Modeling and Model Checking Mobile Phone Payment Systems

Tim Kempster* and Colin Stirling

School of Informatics, University of Edinburgh,
JCMB, Edinburgh, Scotland, EH9 3JZ,
{tdk,cps}@dcs.ed.ac.uk

Abstract. Recently a technique for transacting goods using GSM mobile phones has become very popular. We present a formal model of these novel transactions using a views based modeling technique. We show how to express two safety properties namely goods and money atomicity within this model using a sub-logic of CTL. By automatically generating a labelled transition system from our views model we can model check these properties. We show how to generalise this model to arbitrary numbers of processes. Goods atomicity fails under certain circumstances thus exposing some deficiencies that exist in existing implementations.

1 Introduction

Using mobile telephones to pay for goods and services has become very popular recently, particularly with younger people. A major reason is that these payment techniques extend the hugely popular Short Message Services (SMS) available on 600 million mobile phones world wide. Acording to the GSM association over 24 billion SMS text messages were sent around the globe in the month of May 2002 alone.

The sort of goods and services that are purchased include customised telephone ring-tones, software licenses, mobile games, digital music and internet content. Although the value of individual transactions conducted in this way is quite small, typically around €1-5, the Wireless World Forum 2002 estimate €47.2 billion per year will be transacted by 2006.

In this paper we model an abstracted real life mobile phone payment systems currently in use. We use temporal logic to express properties of the payment systems, and observe that model checking can be used to prove or refute properties. Finaly we expose some non-trivial problems with these payment systems.

The payment systems we model are not fictitious, they are currently implemented and widely used. Furthermore, the problems we expose can be readily demonstrated to exist in commercial implementations. Currently, these deficiencies restrict the goods that can be transacted to a fairly trivial nature, for example mobile phone ring-tones. Developing tools that allow formal modeling and verification of properties of these types of payment systems will lead to more

* This research was supported by EPSRC grant GR/N21141.

H. König, M. Heiner, and A. Wolisz (Eds.): FORTE 2003, LNCS 2767, pp. 95–110, 2003.

robust systems in the future which could then be used to transact a wider range of goods and services with increased confidence.

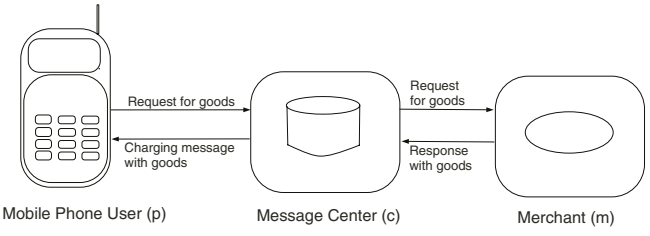

Fig. 1. A very simple mobile transaction. The mobile phone user requests some goods. The goods are delivered using a charging message. A mobile message center relays the requests and goods from merchant to phone.

2 Simple Mobile Transactions

A user wishing to purchase some goods (throughout we use the term goods to mean both goods and services) makes a request using a mobile phone (p), by texting a keyword, to a particular charging number called a *short code*. The mobile operator's message center (c) then routes this request to a particular merchant (m). The keyword allows the mobile operator to route the request to a particular merchant and also specifies the goods requested. Once the merchant receives the request it then replies to p by sending the goods, via the message center (c). When the mobile user receives the reply, containing the goods, their mobile phone bill is debited by a particular amount. The message they receive is called a *charging message*.

Although this seems very straightforward there are some interesting attributes which makes the analysis more interesting. Messages are passed asynchronously [10]. Asynchronous message passing systems are amongst the hardest to implement correctly. Message failure is possible. Cellular Mobile devices can drift in an out of coverage areas. If a message cannot be conveyed immediately, between message center and mobile phone, it is stored and retried later. There is a limit to the time a message is stored, at the message center, and thus if a mobile device is out of a coverage area for long enough, messages are deleted. Many phone subscribers opt for a *pay-as-you-use* service. This is where credits are purchased in advance and then used up as the mobile phone is used. If sufficient credits do not exist to accept a charging message the message will not be delivered. Many transactions rely on the delivery of several charging messages to supply goods. We will investigate these later and see how problems can arise.

3 The Views Model

In this section we introduce a views based model [9] to understand the exchange of information which takes place during mobile transactions. The term *protocol* is often used to describe the interaction of the various parties involved in a transaction. We first describe the components of our model and then discuss how they can be used.

3.1 Processes, Local State and Views

We model a transaction as a system of processes. Processes communicate by means of message passing. Each process belongs to a particular class of processes, corresponding to the role it plays in the transaction. Processes have a set of local state variables, together these variables constitute a process's *internal state*. Each class has a set of rules that determine the behaviour of all processes that belong to it. We are interested in three such classes: the merchants, the message centers, and the mobile phones. Initially, we only consider the interaction of a single merchant, m, message center c and phone user p as they carry out a single transaction.

Let p be a phone process that has a local state variable g, denoted $p.g$. Variables can have values. If p's variable g has value `tt` (true) we say $p.g$ holds at p. The variables of each process have initial values. Most of our variables will be boolean type. The type of a view is the same as the variable it is viewing with the addition of $\perp$ which means undefined or unknown. Views have initial value $\perp$. If $p.g = \perp$ then $p.g$ fails but $\neg p.g$ holds. In other words $\perp$ is treated like `ff` (false) when tested.

Together with its local state variables a process may have a *view* of the internal state variables of other processes. This view is constructed from information it receives from other processes in the form of messages. We say, $@p(c.g)$ holds at process p if the most up-to-date view p has of $c.g$ is `tt`. That is p has received a message from c informing p that c's variable g *had* the value `tt`. It is important to note that if at some point, $c.g$ has value `tt` then this does *not* imply that, $@p(c.g)$ holds. This is because the message reporting c's state change may not have arrived at p. Similarly, if $@p(c.g)$, this does not imply that c's variable g still has value `tt`, merely that at some point in the past, c's variable g was true. We also allow views of views. So for example $@c(@p(c.g))$ holds if c receives a message from p informing c that p has updated its view of $c.g$ to be true.

Views can be thought of as capturing knowledge in our model. If $@c(@p(c.g))$ then c knows that p knows that $c.g$ holds. The approach of modeling protocols using knowledge-based systems has gained much interest. Halpern and Zuck [5] use knowledged based reasoning to derive correctness proofs for a family of protocols. More recently Stulp [12] used a knowledge based algorithm to reason about the Internet protocol TCP.

3.2 Protocol Rules

The behaviour of each process within a particular class is defined by a set of *protocol rules*. Each rule consists of a pre-condition and a post-action. Let **R** be a rule for a class of phone processes P, and p be a phone process from that class. The pre-condition of **R** makes assertions over p's local state variables together with p's *view* of remote process variables. If the pre-condition of rule **R** holds at p, we say **R** is applicable at p and then **R**'s post-action may happen changing the local state at p. When p's local state is updated in the post-action of a rule, messages are sent to any process that maintains a view of p's state, enabling them to update their view of p, when these messages arrive. For example, suppose we have a system consisting of a single process p and a single process c. To express the behaviour that p's variable $p.g$ may move to state `tt`, from initial state `ff`, if it believes c's state variable g to be in state `tt`, we write the following rule.

$$\mathbf{R}(p)\ \frac{g = \mathtt{ff} \wedge @p(c.g)}{g := \mathtt{tt}}$$

The pre-condition of this rule is a conjunction of two clauses, that the process executing the rule, p, has state variable g set to `ff`, and that that p *views* c with its internal state variable $c.g$ set to `tt`. If this is the case p can execute the post-action and update its internal state variable $p.g$ to `tt`. In general a post-action may contain more than one assignment. It is assumed that all the assignments in the post-action are performed atomically and are ordered from left to right.

3.3 Environment Rules

If p has a view of variable g at a remote process c and c assigns a value `tt`, by executing an assignment in the post-action of a rule, then a message is sent to p informing p that $c.g$ was updated. At the point the message arrives and is delivered[1] to p, p's view of $c.g$ is updated. That is $@p(c.g = \mathtt{tt})$ now holds at p. Again we model this behaviour using rules. We call these rules *environment rules* rather than protocol rules.

If process p updates its view of $c.g$ to `tt` we write $@p(c.g := \mathtt{tt})$ in the post-action of the environment rule. A typical rule for updating p's view of a process c is as follows.

$$\mathbf{UV}\ \frac{c.g = \mathtt{tt} \wedge @p(\neg c.g)}{@p(c.g := \mathtt{tt})}$$

Unlike the protocol rules, this rule does not take place at any particular process. It is used to model the flow of information caused when messages are sent.

3.4 Message Exchange in Our Protocols

In the protocols we study connection based message passing is used. The protocols that implement connection based message passing are examples of transport

[1] The term "delivered" is often used to mean that the incoming message is processed rather than just residing in a buffer on the input device.

layer protocols in the ISO OSI reference model [8] for example the widely used Transfer Control Protocol (TCP) [3]. In these protocols a connection between sender and receiver is set up and an initial message sequence number is agreed upon. The sender then sends messages tagged with sequence numbers. The receiver sends back acknowledgements to these messages with matching sequence numbers. If an acknowledgement is not received within a certain time the sender resends the message. In this way a reliable message passing channel is implemented from sender to receiver. This method allows a byte stream to be sent without error. If two back to back channels are created in this way, a sender can reliably send a message to a receiver on one channel *and reliably* receive an acknowledgement to this message on the other.

We model this in our views model by not only updating the view of a receiving process of a senders variable, but also updating the view at the sender of the view at the receiver of the senders variable. Suppose we have two processes, a sender s and a receiver r. s has a viewable variable x, r maintains a view of $s.x$ and s maintains a view of r's view of $s.x$. To model s sending x to r *and* r acknowledging this with a receipt we write $@r(s.x := \mathtt{tt}) \wedge @s(@r(s.x = \mathtt{tt}))$.

3.5 Global State and Executions

In our simple example we are interested in modeling a phone process p, a message center c, and a merchant process m. A *protocol configuration* C can be modeled as a triple of internal states and views, with one entry for p, c and m respectively. We denote this $\langle p, c, m\rangle$, where each p, c and m represent the internal state and view of the phone, message center and merchant respectively.

The system takes a step whenever a process within the composition takes a step by executing a protocol rule. Thus if $\mathbf{R}(p)$ is applied to process p which we can write as $p \xrightarrow{\mathbf{R}(p)} p'$[2], then $\mathsf{C} \xrightarrow{\mathbf{R}(p)} \mathsf{C}'$ where $\mathsf{C} = \langle p, c, m\rangle$ and $\mathsf{C}' = \langle p', c, m\rangle$. A process (and therefore the system configuration) can also take a step when its view is updated through the application of an environment rule.

An *execution* , therefore is a possibly infinite evolution of system configurations.

$$\mathsf{C}_0 \xrightarrow{\mathbf{R_1}} \mathsf{C}_1 \xrightarrow{\mathbf{R_2}} \ldots$$

where C_0 is the initial configuration obtained by composing processes in their initial states.

Sometimes the pre-condition of more than one rule might hold for a process allowing more than one rule to be applied at that process. If this is the case we allow both rules to be applied leading to two new configurations.

4 Modeling a Simple Transaction

As we saw in section 2 a simple transaction is carried out by the mobile phone user making a request for some goods to a merchant via a message center. The

[2] Sometimes we omit the name of the process that the rule is applied to.

merchant then replies with a charging message to the phone containing the goods via the message center. In our simple example we model the goods as a one bit value. In reality this may consist of a software license key, a mobile phone ringtone or perhaps a password for web site content.

We first consider the internal state of processes in our system and then go on to examine the rules that drive the protocol. The state of the phone process will consist of two boolean variables and a credit counter which is an integer variable. A request variable $p.r$ initially set to $\texttt{ff}$ is used to convey a request to the merchant for some goods. Setting this request variable models the user texting a keyword to a short code. The goods variable $p.g$ is initially $\texttt{ff}$ denoting that the single bit good has not yet been received. The credit counter $p.b$ is set to 0 or 1 to show that the mobile phone account has enough credit to receive one charging message. The phone process also keeps a view of the message center's goods variable in our notation $@p(c.g)$.

The message center c is a little more complicated. It stores a boolean variable $c.r$ which is initially $\texttt{ff}$ reflecting that p has not yet requested goods. It also stores $@c(p.r)$, c's view of $p.r$. It stores boolean variable $c.g$ which are the goods ready to ship to p and also $@c(m.g)$ a view of the merchant's goods variable. $c.b$ is an integer counter which stores the money the merchant has earned from sending charging messages. The message center stores $@c(@p(c.g))$ which is c's view of p's view of $c.g$. Finally, the message center holds a timeout variable t which becomes true if the message cannot be sent within a certain time.

The merchant is relatively simple. It keeps a view of the message centers request variable $@m(c.r)$ and also some goods to ship $m.g$. Table 1 summarises the state of the processes.

We now examine the rules that drive our protocol. The first rule **REQ** is very simple it just allows a phone to make a request for some goods for the first time.

Table 1. The state stored at the phone, message center and merchant processes.

state of p		Initial value
$p.r \in \{\texttt{tt}, \texttt{ff}\}$	p is requesting goods	$\texttt{ff}$
$p.g \in \{\texttt{tt}, \texttt{ff}\}$	p's goods	$\texttt{ff}$
$@p(c.g) \in \{\texttt{tt}, \bot\}$	p's view of c's goods	$\bot$
$p.b \in \{0, 1, \ldots\}$	p's bank account	1 or 0
state of c		
$c.r \in \{\texttt{tt}, \texttt{ff}\}$	c's request for goods	$\texttt{ff}$
$@c(p.r) \in \{\texttt{tt}, \bot\}$	c's view of $p.r$	$\bot$
$@c(m.g) \in \{\texttt{tt}, \bot\}$	c's view of m's goods	$\bot$
$c.g \in \{\texttt{tt}, \texttt{ff}\}$	c's goods for p	$\texttt{ff}$
$c.b \in \{0, 1, \ldots\}$	c's bank account for m	0
$@c(@p(c.g)) \in \{\texttt{tt}, \texttt{ff}, \bot\}$	c's view of p's view of $c.g$	$\bot$
$c.t \in \{\texttt{tt}, \texttt{ff}\}$	c's has timed out	$\texttt{ff}$
state of m		
$m.g \in \{\texttt{tt}, \texttt{ff}\}$	m's goods for c	$\texttt{ff}$
$@m(c.r) \in \{\texttt{tt}, \bot\}$	m's view of $c.r$	$\bot$

Next **UV1** is an environment rule that propagates this request to the message center.

$$\mathbf{REQ}(p)\ \frac{\neg r}{r := \mathtt{tt}}\quad \mathbf{UV1}\ \frac{@c(\neg p.r) \wedge p.r}{@c(p.r := \mathtt{tt})}$$

Once the message center views this request it can set it's own request variable $c.r$ to propagate the request to the merchant. **UV2** is an environment rule that updates the merchants view of this request.

$$\mathbf{PROP}(c)\ \frac{@c(p.r) \wedge \neg r}{r := \mathtt{tt}}\quad \mathbf{UV2}\ \frac{@m(\neg c.r) \wedge c.r}{@m(c.r := \mathtt{tt})}$$

The merchant once it views the request can service the request and provide some goods. Again **UV3** updates the view as required.

$$\mathbf{SER}(m)\ \frac{@m(c.r) \wedge \neg g}{g := \mathtt{tt}}\quad \mathbf{UV3}\ \frac{@c(\neg m.g) \wedge m.g}{@c(m.g := \mathtt{tt})}$$

Once the message center views that goods are to be sent it attempts to send the goods to the phone user. The merchant's bank is credited with one unit.

$$\mathbf{SEND}(c)\ \frac{@c(m.g) \wedge \neg g}{g := \mathtt{tt} \wedge b{+}{+}}$$

Two possible update view rules are now possible to propagate the message to the phone user. If the phone has enough credit to receive a message the next rule applies. Notice the message is propagated to the phone only if it has enough credit. Furthermore, in the post-action of the rule, the $@c(@p(c.g))$ is updated reflecting that the phone has acknowledged the receipt of this message.

$$\mathbf{UV4}\ \frac{@p(\neg c.g) \wedge c.g \wedge p.b > 0 \wedge @c(@p(c.g = \bot)) \wedge \neg c.t}{@p(c.g := \mathtt{tt}) \wedge @c(@p(c.g := \mathtt{tt})) \wedge p.b{-}{-}}$$

The next update view rule covers the case that no credit was available at the phone so the message was not delivered. In this case we decrease the merchants bank account.

$$\mathbf{UV5}\ \frac{@p(\neg c.g) \wedge c.g \wedge p.b = 0 \wedge @c(@p(c.g = \bot)) \wedge \neg c.t}{c.b{-}{-} \wedge @c(@p(c.g := \mathtt{ff}))}$$

At last the goods message arrives at the phone.

$$\mathbf{RCV}(p)\ \frac{@p(c.g = \mathtt{tt}) \wedge \neg g}{g := \mathtt{tt}}$$

To make our model a little more realistic we allow a timeout rule to delete any pending message which have not been propagated to the phones. This rule allows us to model the situation where a phone has been out of coverage for so long that a message waiting to be delivered to it is deleted by the message center.

Notice in the post action c's view of p's view is set to false because c knows that p will not receive the goods as they are now not going to be sent.

$$\mathbf{DELETE}(c)\ \frac{@c(@p(c.g = \bot)) \wedge g \wedge t}{b{-}{-} \wedge @c(@p(c.g := \mathtt{ff}))}$$

$$\mathbf{TIMEOUT}(c)\ \frac{\neg t \wedge g \wedge @c(@p(c.g := \bot))}{t := \mathtt{tt}}$$

4.1 An Execution

As we saw before an execution is a sequence of configuration changes by the application of rules. A configuration $\langle p, c, m\rangle$ has the form

$$\langle (p.r, p.g, @p(c.g), p.b),\ (c.r, @c(p.r), @c(m.g), c.g, c.b, @c(@p(c.g)), c.t),\ (m.g, @m(c.r))\rangle$$

We can now write a possible execution of our transaction as follows. We underline the last state change to happen in the configuration.

$$\langle (\mathtt{ff}, \mathtt{ff}, \bot, 1),\ (\mathtt{ff}, \bot, \bot, \mathtt{ff}, 0, \bot, \mathtt{ff}),\ (\mathtt{ff}, \bot)\rangle \xrightarrow{\mathbf{REQ}(p)} \quad (1)$$
$$\langle (\underline{\mathtt{tt}}, \mathtt{ff}, \bot, 1),\ (\mathtt{ff}, \bot, \bot, \mathtt{ff}, 0, \bot, \mathtt{ff}),\ (\mathtt{ff}, \bot)\rangle \xrightarrow{\mathbf{UV1}} \quad (2)$$
$$\langle (\mathtt{tt}, \mathtt{ff}, \bot, 1),\ (\mathtt{ff}, \underline{\mathtt{tt}}, \bot, \mathtt{ff}, 0, \bot, \mathtt{ff}),\ (\mathtt{ff}, \bot)\rangle \xrightarrow{\mathbf{PROP}(c)} \quad (3)$$
$$\langle (\mathtt{tt}, \mathtt{ff}, \bot, 1),\ (\underline{\mathtt{tt}}, \mathtt{tt}, \bot, \mathtt{ff}, 0, \bot, \mathtt{ff}),\ (\mathtt{ff}, \bot)\rangle \xrightarrow{\mathbf{UV2}} \quad (4)$$
$$\langle (\mathtt{tt}, \mathtt{ff}, \bot, 1),\ (\mathtt{tt}, \mathtt{tt}, \bot, \mathtt{ff}, 0, \bot, \mathtt{ff}),\ (\mathtt{ff}, \underline{\mathtt{tt}})\rangle \xrightarrow{\mathbf{SER}(m)} \quad (5)$$
$$\langle (\mathtt{tt}, \mathtt{ff}, \bot, 1),\ (\mathtt{tt}, \mathtt{tt}, \bot, \mathtt{ff}, 0, \bot, \mathtt{ff}),\ (\underline{\mathtt{tt}}, \mathtt{tt})\rangle \xrightarrow{\mathbf{UV3}} \quad (6)$$
$$\langle (\mathtt{tt}, \mathtt{ff}, \bot, 1),\ (\mathtt{tt}, \mathtt{tt}, \underline{\mathtt{tt}}, \mathtt{ff}, 0, \bot, \mathtt{ff}),\ (\mathtt{tt}, \mathtt{tt})\rangle \xrightarrow{\mathbf{SEND}(c)} \quad (7)$$
$$\langle (\mathtt{tt}, \mathtt{ff}, \bot, 1),\ (\mathtt{tt}, \mathtt{tt}, \mathtt{tt}, \underline{\mathtt{tt}}, \underline{1}, \bot, \mathtt{ff}),\ (\mathtt{tt}, \mathtt{tt})\rangle \xrightarrow{\mathbf{UV4}} \quad (8)$$
$$\langle (\mathtt{tt}, \mathtt{ff}, \underline{\mathtt{tt}}, \underline{0}),\ (\mathtt{tt}, \mathtt{tt}, \mathtt{tt}, \mathtt{tt}, 1, \underline{\mathtt{tt}}, \mathtt{ff}),\ (\mathtt{tt}, \mathtt{tt})\rangle \xrightarrow{\mathbf{RCV}(p)} \quad (9)$$
$$\langle (\mathtt{tt}, \underline{\mathtt{tt}}, \mathtt{tt}, 0),\ (\mathtt{tt}, \mathtt{tt}, \mathtt{tt}, \mathtt{tt}, 1, \mathtt{tt}, \mathtt{ff}),\ (\mathtt{tt}, \mathtt{tt})\rangle \quad (10)$$

In fact, states (1) to (8) are the same for all executions because there is exactly one rule that can be applied at every transition. From state (8) depending on the initial value of $p.b$ either **UV4**, **UV5** or **TIMEOUT** can happen. The case of **UV4** is dealt with above. The case of **UV5** forms the last transition. The case **TIMEOUT** must be before **UV4** or **UV5** and not before state (7). After **TIMEOUT**, **DELETE** is applied and is the last applicable rule. Figure 2 is the transition graph of the behaviour of the system.

In our example in the initial state $p.b = 1$ and $c.b = 0$ and in the final state $p.b = 0 \wedge p.g = \mathtt{tt} \wedge m.b = 1$. We see our phone user used one credit to buy a single bit good from the merchant who acquired a single credit. Suppose however we started off with our phone being void of any credits *i.e.* $p.b = 0$. We see then that the last rule $\mathbf{RCV}(p)$ could never be applied and therefore the goods will not be delivered. We will return to these points later.

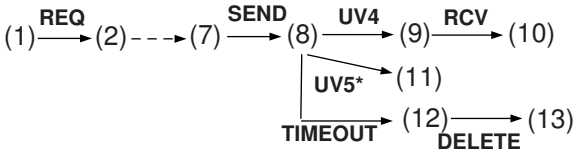

Fig. 2. The labeled transition system for the simple transaction protocol.

5 Goods and Money Atomicity

Goods and Money atomicity were defined in [6], they provide the basis of some useful properties that we would like to show hold for the protocols we examine.

- Money atomicity states that, "Money should be neither created nor destroyed in electronic commerce protocols."
- Goods atomicity states that, "A merchant should receive payment if and only if the consumer receives the goods."

Suppose we want to show that our mobile transaction has the desirable property of goods atomicity. That is, in any execution, a phone receives goods if and only if the phone pays a credit. To show this we need to verify that in every execution $p.g := \mathtt{tt}$ happens if and only if $p.b--$ happens. More generally, for any property Φ we write $\mathsf{C} \models \Phi$ if property Φ holds of a configuration C.

Since temporal logic expresses the capabilities over time, we are able to determine properties of executions by determining if a temporal property holds of the initial configuration. The temporal logic we use to express properties is a sub-logic of the widely used computation tree temporal logic, CTL due to Clarke, Emerson and Sistla [1]. Infact, a selection of other logics could have been used instead for example LTL.

Let Z range over basic propositions. A basic proposition details information about variables and views of variables of a configuration: for example, $p.g = \mathtt{tt}$, $c.b = 0$ or $@p(c.g = \mathtt{tt})$. We also allow arithmetic operations: for example $p.b+c.b = 1$. Given a basic proposition Z we let $C(Z)$ be the set of configurations where Z is true. A property in our logic can now be expressed as $\Phi ::= \mathtt{tt} \mid Z \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathrm{AG}\Phi \mid \mathrm{AF}\Phi$

We write $\mathsf{C} \to \mathsf{C}'$ to mean there exists a rule $\mathbf{R}$ where $\mathsf{C} \xrightarrow{\mathbf{R}} \mathsf{C}'$. The definition of satisfaction between a configuration C_0 and a formula proceeds by induction on the formula.

$$
\begin{array}{ll}
\mathsf{C}_0 \models \mathtt{tt} & \\
\mathsf{C}_0 \models Z & \text{iff } \mathsf{C}_0 \in C(Z) \\
\mathsf{C}_0 \models \neg\Phi & \text{iff } \mathsf{C}_0 \not\models \Phi \\
\mathsf{C}_0 \models \Phi \wedge \Psi & \text{iff } \mathsf{C}_0 \models \Phi \text{ and } \mathsf{C}_0 \models \Psi \\
\mathsf{C}_0 \models \mathrm{AF}(\Phi) & \text{iff for all executions } \mathsf{C}_0 \longrightarrow \mathsf{C}_1 \longrightarrow \ldots \\
 & \quad \text{there is } i \geq 0 \text{ with } \mathsf{C}_i \models \Phi \\
\mathsf{C}_0 \models \mathrm{AG}(\Phi) & \text{iff for all executions } \mathsf{C}_0 \longrightarrow \mathsf{C}_1 \longrightarrow \ldots \\
 & \quad \text{for all } i \geq 0,\ \mathsf{C}_i \models \Phi
\end{array}
$$

We let $\mathtt{ff}$ abbreviate $\neg\mathtt{tt}$, $\Phi \vee \Psi$ abbreviate $\neg(\neg\Phi \wedge \neg\Psi)$ and $\Phi \Rightarrow \Psi$ abbreviate $\neg\Phi \vee \Psi$ and $\Phi \Leftrightarrow \Psi$ iff $\Phi \Rightarrow \Psi$ and $\Psi \Rightarrow \Phi$.

Strong liveness properties are expressed using $AF\Phi$. For example if Z is the proposition $p.g = \mathbf{1}$, "The phone receives the goods", then formula $\mathrm{AF}Z$ means in all executions the phone eventually receives the goods.

Goods atomicity therefore can be expressed as

$$p.b = k \Rightarrow \mathrm{AF}((k = 1 \wedge p.b = k - 1) \Leftrightarrow p.g = \mathtt{tt}),\ k \in \{0, 1\}$$

That is in all futures we eventually reach a situation where we *only* pay for goods that are received. Money atomicity can also be expressed. Let Y be $p.b + c.b = k,\ k \in \{0, 1\}$, informally "The center's bank account for the merchant and the phone's bank account sums to the value k". We can express money atomicity as $Y \Rightarrow \mathrm{AF}(\mathrm{AG}(Y))$.

This formula states if Y holds then in all executions we eventually reach a point where Y holds again and continues to do so forever. This property takes into account that Y may fail to be true for a time (e.g. after rule $\mathbf{SEND}(m)$ but before rule $\mathbf{RCV}(p)$.), but in all futures Y is eventually restored.

Fact 1. In any execution of our simple protocol both goods and money atomicity hold.

Proof. The proof is straightforward by examining the possible executions of the system. There are only three possible different executions see Fig. 2. Clearly Money and Goods atomicity holds in each execution. □

6 Systems of Arbitrary Numbers of Phone Processes

In this section we enrich our model to include an arbitrary number of phone processes. We do this by extending the state at the message center and by adding the extra phone processes. In order to keep the model simple we only consider the second half of the protocol and omit the part of the protocol where a phone requests some goods. This is done by removing rules **REQ**, **UV1**, **PROP**, **UV2**, **SER** and **UV3**. We now also remove the merchant process as it is no-longer required. The protocol starts when the message center sends some goods using the **SEND** rule. We weaken the pre-condition of this rule so it can spontaniously send without the requirement of a having received a request.

A configuration is now $\mathsf{C} = \langle c, p_1, \ldots, p_n \rangle$ a message center and n phone processes p_j for $j \in \{1, \ldots, n\}$. The state at the message center and at each phone process is now also changed as shown in Table 2.

The rules of our system are as follows where $j \in \{1, \ldots, n\}$

$$\mathbf{SEND}_j(c)\ \frac{\neg g_j}{g_j := \mathtt{tt} \wedge b_j{+}{+}}$$

$$\mathbf{UV4}_j\ \frac{@p_j(\neg c.g_j) \wedge c.g_j \wedge p_j.b > 0 \wedge @c(@p_j(c.g_j = \bot)) \wedge \neg c.t_j}{@p_j(c.g_j := \mathtt{tt}) \wedge @c(@p_j(c.g_j := \mathtt{tt})) \wedge p_j.b{-}{-}}$$

Table 2. The state stored at the phone, message center for a model with arbitrary phone processes p_j for $j \in \{1, \ldots, n\}$.

state of each p_j		Initial value	Colour
$p_j.g \in \{\mathtt{tt}, \mathtt{ff}\}$	p's goods	$\mathtt{ff}$	j
$@p(c.g_j) \in \{\mathtt{tt}, \bot\}$	p_j's view of c's goods for p_j	$\bot$	j
$p_j.b \in \{0, 1, \ldots\}$	p_j's bank account	1 or 0	j
state of c			
$c.g_j \in \{\mathtt{tt}, \mathtt{ff}\}$	c's goods for p_j	$\mathtt{ff}$	j
$c.b_j \in \{0, 1, \ldots\}$	c's bank for p_j	0	j
$@c(@p_j(c.g_j)) \in \{\mathtt{tt}, \mathtt{ff}, \bot\}$	c's view of p_j's view of $c.g_j$	$\bot$	j
$c.t_j \in \{\mathtt{tt}, \mathtt{ff}\}$	c's has timed out on p_j	$\mathtt{ff}$	j

$$\mathbf{UV5}_j \ \frac{@p_j(\neg c.g_j) \wedge c.g_j \wedge p_j.b = 0 \wedge @c(@p_j(c.g_j = \bot)) \wedge \neg c.t_j}{c.b_j{-}{-} \wedge @c(@p_j(c.g_j := \mathtt{ff}))}$$

$$\mathbf{RCV}(p_j) \ \frac{@p_j(c.g_j = \mathtt{tt}) \wedge \neg g}{g := \mathtt{tt}} \quad \mathbf{DELETE}_j(c) \ \frac{@c(@p_j(c.g_j = \bot)) \wedge g_j \wedge t_j}{b_j{-}{-} \wedge @c(@p_j(c.g_j := \mathtt{ff}))}$$

$$\mathbf{TIMEOUT}_j(c) \ \frac{\neg t_j \wedge g_j \wedge @c(@p_j(c.g_j := \bot))}{t := \mathtt{tt}}$$

Using these rules we automatically generated the transition system for systems with one and two phone processes. These systems can be seen in Fig. 3.

It is useful to partition a configuration into coloured parts. We use colour j to colour a variable within the configuration if it is associated with phone process j in the system otherwise a variable is not coloured. Table 2 shows how we set up colours in our configuration. It is also possible to colour the rules of our system. We say a rule is colour j if it only reads j-colour variables in it's pre-condition and only changes j-colour or no-colour variables in it's post-action. $\mathbf{SEND}_j(c)$ is therefore a j-colour rule.

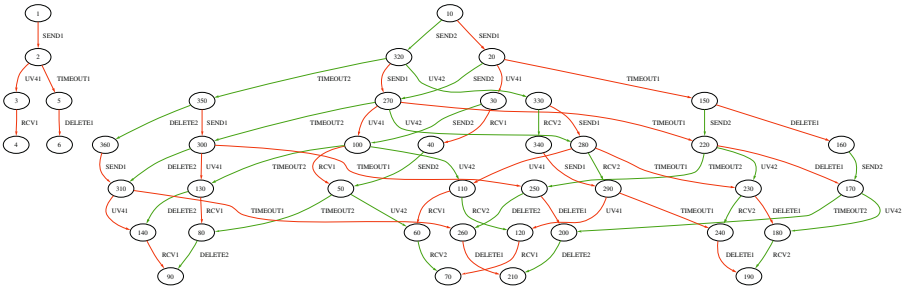

Fig. 3. One and Two Phone Process. Each phone starts with one credit. The graph for three phone processes (omitted) has 539 states.

6.1 Goods and Money Atomicity for Multiple Phones

We can define goods atomicity for the multiple-phone case as follows

$$\forall j \in \{1,\ldots,n\},\ p_j.b = k \Rightarrow \mathrm{AF}((p_j.b = 1 \wedge p_j.b = k-1) \Leftrightarrow p_j.g = \mathtt{tt}),\ k \in \{0,1\}$$

Goods atomicity, defined in this way, is really just a collection of smaller properties one for each process p_j all of which must hold. The smaller property for p_j is identical to the goods atomicity property we saw for phone process p in the single phone case of section 5. Let Φ_j be the property for p_j. It is clear Φ_j only makes reference to state in the configuration which has colour j.

In order to show goods atomicity holds we must show that Φ_j holds for all j. It is useful to this end to note (1) Each Φ_j only makes reference to variables in the configuration that have colour j. (2) Any rule that changes variables of colour j *only* changes variables of colour j. (3)Any rule that depend on colour j variables *only* depends on colour j variables.

To verify that Φ_j holds of the initial configuration we need only consider rules in our execution which are of colour j and which change configuration of colour j. This means we only need verify Φ_j in the very small transition system of a single phone process p_j and infer from this, and the properties above, that Φ_j holds for any j and thus goods atomiciy holds in the general case of arbitrary many phone processes. To see this note that the application of any non j-colour rule will not effect the validity of Φ_j nor will it change any state that could effect the application of a j-colour rule. This type of sate space reduction is an example of the partial order reduction techniques of [11].

Recall from section 5 we defined money atomicity as $Y \Rightarrow AF(AG(Y))$ where Y is $p.b + c.b = k$, $k \in \{0,1\}$. For the n-phone case we can generalise this. Let Y_j be $p_j.b + c.b_j = k_j$, then money atomicity can be defined as

$$\forall j \in \{1,\ldots,n\},\ Y_j \Rightarrow \mathrm{AF}(\mathrm{AG}(Y_j))$$

Using the same colouring arguments as before we only need to show this holds in the very small transition system of a single phone process p_j and infer that it holds for all j. Interestingly, we can use the linear nature of our property to infer a more global definition of money atomicity. We can use the fact that $\mathrm{AF}(\mathrm{AG}(\Phi_1)) \wedge \mathrm{AF}(\mathrm{AG}(\Phi_2)) \Leftrightarrow \mathrm{AF}(\mathrm{AG}(\Phi_1 \wedge \Phi_2))$ to show that

$$\sum p_j.b + c.b = k \Rightarrow \mathrm{AF}(\mathrm{AG}(\sum p_j.b + c.b = k))$$

where

$$k = \sum k_j \text{ and } c.b = \sum c.b_j$$

7 Multiple Charging Messages

We now return to the case of a single phone process but consider multiple charging messages. As we saw in the simple transaction in section 2 the delivery of a

charging message to a phone debits the phone account by one credit. The value of a credit is fixed by the mobile operator and government regulators, in most countries. Currently, the highest charging value in the UK is £1.50. Unfortunately, many goods have a greater value than a single credit. As a consequence merchants send two (or more) messages in order to deliver the goods. We will now extend our model to capture this situation and examine the consequences.

We replace the single variable g, and all its views, at each process with two variables g^1 and g^2. We also include two timeout variables t^1 and t^2 to replace the single t at the message center. The state at the message center then becomes.

$$(c.r,@c(p.r),@c(m.g^1),@c(m.g^2),c.g^1,c.g^2,c.b,@c(@p(c.g^1)),@c(@p(c.g^2)),c.t^1,c.t^2)$$

The rules are modified to accommodate these extra variables. In the following we use i to be either 1 or 2 giving a pair of rules in each case.

$$\mathbf{SER^i}(m)\ \frac{@m(c.r) \wedge \neg g^i}{g^i := \mathtt{tt}} \quad \mathbf{UV3^i}\ \frac{@c(\neg m.g^i) \wedge m.g^i}{@c(m.g^i := \mathtt{tt})}$$

$$\mathbf{SEND^i}(c)\ \frac{@c(m.g^i) \wedge \neg g^i}{g^i := \mathtt{tt} \wedge b{+}{+}}$$

$$\mathbf{UV4^i}\ \frac{@p(\neg c.g^i) \wedge c.g^i \wedge p.b > 0 \wedge @c(@p(c.g^i = \bot)) \wedge \neg c.t^i}{@p(c.g^i := \mathtt{tt}) \wedge @c(@p(c.g^i := \mathtt{tt})) \wedge p.b{-}{-}}$$

$$\mathbf{UV5^i}\ \frac{@p(\neg c.g^i) \wedge c.g^i \wedge p.b = 0 \wedge @c(@p(c.g^i = \bot)) \wedge \neg c.t^i}{c.b{-}{-} \wedge @c(@p(c.g^i := \mathtt{ff}))}$$

$$\mathbf{RCV}(p)\ \frac{@s(c.g^i = \mathtt{tt}) \wedge \neg g^i}{g^i := \mathtt{tt}} \quad \mathbf{DELETE^i}(c)\ \frac{@c(@p(c.g^i = \bot)) \wedge g^i \wedge t^i}{b{-}{-} \wedge @c(@p(c.g^i := \mathtt{ff}))}$$

$$\mathbf{TIMEOUT^i}(c)\ \frac{\neg t^i \wedge g^i \wedge @c(@p(c.g^i := \bot))}{t^i := \mathtt{tt}}$$

Some much more interesting executions are now possible. It is a relatively simple task to automatically generate a state transition system representing all possible executions from a particular initial state from the rules given. Using Microsoft's C# .Net framework we created a class for each participant and coded the rules as class methods. A configuration class was built by composing the phone, message center and merchant classes. By using a breadth first search algorithm we automatically generated the complete transition system. Using Graphviz [4] we rendered the transition system of Fig. 4. Initially we gave the phone just a single credit.

Many merchants use the technique of sending two charging messages for goods to charge twice the fixed charge. A question now arises. Should the goods be sent in one message, both messages, or split into two (if this is possible) and be sent half in one message and half in the other? Different goods providers choose different methods. It turns out that none of these methods guarantees goods atomicity.

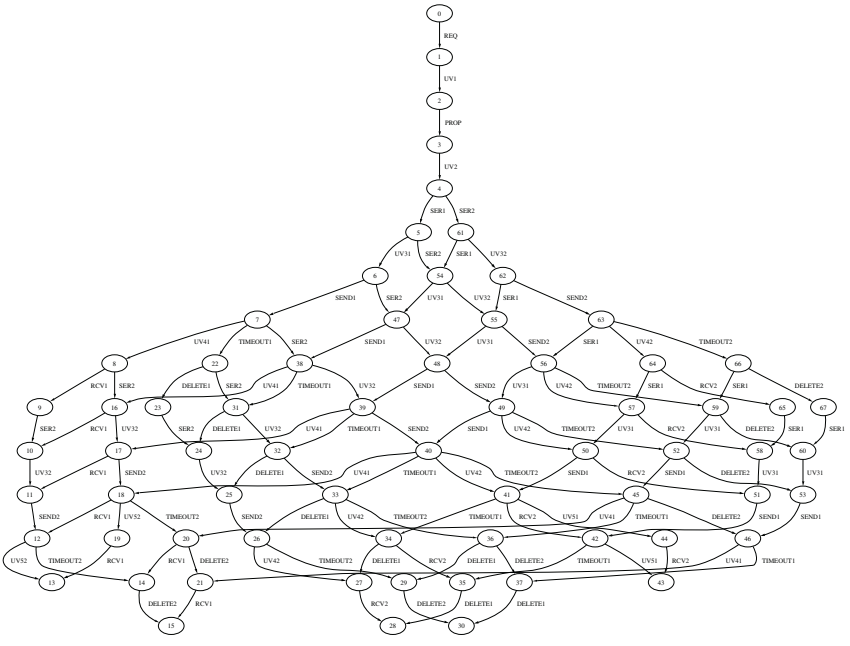

Fig. 4. The transition system where two charging messages convey goods to a phone with only a single credit.

Clearly, if the goods are sent in both messages and the phone starts off with a credit balance of 1, goods atomicity is violated since the phone user can acquire the goods by spending just one credit. Similarly, if the goods are split over the two messages [3], and the phone has only one credit, goods atomicity is again violated. In this case the merchant will earn a credit without supplying the goods. Since the order in which the messages arrive cannot be specified, sending the goods in one of the two messages is also not an option if goods atomicity is to hold.

We now use our model to show more formally that goods atomicity fails. Suppose we send the goods in both messages then we can let $Z = p.g^1 \vee p.g^2$. Z holds if the goods are delivered either in one message or the other. Goods atomicity can be stated as follows.

$$p.b = k \Rightarrow \mathrm{AF}((Z \Rightarrow k > 1 \wedge p.b = k - 2) \wedge (p.b < k \Rightarrow Z)),\ k \in \{0, 1, 2\}$$

Note we need to be a little more accurate in describing goods atomcity now than before. We have replaced $Z \Leftrightarrow k = 1 \wedge p.b = k - 1$ with $(Z \Rightarrow k > 1 \wedge p.b = k-2) \wedge (p.b < k \Rightarrow Z)$. The reason for this is that we want capture the case where the phone user spends just one credit but does not receive the goods, while still capturing the case that the goods are received without being fully paid for.

[3] Some goods providers send a user name in one message and a password in another which can be used to gain access to premium web content.

By examining Fig. 4 we see that the path through states 0, 1, 2, 3, 4, 61, 62, 55, 48, 39, 40, 41, 42, 43 violates goods atomicity. Initially (state 0) $p.b = 1$. Let us examine the final state 43 more carefully. In state 43 we see that the local state at p is

$$r = \texttt{tt},\ g^1 = \texttt{ff},\ g^2 = \texttt{tt},\ @p(c.g^1) = \bot,\ @p(c.g^2) = \texttt{tt}, b = 0$$

the local state at c is

$$c.r = \texttt{tt},\ @c(p.r) = \texttt{tt},\ @c(m.g^1) = \texttt{tt},\ @c(m.g^2),\ c.g^1 = \texttt{tt},\ c.g^2 = \texttt{tt},$$
$$c.b = 1,\ @c(@p(c.g^1)) = \texttt{ff},\ @c(@p(c.g^2)) = \texttt{tt},\ c.t^1 = \texttt{ff},\ c.t^2 = \texttt{ff}$$

and the local state at the merchant m is

$$m.g^1 = \texttt{tt},\ m.g^2 = \texttt{tt},\ @m(c.r) = \texttt{tt}$$

In state 43 $p.b = 0$ and $Z = p.g^1 \vee p.g^2$ holds but $k > 2$ fails thus $Z \Rightarrow k > 1 \wedge p.b = -1$ fails so goods atomicity is violated.

Suppose we change the proposition Z to be $p.g^1 \wedge p.g^2$. This represents the scenario where goods are split up over each of the messages. The same execution provides a counter example. Initially $p.b = 1$ holds but in the final state 43, $p.b < 1$ but Z now fails because $p.g^1 = \texttt{ff}$. The phone user parts with half the money but did not receive the goods (only half of them).

It is also very easy to show that if Z is defined as just $Z = p.g^1$, where the goods are sent in one particular message an execution can be found where goods atomicity fails. The same is of course true for $Z = p.g^2$.

A possible remedy to these problems is to have the message center alert the merchant of failure of delivery of messages. The merchant could then take appropriate action. For example the undelivered message could be resent or the goods invalidated. The main problem with this is that in the case of a timeout event the notification of delivery failure may be several days later. It might be too late to invalidate the goods at this point, for example if the goods were access to web content which might have alrady taken place.

8 Conclusions

We saw how mobile telephones can be used to acquire goods where the payment of these goods is made via the users telephone bill these transactions were modeled using a views model. We expressed important safety properties, namely goods and money atomicity. and showed that these properties hold in our model of a simple transaction. Our model produces a labeled transition system so we could have made use of the many excellent model checking tools available [7][2]. We extended the model for arbitrary numbers of phone processes and we verified properties in these much larger systems using a technique to reduce the size of the resulting transition system. Reverting back to our initial model with a single phone process we enriched it to include multiple charging messages. In this model goods atomicity fails.

Although our techniques are applicable for many different types of protocols, the mobile commerce example in this paper provided us with novel subject matter. Due to it's infancy using it in specific ways exhibits some early defects which we were able to highlight. Our views model provides a rules based operational semantics with a flavour of knowledge-based reasoning. This allowed us to retain the advantages of automated model checking techniques with an ability to model and reason about knowledge. It also allowed us to develop techniques to combat the well known state explosion problem when model checking.

In the future we intend to investigate further methods for reducing state spaces in a more general model. In particular we believe by providing a formal syntax for our protocol and environment rules we will be able to derive general results for models of arbitrary numbers of processes.

The authors would like to thank Cormac Long, SMPP Protocol Specialist http://www.smsforum.net for his help and advice when writing this paper.

References

1. E. Clarke, E. Emerson, and A.Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
2. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
3. D.E.Comer. *Internetworking with TCP/IP*. Prentice–Hall, Upper Saddle River, NJ 07458, 1995. Volume 1.
4. John Ellson, Emden Gansner, Eleftherios Koutsofios, and Stephen North. Graphviz. http://www.research.att.com/sw/tools/graphviz.
5. Joseph Y. Halpern and Lenore D. Zuck. A little knowledge goes a long way: Knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM*, 39(3):449–478, July 1992.
6. N. Heintze, J. D. Tygar, J. Wing, and H. C. Wong. Model checking electronic commerce protocols. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 147–164, November 1996.
7. Gerard J. Holzmann. The Spin model checker. *IEEE Transactions on Software Engineering*, 23(5):279–95, May 1997.
8. J.D.Day and H. Zimmermann. The OSI reference model. In *Proceedings of the IEEE*, volume 71, pages 1334–1340. IEEE Comput. Soc. Press, December 1983.
9. Tim Kempster, Colin Stirling, and Peter Thanisch. A more committed quorum-based three phase commit protocol. In *International Symposium on Distributed Computing*, pages 246–257, 1998.
10. Nancy A. Lynch. *Distributed Algorithms*. Morgan-Kaufmann, San Francisco, CA, 1993. chapter 8.
11. D. Peled. Ten years of partial order reduction. *Lecture Notes in Computer Science*, 1427, 1998.
12. Freek Stulp and Rineke Verbrugge. A knowledge-based algorithm for the internet transmission control protocol (tcp). In G. Bonanno and W. van der Hoek, editors, *Proceedings 4rd Conference on Logic and the Foundations of Game and Descision Theory (LOFT 4)*, 2000.