of a different good $\hat{g}$ as well as the customer payment transitions leave the invariant unchanged. Moreover, although not represented in the diagram, it is easy to see that the invariant is also left unchanged by transitions of other customers, other merchants or the bank.

4.2 Black Box Properties

From the invariance and enabledness properties of the previous two sections, we now derive black box safety and liveness properties. The liveness properties are of a special form, which we call *progress*: They express that all external input to a component is processed.

4.2.1 Safety Properties

From the transition view invariant, we can derive a number of black box safety properties for a fixed transaction t, customer c, merchant m and the bank:

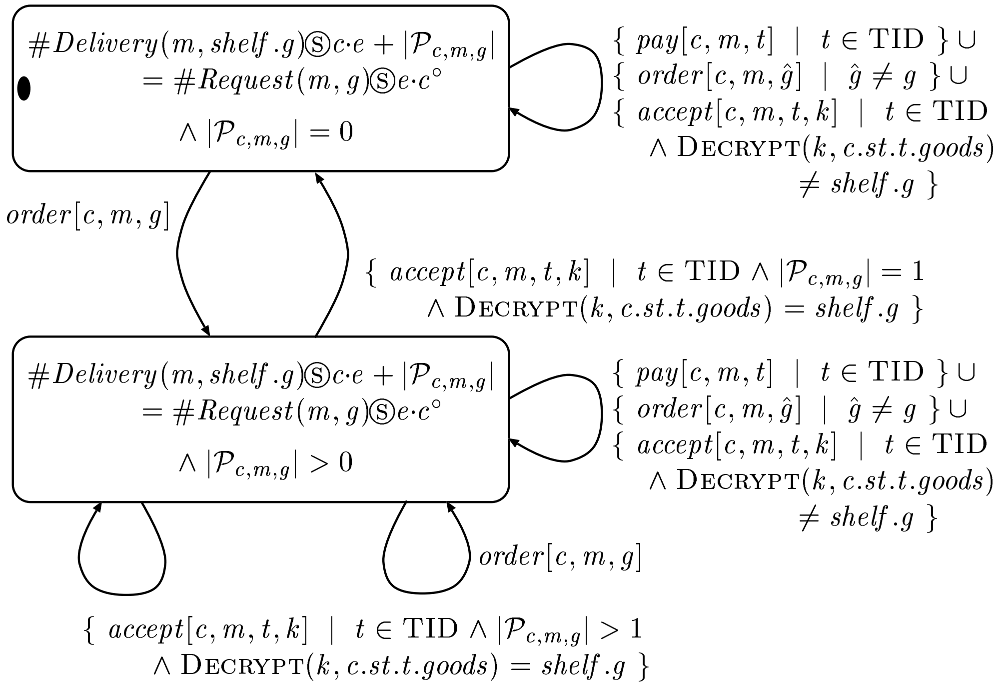


Figure 10: Customer View Invariant

Customer:	$\#Cheque(t, \cdot)Ⓢc{\cdot}m$	$=$	$\#Invoice(t, \cdot)Ⓢm{\cdot}c^{\circ}$
Merchant:	$\#Invoice(t, \cdot)Ⓢm{\cdot}c$	$=$	$\#Order(t, \cdot)Ⓢc{\cdot}m^{\circ}$
	$\#KeyCheque(t, \cdot, \cdot)Ⓢm{\cdot}b$	$=$	$\#Cheque(t, \cdot)Ⓢc{\cdot}m^{\circ}$
	$\#Receipt(t, \cdot)Ⓢm{\cdot}c$	$=$	$\#Receipt(t, \cdot)Ⓢb{\cdot}m^{\circ}$
Bank:	$\#Receipt(t, \cdot)Ⓢb{\cdot}m$	$=$	$\#KeyCheque(t, \cdot, \cdot)Ⓢm{\cdot}b^{\circ}$

To see that each of these properties indeed holds for the systems black box view, note that each property is an equality of continuous functions on streams. Moreover, each property is a state machine invariant of the Net-Bill system, which can be seen by comparing the entries for the left and right hand side of each equation in the transition view invariant table shown in Figure 8. For example, the third and the sixth column have the same entries, which means the the number of *Cheque*- and *Invoive*−messages on $c \cdot m$ and $m \cdot c^{\circ}$ are equal in all phases of a transaction. This proves the customer property.

In addition to the property above, the customer satisfies two other black box safety properties:

$$\begin{aligned} \#Order(\cdot, \cdot)Ⓢc{\cdot}m &= \#Request(m, \cdot)Ⓢe{\cdot}c^{\circ} \\ \#Delivery(m, \cdot)Ⓢc{\cdot}m &= \#Receipt(\cdot, \cdot)Ⓢm{\cdot}c^{\circ} \end{aligned}$$

These properties do not refer to given goods or transactions. They are based on invariants which again are easy to show using standard invariant verification techniques.

4.2.2 Progress Properties

The black box properties above relate the consumed input and the produced output of each component; they are pure safety properties, derived from state machine invariants like the transition invariant of Figure 8. The NetBill system enjoys also black box properties that relate the external input and the consumed input for each component:

$$
\begin{array}{lrcl}
\textbf{Customer:} & \#\mathit{Request}(\cdot,\cdot)Ⓢe{\cdot}c^{\circ} & = & \#\mathit{Request}(\cdot,\cdot)Ⓢe{\cdot}c \\
 & \#\mathit{Invoice}(t,\cdot)Ⓢm{\cdot}c^{\circ} & = & \#\mathit{Invoice}(t,\cdot)Ⓢm{\cdot}c \\
 & \#\mathit{Receipt}(t,\cdot)Ⓢm{\cdot}c^{\circ} & = & \#\mathit{Receipt}(t,\cdot)Ⓢm{\cdot}c \\
\textbf{Merchant:} & \#\mathit{Order}(t,\cdot)Ⓢc{\cdot}m^{\circ} & = & \#\mathit{Order}(t,\cdot)Ⓢc{\cdot}m \\
 & \#\mathit{Cheque}(t,\cdot,\cdot)Ⓢc{\cdot}m^{\circ} & = & \#\mathit{Cheque}(t,\cdot)Ⓢc{\cdot}m \\
 & \#\mathit{Receipt}(t,\cdot)Ⓢb{\cdot}m^{\circ} & = & \#\mathit{Receipt}(t,\cdot)Ⓢb{\cdot}m \\
\textbf{Bank:} & \#\mathit{KeyCheque}(t,\cdot,\cdot)Ⓢm{\cdot}b^{\circ} & = & \#\mathit{KeyCheque}(t,\cdot,\cdot)Ⓢm{\cdot}b
\end{array}
$$

These properties are immediate consequences of the five black box properties $e{\cdot}c^{\circ} = e{\cdot}c$, $m{\cdot}c^{\circ} = m{\cdot}c$, $c{\cdot}m^{\circ} = c{\cdot}m$, $b{\cdot}m^{\circ} = b{\cdot}m$ and $m{\cdot}b^{\circ} = m{\cdot}b$.

For the customer property $m \cdot c^{\circ} = m \cdot c$, it is sufficient to show that $\#m{\cdot}c^{\circ} \geq \#m{\cdot}c$, since $m{\cdot}c^{\circ} \sqsubseteq m{\cdot}c$ is by construction an invariant. This length property can be reduced to the following temporal logic property:

$$\Box\big(\#m{\cdot}c^{\circ} = k \wedge \#m{\cdot}c > k \Rightarrow \Diamond(\#m{\cdot}c^{\circ} > k)\big)$$

This is an example of an output extension property. To prove is, it is sufficient to find a helpful transition τ such that:

$$
\begin{array}{l}
\#m{\cdot}c^{\circ} = k \wedge \#m{\cdot}c > k \Rightarrow \mathsf{En}(\tau) \quad \text{and} \\
\#m{\cdot}c^{\circ} = k \wedge \#m{\cdot}c > k \wedge \tau \Rightarrow \#(m{\cdot}c^{\circ})' > k
\end{array}
$$

Assume now that the system is in a state where $\#m{\cdot}c^{\circ} = k \wedge \#m{\cdot}c > k$. Then we know that there is at least one unprocessed message in $m{\cdot}c$. In other words, there exists a message x such that

$$m{\cdot}c^{\circ} \frown \langle x \rangle \sqsubseteq m{\cdot}c$$

According to the type definition of T_{mc} in Section 3.1, the message x is either an invoice, or a receipt with a key; in any case, it refers to a transaction t:

1. $x \in \mathit{Invoice}(t,\cdot)$: Then $\#\mathit{Invoice}(t,\cdot)\circledS m\cdot c^\circ < \#\mathit{Invoice}(t,\cdot)\circledS m{\cdot}c$. From Figure 8, we now that the system must then be in a state characterized by Γ_3, and thus by $\Gamma_3 \wedge \Delta_3$.

   In this case, the helpful transition is the customer transition *pay*. It is enabled because according to Δ_3 the customer control state of t is ORDERED, and according to the assumption the next unread message is an invoice for t. Moreover, since *pay* indeed consumes the invoice message, the length of the processed input $m{\cdot}c^\circ$ is extended.

2. $x \in \mathit{Receipt}(t,\cdot)$: This case is similar to the one above; instead of the third phase, however, the system must be in phase 7: The system state is characterized by $\Gamma_7 \wedge \Delta_7$, and the helpful transition is the customer transition *accept*.

The proof for the other internal channels is analogous. The proof that $e{\cdot}c^\circ = e\cdot c$ is a bit different: Instead of appealing to the transition invariant table, we make use of the fact that there is always a free transaction ID (*nextid*), so that transition *order* is enabled whenever there is unprocessed input on channel $e{\cdot}c$.

4.3 Guaranteed Delivery

The black box correctness property states that all ordered goods are delivered, and that all delivered goods have been ordered:

$$\forall\, c, m, g \bullet \#\mathit{Delivery}(m, \mathit{shelf}.g)\circledS c{\cdot}e = \#\mathit{Request}(m,g)\circledS e{\cdot}c$$

This property can be split into a safety and a liveness part:

$$\forall\, c, m, g \bullet \#\mathit{Delivery}(m, \mathit{shelf}.g)\circledS c{\cdot}e \leq \#\mathit{Request}(m,g)\circledS e{\cdot}c$$

$$\forall\, c, m \bullet \#\mathit{Delivery}(m,\cdot)\circledS c{\cdot}e \geq \#\mathit{Request}(m,\cdot)\circledS e{\cdot}c$$

The liveness part need not to refer to a goods ID. Since the correctness of the output follows already from the safety part, it is sufficient to show that enough output is produced.

4.3.1 Safety

For the safety part, we observe that

$$\begin{aligned}
&\#\mathit{Delivery}(m, \mathit{shelf}.g)\circledS c{\cdot}e \\
&\quad\leq \#\mathit{Delivery}(m, \mathit{shelf}.g)\circledS c{\cdot}e + |\mathcal{P}_{c,m,g}| && \text{since } |\mathcal{P}_{c,m,g} \geq 0| \\
&\quad\leq \#\mathit{Request}(m,,\cdot)\circledS e{\cdot}c^\circ && \text{by customer view invariant} \\
&\quad\leq \#\mathit{Request}(m,g)\circledS e{\cdot}c && \text{since } e{\cdot}c^\circ \sqsubseteq e{\cdot}c
\end{aligned}$$

Thus, the safety formula is by itself an invariant of the NetBill system. It is also admissible, and is thus also valid at the black box level, where all free variables range over the limits of a system execution.

4.3.2 Liveness

For the liveness part we show the following equality sequence for all c and m:

$$
\begin{aligned}
&\#\mathit{Delivery}(m,\cdot)\}\circledS c{\cdot}e \\
&\quad = \#\mathit{Receipt}(\cdot,\cdot)\circledS m{\cdot}c^{\circ} && \text{by customer safety} \\
&\quad = \#\mathit{Order}(\cdot,\cdot)\circledS c{\cdot}m && \text{see below} \\
&\quad = \#\mathit{Request}(m,\cdot)\circledS e{\cdot}c^{\circ} && \text{by customer safety} \\
&\quad = \#\mathit{Request}(m,\cdot)\circledS e{\cdot}c && \text{by customer progress}
\end{aligned}
$$

To show the second equality, we observe that for all c, m and t:

$$
\begin{aligned}
&\#\mathit{Receipt}(t,\cdot)\circledS m{\cdot}c^{\circ} \\
&\quad = \#\mathit{Receipt}(t,\cdot)\circledS m{\cdot}c && \text{by customer progress} \\
&\quad = \#\mathit{Receipt}(t,\cdot)\circledS b{\cdot}m^{\circ} && \text{by merchant safety} \\
&\quad = \#\mathit{Receipt}(t,\cdot)\circledS b{\cdot}m && \text{by merchant progress} \\
&\quad = \#\mathit{KeyCheque}(t,\cdot,\cdot)\circledS m{\cdot}b^{\circ} && \text{by bank safety} \\
&\quad = \#\mathit{KeyCheque}(t,\cdot,\cdot)\circledS m{\cdot}b && \text{by bank progress} \\
&\quad = \#\mathit{Cheque}(t,\cdot)\circledS c{\cdot}m^{\circ} && \text{by merchant safety} \\
&\quad = \#\mathit{Cheque}(t,\cdot)\circledS c{\cdot}m && \text{by merchant progress} \\
&\quad = \#\mathit{Invoice}(t,\cdot,\cdot)\circledS m{\cdot}c^{\circ} && \text{by customer safety} \\
&\quad = \#\mathit{Invoice}(t,\cdot,\cdot)\circledS m{\cdot}c && \text{by customer progress} \\
&\quad = \#\mathit{Order}(t,\cdot)\circledS c{\cdot}m^{\circ} && \text{by merchant safety} \\
&\quad = \#\mathit{Order}(t,\cdot)\circledS c{\cdot}m && \text{by merchant progress}
\end{aligned}
$$

Each of the safety and progress steps in this sequence is immediate from the black box properties of Section 4.2.

4.4 Guaranteed Payment and Money Atomicity

The previous section showed the correctness of a black box property, which only refers to the channel communication histories. The correctness proof made use of both temporal logic properties (the invariance and response properties), and of predicate logic properties (for example, to show the enabledness of a transition).

However, temporal and predicate logic are useful not only for the derivation of black box properties. There are some properties, which are best expressed by temporal logic because they refer to the internal state of a component, and properties best expressed by predicate logic, because they also refer to a given transition.

4.4.1 Temporal Logic

This level is useful when properties about the internal states of a component are formalized. In the NetBill example, we can specify that if the customer orders a good, at a later state the price of the good will have been subtracted from the customer account.

$$
\begin{aligned}
&\forall\, x \in \mathbb{M}, c \in \mathrm{CID}, m \in \mathrm{MID}, t \in \mathrm{TID} \bullet \\
&\Box\big(\mathit{Bank}.\textsc{Caccount}.c = x \wedge \\
&\quad \mathit{Customer}[c].\mathit{st}.t.\mathit{phase} \in \{\textsc{Ordered}, \textsc{Confirmed}\} \\
&\quad\; \Rightarrow \Diamond\big(\mathit{Bank}.\textsc{Caccount}.c \le x - \mathit{price}.(\mathit{Customer}[c].\mathit{st}.t.\mathit{gid})\big)\big)
\end{aligned}
$$

Note that it would be quite difficult to precisely characterize the amount of money on the customer account, because there can be any number of active transaction being processed at any time.

4.4.2 Predicate Logic

The predicate logic level is useful when properties of single transitions are formalized. In the NetBill example, all money related activities occur at a centralized bank server, which is specified by a single transition. The property that within the NetBill system money is neither destroyed nor created can be expressed as the following formula in first-order logic:

$$
\begin{aligned}
&\forall\, x \in \mathbb{M}, c \in \mathrm{CID}, m \in \mathrm{MID} \bullet \\
&\qquad\qquad (\mathit{Bank}.\textsc{Caccount}.c + \mathit{Bank}.\textsc{Maccount}.m = x) \wedge \mathit{transfer} \\
&\qquad\qquad\quad \Rightarrow (\mathit{Bank}.\textsc{Caccount}'.c + \mathit{Bank}.\textsc{Maccount}.m' = x)
\end{aligned}
$$

5 Discussion

In this contribution, we used a simplified formalization of the NetBill protocol, which differs from the description in [11] in various respects. In this section, we discuss possible extensions of our model and briefly summarize the proof methodology from Section 4.

5.1 NetBill Model

In the original description of the NetBill protocol, the customer process can query the bank for the receipt and decryption key for a given transaction. This ensures that the customer can decrypt the goods even if the network connection between bank, merchant and customer can lose messages.

For customer queries, the system structure diagram must be extended by channels that connect the customers and the bank server (Figure 11). The transaction state transition diagrams must be extended as shown in Figure 12. It is straightforward to define the new transitions *query*, *acceptBank* and *dropKey*.

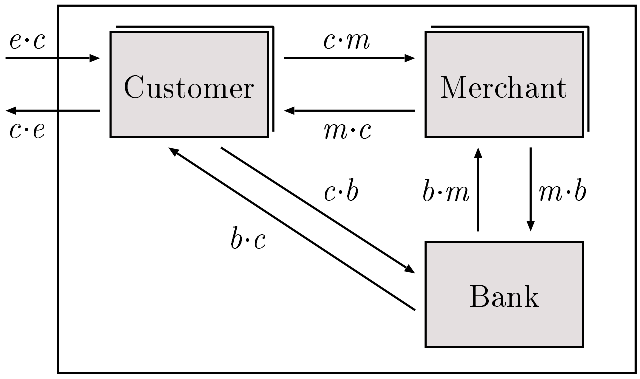


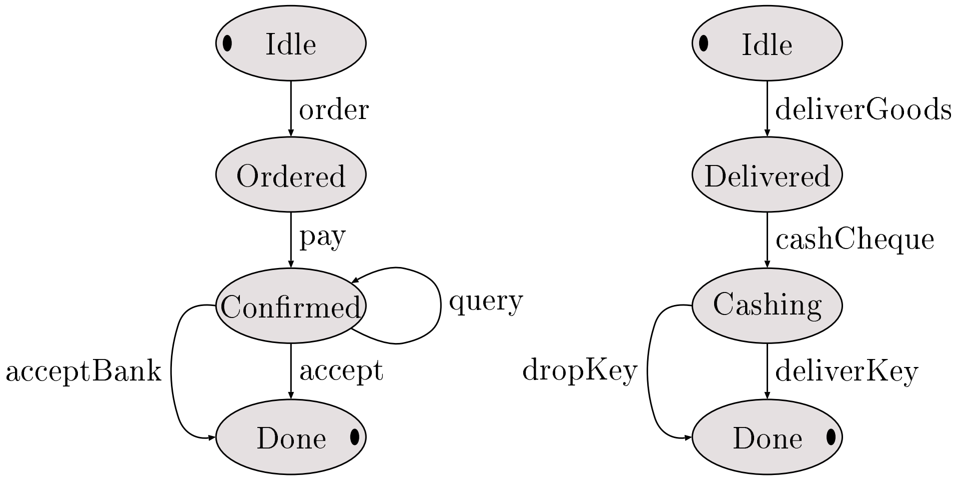
Figure 11: Extended NetBill Architecture

Moreover, the message types in [11] are more complex. In our model we removed much redundancy; we also abstracted from some details of the encryption algorithms. Finally, we do not examine malicious merchants and customers or limited customer credit lines, although of course the NetBill protocol has to be examined carefully for these cases.

5.2 Proof Methodology

Many of the proofs in Sections 4.1–4.3 are not completely formal; they cannot be, since we did not give precise translations definitions of the component state machine transitions into logic. Still, we hope that the reader can get the gist of the proof style that we used:

- Based on the message sequence chart of the NetBill protocol (Figure 2) and the transition state transition diagrams (Figure 3) the reachable states from the perspective of a single transactions could be concisely



(a) Customer view (b) Merchant view

Figure 12: Customer and Merchant View of a Transaction

described in tabular form (Figure 8). Again, we did not give a completely formal translation of the table into a logical expression, we refer to Broy's work on tabular specification techniques [7] for details.

- From the invariant table, a number of black box safety properties that relate input and output of each system component are immediate. Only the communication along the two channels that connect the system to the environent had to be treated specially.

- For black box liveness properties, we first proved component progress: Each component consumes all its input. For progress, we made use of temporal logic and black box verification rules from [2], which are tailored to proving the length properties typical for progress arguments. The premises of these rules —essentially enabledness statements for helpful transitions— could easily be discharged by a case split on the message types, and again a lookup in the invariant table.

- For the final black box correctness proof it is then sufficient to assemble the component black box properties in a chain of (in)equations.

Note that in this way, the correctness of the protocol is proven compositionally: History specifications in the style of Focus are a modular description technique that allows succinct reasoning with black box properties of components.

In previous examples [2, 3], we proved liveness properties directly, instead of showing progress first. The approach here seems to be more schematic and easier to scale to larger systems.

We did not yet attempt the correctness proofs for the extended NetBill version with customer queries, malicious customer or merchant behavior or limited credit lines; it remains to be seen whether and which parts of the current proof can be reused for the new aspects.

6 Conclusion

State based and I/O history based views of the system can be linked by temporal logical formulas for invariance and response. This technique is documented in more detail in [2, 8, 3, 5], where simple buffer examples are verified. In this contribution, we applied specification and proof principles for black box properties of distributed systems to the NetBill protocol [11, 19] for electronic payments over the Internet.

There is no single language for the formulation of mathematical correctness properties of a system. Simple Hoare-like verification conditions, temporal logic formulas and history relations in the style of FOCUS [9, 1] allow the formulation of properties at different abstraction levels. Conversely, verification conditions for each level lead to verification conditions formulated in the lower levels languages.

The black box correctness property of the NetBill system is assembled from black box properties of the individual components. This shows how the inherent modularity of FOCUS specifications can be used for concise compositional proofs: Both liveness and safety arguments are reduced to simple (in)equality reasoning.

The simple structure of the temporal logic formulas needed for the verification of black box properties is well-suited for verification diagrams [14, 15] in order to structure property proofs. While the verification conditions associated with a verification diagram are simple, their sheer number requires tool support. In [6], we present a formalization of our approach in Isabelle/HOL[12]. It consists of extensions of Shankar's PVS formalization of state machines [18] to handle liveness properties and asynchronous communication as well as verification tactics tailored to invariance and response diagrams. The Isabelle formalization and verification diagrams are documented in a technical reports [6]. The theory files and proof scripts can be accessed electronically [4].

Typically, when examining protocols for e-commerce applications, the focus is on security, and less on the safety and liveness issues handled by our approach. Further work will combine our verification techniques with Paulson's inductive approach to protocol verification [17], and with formal models of threat scenarios [13].

References

[1] M. Breitling, U. Hinkel, and K. Spies. Formale Entwicklung verteilter reaktiver Systeme mit Focus. In *FBT'1998, 8. GI/ITG Fachgespräch*, 1998.

[2] M. Breitling and J. Philipps. Black Box Views of State Machines. Technical Report TUM-I9916, Institut für Informatik, Technische Universität München, 1999.

[3] M. Breitling and J. Philipps. Diagrams for dataflow. In *FBT'2000, 10. GI/ITG Fachgespräch*, June 2000.

[4] M. Breitling and J. Philipps. State machine theories and proof scripts for Isabelle/HOL. `http://www.in.tum.de/~philipps/BBV`, 2000.

[5] M. Breitling and J. Philipps. Step by step to histories. In *International Conference on Algebraic Methodology And Software Technology, AMAST 2000, LNCS 1816*, 2000.

[6] M. Breitling and J. Philipps. Verification diagrams for dataflow properties. Technical Report TUM-I0005, Institut für Informatik, Technische Universität München, 2000.

[7] M. Broy. Pragmatic and Formal Specification of System Properties by Tables. Technical Report TUM-I9802, Institut für Informatik, Technische Universität München, 1998.

[8] M. Broy. From states to histories. Lecture Notes of the Marktoberdorf Summer School on Engineering Theories of Software Construction, 2000.

[9] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner, and R. Weber. The Design of Distributed Systems: An Introduction to Focus—Revised Version. Technical Report TUM-I9202-2, Institut für Informatik, Technische Universität München, 1993.

[10] M. Broy and K. Stølen. *Specification and Development of Interactive Systems - FOCUS on Streams, Interfaces and Refinement*. Springer, 2000. To appear.

[11] B. Cox, J. D. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *Proc. of the First USENIX Workshop in Electronic Commerce*, pages 77–88, 1995.

[12] Isabelle home page. `http://isabelle.in.tum.de`.

[13] V. Lotz. Threat scenarios as a means to formally develop secure systems. *Journal of Computer Security 5 (1997), pp. 31-67*, 1997.

[14] Z. Manna, A. Browne, H. B. Sipma, and T. E. Uribe. Visual abstractions for temporal verification. In *LNCS 1548*, pages 28–41, 1998.

[15] Z. Manna and A. Pnueli. Temporal verification diagrams. In *International Symposium on Theoretical Aspects of Computer Software, LNCS 789*, pages 726–765, 1994.

[16] L. C. Paulson. *Logic and Computation.* Cambridge University Press, 1987.

[17] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security 6 (1998), pp. 85-128*, 1997.

[18] N. Shankar. A lazy approach to compositional verification. Technical Report CSL-93-08, Computer Science Laboratory, SRI, 1993.

[19] J. D. Tygar and M. Sirbu. Netbill: An internet commerce system optimized for network delivered services. *IEEE Personal Communications*, 2(4):34–39, 1995.

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

Liste aller erschienenen Berichte von 1990-1994
auf besondere Anforderung

342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids

342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law

342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space

342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes

342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems

342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter

342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology

342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies

342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness

342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design

342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System

342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm

342/13/95 A Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming

342/14/95 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks

342/15/95 A Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs

342/16/95 A Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik

342/17/95 A Georg Stellner: Using CoCheck on a Network of Workstations

342/18/95 A Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications

Reihe A

342/19/95 A Thomas Schnekenburger: Integration of Load Distribution into ParMod-C

342/20/95 A Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication

342/21/95 A Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control

342/22/95 A Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes

342/23/95 A Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation

342/24/95 A M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems

342/01/96 A Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries

342/02/96 A Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs

342/03/96 A Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes

342/04/96 A Thomas Huckle: Efficient Computation of Sparse Approximate Inverses

342/05/96 A Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification

342/06/96 A Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components

342/07/96 A Richard Mayr: Some Results on Basic Parallel Processes

342/08/96 A Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht

342/09/96 A P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme

342/10/96 A Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFSLib – A File System for Parallel Programming Environments

342/11/96 A Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions

342/12/96 A Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete

342/13/96 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks

342/14/96 A Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project

Reihe A

342/15/96 A Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus

342/16/96 A Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen

342/17/96 A Richard Mayr: Model Checking PA-Processes

342/18/96 A Michaela Huhn, Peter Niebert, Frank Wallner: Put your Model Checker on Diet: Verification on Local States

342/01/97 A Tobias Müller, Stefan Lamberts, Ursula Maier, Georg Stellner: Evaluierung der Leistungsf"ahigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken

342/02/97 A Hans-Joachim Bungartz and Thomas Dornseifer: Sparse Grids: Recent Developments for Elliptic Partial Differential Equations

342/03/97 A Bernhard Mitschang: Technologie f"ur Parallele Datenbanken - Bericht zum Workshop

342/04/97 A nicht erschienen

342/05/97 A Hans-Joachim Bungartz, Ralf Ebner, Stefan Schulte: Hierarchische Basen zur effizienten Kopplung substrukturierter Probleme der Strukturmechanik

342/06/97 A Hans-Joachim Bungartz, Anton Frank, Florian Meier, Tilman Neunhoeffer, Stefan Schulte: Fluid Structure Interaction: 3D Numerical Simulation and Visualization of a Micropump

342/07/97 A Javier Esparza, Stephan Melzer: Model Checking LTL using Constraint Programming

342/08/97 A Niels Reimer: Untersuchung von Strategien für verteiltes Last- und Ressourcenmanagement

342/09/97 A Markus Pizka: Design and Implementation of the GNU INSEL-Compiler gic

342/10/97 A Manfred Broy, Franz Regensburger, Bernhard Schätz, Katharina Spies: The Steamboiler Specification - A Case Study in Focus

342/11/97 A Christine Röckl: How to Make Substitution Preserve Strong Bisimilarity

342/12/97 A Christian B. Czech: Architektur und Konzept des Dycos-Kerns

342/13/97 A Jan Philipps, Alexander Schmidt: Traffic Flow by Data Flow

342/14/97 A Norbert Fröhlich, Rolf Schlagenhaft, Josef Fleischmann: Partitioning VLSI-Circuits for Parallel Simulation on Transistor Level

342/15/97 A Frank Weimer: DaViT: Ein System zur interaktiven Ausführung und zur Visualisierung von INSEL-Programmen

342/16/97 A Niels Reimer, Jürgen Rudolph, Katharina Spies: Von FOCUS nach INSEL - Eine Aufzugssteuerung

342/17/97 A Radu Grosu, Ketil Stølen, Manfred Broy: A Denotational Model for Mobile Point-to-Point Data-flow Networks with Channel Sharing

342/18/97 A Christian Röder, Georg Stellner: Design of Load Management for Parallel Applications in Networks of Heterogenous Workstations

Reihe A

342/19/97 A	Frank Wallner: Model Checking LTL Using Net Unfoldings
342/20/97 A	Andreas Wolf, Andreas Kmoch: Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung zur Lösung eines Beispiels aus der Hardware-Verifikation – Fallstudie –
342/21/97 A	Andreas Wolf, Marc Fuchs: Cooperative Parallel Automated Theorem Proving
342/22/97 A	T. Ludwig, R. Wismüller, V. Sunderam, A. Bode: OMIS - On-line Monitoring Interface Specification (Version 2.0)
342/23/97 A	Stephan Merkel: Verification of Fault Tolerant Algorithms Using PEP
342/24/97 A	Manfred Broy, Max Breitling, Bernhard Schätz, Katharina Spies: Summary of Case Studies in Focus - Part II
342/25/97 A	Michael Jaedicke, Bernhard Mitschang: A Framework for Parallel Processing of Aggregat and Scalar Functions in Object-Relational DBMS
342/26/97 A	Marc Fuchs: Similarity-Based Lemma Generation with Lemma-Delaying Tableau Enumeration
342/27/97 A	Max Breitling: Formalizing and Verifying TimeWarp with FOCUS
342/28/97 A	Peter Jakobi, Andreas Wolf: DBFW: A Simple DataBase FrameWork for the Evaluation and Maintenance of Automated Theorem Prover Data (incl. Documentation)
342/29/97 A	Radu Grosu, Ketil Stølen: Compositional Specification of Mobile Systems
342/01/98 A	A. Bode, A. Ganz, C. Gold, S. Petri, N. Reimer, B. Schiemann, T. Schnekenburger (Herausgeber): "'Anwendungsbezogene Lastverteilung"', ALV'98
342/02/98 A	Ursula Hinkel: Home Shopping - Die Spezifikation einer Kommunikationsanwendung in Focus
342/03/98 A	Katharina Spies: Eine Methode zur formalen Modellierung von Betriebssystemkonzepten
342/04/98 A	Stefan Bischof, Ernst-W. Mayr: On-Line Scheduling of Parallel Jobs with Runtime Restrictions
342/05/98 A	St. Bischof, R. Ebner, Th. Erlebach: Load Balancing for Problems with Good Bisectors and Applications in Finite Element Simulations: Worst-case Analysis and Practical Results
342/06/98 A	Giannis Bozas, Susanne Kober: Logging and Crash Recovery in Shared-Disk Database Systems
342/07/98 A	Markus Pizka: Distributed Virtual Address Space Management in the MoDiS-OS
342/08/98 A	Niels Reimer: Strategien für ein verteiltes Last- und Ressourcenmanagement
342/09/98 A	Javier Esparza, Editor: Proceedings of INFINITY'98
342/10/98 A	Richard Mayr: Lossy Counter Machines
342/11/98 A	Thomas Huckle: Matrix Multilevel Methods and Preconditioning

Reihe A

342/12/98 A	Thomas Huckle: Approximate Sparsity Patterns for the Inverse of a Matrix and Preconditioning
342/13/98 A	Antonin Kucera, Richard Mayr: Weak Bisimilarity with Infinite-State Systems can be Decided in Polynomial Time
342/01/99 A	Antonin Kucera, Richard Mayr: Simulation Preorder on Simple Process Algebras
342/02/99 A	Johann Schumann, Max Breitling: Formalisierung und Beweis einer Verfeinerung aus FOCUS mit automatischen Theorembeweisern – Fallstudie –
342/03/99 A	M. Bader, M. Schimper, Chr. Zenger: Hierarchical Bases for the Indefinite Helmholtz Equation
342/04/99 A	Frank Strobl, Alexander Wisspeintner: Specification of an Elevator Control System
342/05/99 A	Ralf Ebner, Thomas Erlebach, Andreas Ganz, Claudia Gold, Clemens Harlfinger, Roland Wism"uller: A Framework for Recording and Visualizing Event Traces in Parallel Systems with Load Balancing
342/06/99 A	Michael Jaedicke, Bernhard Mitschang: The Multi-Operator Method: Integrating Algorithms for the Efficient and Parallel Evaluation of User-Defined Predicates into ORDBMS
342/07/99 A	Max Breitling, Jan Philipps: Black Box Views of State Machines
342/08/99 A	Clara Nippl, Stephan Zimmermann, Bernhard Mitschang: Design, Implementation and Evaluation of Data Rivers for Efficient Intra-Query Parallelism
342/09/99 A	Robert Sandner, Michael Mauderer: Integrierte Beschreibung automatisierter Produktionsanlagen - eine Evaluierung praxisnaher Beschreibungstechniken
342/10/99 A	Alexander Sabbah, Robert Sandner: Evaluation of Petri Net and Automata Based Description Techniques: An Industrial Case Study
342/01/00 A	Javier Esparza, David Hansel, Peter Rossmanith, Stefan Schwoon: Efficient Algorithm for Model Checking Pushdown Systems
342/02/00 A	Barbara König: Hypergraph Construction and Its Application to the Compositional Modelling of Concurrency
342/03/00 A	Max Breitling and Jan Philipps: Verification Diagrams for Dataflow Properties
342/04/00 A	Günther Rackl: Monitoring Globus Components with MIMO
342/05/00 A	Barbara König: Analysing Input/Output Capabilities of Mobile Processes with a Generic Type System
342/06/00 A	Michael Bader, Christoph Zenger: A Parallel Solver for Convection Diffusion Equations based on Nested Dissection with Incomplete Elimination

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech1: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox -Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS