A Mechanism for Establishing Policies for Electronic Commerce

Naftaly H. Minsky* Victoria Ungureanu*

Department of Computer Science
Rutgers University
New Brunswick, NJ, 08903 USA
Phone: (908) 445-2085
Fax: (908) 445-0537
Email: {minsky,ungurean}@cs.rutgers.edu

Abstract

This paper introduces a mechanism for establishing policies for electronic commerce in a unified and secure manner. A commercial policy can be viewed as the embodiment of a contract between the principals involved in a certain type of commercial activity, and it may be concerned with such issues as: ensuring that a payment for services is refunded under specified circumstances; preventing certificates representing e-cash from being duplicated; ensuring that credit card numbers are used only for the transaction they are intended for; and, for certain socially sensitive transactions like the purchase of drugs, ensuring auditability by proper authorities.

Our mechanism is based on a previously published concept of law-governed interaction. It makes a strict separation between the formal statement of a policy, which we call a "law," and the enforcement of this law, which is carried our by a set of policy-independent trusted controllers*. A new policy under this scheme is created basically by formulating its law, and can be easily deployed throughout a distributed system. This mechanism enables a single agent to engage in several different activities, subject to disparate policies.*

Two examples policies are discussed here in detail: one ensures refundability *of payment, under certain circumstances; the other provides for payment by means of* non-copyable tickets*.*

1 Introduction

Research on electronic-commerce focused mainly on means for secure and efficient transfer of funds [15, 4, 14]. Such means are indeed necessary for e-commerce, but they are not sufficient. Commercial activities are not limited to simple exchange of funds and merchandise between a client and a vendor—they often consist of multi-step transactions, sometimes involving several participants, that need to be carried out in accordance to a certain policy. A commercial policy is the embodiment of a contract (which may be explicit or implicit) between the principals involved in a certain type of commercial activity—and it may be concerned with such issues as: ensuring that a payment for services is refunded under specified circumstances; preventing certificates representing e-cash from being duplicated; ensuring that credit card numbers are used only for the transaction they are intended for; and, for certain socially sensitive transactions like the purchase of drugs, ensuring auditability by proper authorities.

There is, of course, no universal policy. Many different policies are being employed in traditional commerce, and many will have to be employed in electronic commerce as well. For instance, a recent paper [6] identified several examples of policies for purchasing of information, using such things as *subscription*, *pay-per-view* and *pre-paid voucher*, which use different form of payment, employ different interaction protocols, and are suitable for different circumstances. The potential diversity of e-commerce policies creates a need for convenient and reliable means for defining, deploying, enforcing and using policies. It should, in particular, be possible for a single agent to engage in several different activities that are subject to different policies, as is the case for traditional commerce.

The currently prevailing method for establishing e-commerce policies is to build an interface that implements a desired policy, and distribute this interface among all who may need to operate under it. Unfortunately, such "manual" implementation of e-commerce

*Work supported in part by NSF grants No. CCR-9626577

policies is both unwieldy and unsafe. It is *unwieldy* in that it is time consuming and expensive to carry out, and because the policy being implemented by a given set of interfaces is obscure, being embedded into the code of the interface. A manually implemented policy is *unsafe* because it can be circumvented by any participant in a given commercial transaction, by modifying his interface for the policy. The first difficulty has been addressed recently in [6, 1], and the second has been addressed in [5], but we are not aware of any attempt to alleviate both of these difficulties together.

In this paper we introduce a mechanism, based on what we call "law-governed interaction" (LGI) [9], that can support a wide range of commercial policies in a unified and secure manner. Our mechanism makes a strict separation between the formal statement of a policy, which we call a "law," and the enforcement of this law, which is carried our by a set of policy-independent trusted *controllers*. A new policy under this scheme is created basically by formulating its law, which is easy to deploy and to use, and quite scalable.

We start, in Section 2, with a definition of our concept of a *policy*, illustrating it with an informal example of a policy that ensures refundability of payments—this policy is formalized in Section 4. We continue by presenting the implementation of this concept by a toolkit called Moses, illustrating it with another example policy that provide for payments by means of non-copyable tickets. In Section 3 we explain how the proposed mechanism can be made secure, and we conclude with a thought about some broader implications of this work.

2 The Concept of a Policy

Before defining our concept of a policy, we illustrate it by presenting an analogue of the *refund policies* often employed in traditional retail. This analogue is the following "money back" policy, which we denote by $\mathcal{MB}$.

> Let an order sent to a vendor have the form `order(specs,fee,eCash,gp)`, where `specs` specifies the required merchandise, `fee` is the sum the client is willing to pay for the merchandise, `eCash` is the digital certificate for `fee`, and `gp` is the "grace period" (say, in minutes) required by the client. It is the essence of this policy that the client be able to get his money back by sending the message `refund(specs)` to the vendor. The refund is to be granted if either the ordered item has not been shipped, or less than `gp` minutes have passed since the ordered merchandise arrived at the client[1].

Note that refund policies are effective in traditional commerce because the clients of large respectable vendors tend to trust them to honor their "money-back guarantee"—since such vendors value their reputation. But this kind of trust does not occur naturally in e-commerce because of the open nature of the Internet. This is why e-commerce policies must be *enforced* to be effective, and why they must be formalized. The following, then, is our definition of a policy for e-commerce.

Definition 1 *A policy $\mathcal{P}$ is a four-tuple $\langle \mathcal{M}, \mathcal{G}, \mathcal{CS}, \mathcal{L} \rangle$ where*

1. *$\mathcal{M}$ is the set of messages regulated by this policy—they are called $\mathcal{P}$-messages.*
2. *$\mathcal{G}$ is an open and distributed group of* agents *that are allowed to exchange $\mathcal{P}$-messages.*
3. *$\mathcal{CS}$ is a mutable set $\{\mathcal{CS}_x \mid x \in \mathcal{G}\}$ of what we call* control states*, one per member of the group $\mathcal{G}$.*
4. *$\mathcal{L}$ is an enforced set of* rules of engagement *that regulates: (a) the exchange of $\mathcal{P}$-messages between members of group $\mathcal{G}$, and (b) the changes in the control states of members of $\mathcal{G}$.*

We now elaborate on this definition, using the money back ($\mathcal{MB}$) policy as an example:

- The set $\mathcal{M}$ of $\mathcal{P}$-messages consist of all messages regulated by policy $\mathcal{P}$. Under the $\mathcal{MB}$ policy, for example, $\mathcal{M}$ would consists of all the messages exchanged during the purchase process, including the messages `order` and `refund` mentioned above.
- $\mathcal{G}$ is an open and distributed group of *agents* that are allowed to exchange $\mathcal{P}$-messages. In the case of the $\mathcal{MB}$ policy, the group $\mathcal{G}$ consists of all the vendors and clients participating in it. Group $\mathcal{G}$ is *open* in two (orthogonal) respects. First, the membership in $\mathcal{G}$ can change, subject to the law $\mathcal{L}$ of this policy. Second, nothing is assumed about the structure or behavior of the agents that belong to this group, except that they follow the rules of engagement prescribed by the policy. Each agent is viewed as a black box, which is able to send and receive $\mathcal{P}$-messages. In particular, nothing is assumed about the language the agents are programmed, if any—it may well be they are human agents.

[1]The return of the merchandise, if refund is requested, will be discussed later.

- The *control-state* $\mathcal{CS}_x$ of a given agent x is a set of attributes associated with this agent. These attributes—represented in the current implementation by Prolog-like terms— are used to structure the group $\mathcal{G}$, and to provide agents with *bundles of rights.* Some of the attributes of an agent have a predefined semantics, such as the term `self(n)` that defines `n` to be the unique identifier for this agent (unique within the group). In general, however, the semantics of attributes under a given policy is defined by the law of this policy. For instance, in the implementation of the *money back* policy, described in Section 2, a term `vendor` in the control state of an agent identifies it as one of the vendors.

- The *law* $\mathcal{L}$ consists of the *rules of engagement* that each agents should, individually, observe under a given policy. These rules deal with a certain kind of events that may occur at any agent in $\mathcal{G}$, prescribing the effect that any such event should have. More specifically, the law deals with the following two kinds of events that are regulated under Moses[2]:

  1. `sent(x,m,y)`—occurs when agent `x` sends a $\mathcal{P}$-message `m` addressed to `y`.
  2. `arrived(x,m,y)`—occurs when a $\mathcal{P}$-message `m` sent by `x` arrives at destination `y`.

  We assume no prior knowledge of, or control over, the occurrence of these *regulated events.* But the effect that any given event `e` would actually have is prescribed by the law $\mathcal{L}$ of the policy in question. This prescription, called the *ruling* of the law for this event, is a (possibly empty) sequence of *primitive operations* (discussed in Section 2.2) which are to be carried out as the immediate response to its occurrence. In the case of the $\mathcal{MB}$ policy, for example, $\mathcal{L}$ would state, in particular, that if a client `c` sends a `refund(specs)` after the grace period ended, `c`'s request is declined. The language in which $\mathcal{L}$ is written is described in Section 2.3.

Several clarifications of our concept of a policy are in order. First, we take a policy to have an independent existence, separate from the agents participating in it, and independent from any other policy. We provide means for an agent to *join* a given policy $\mathcal{P}$—subject to the law of this policy—which will enable this agent to send and receive $\mathcal{P}$-messages. A given agent may thus join several policies, and be active under them concurrently. Note that by definition, different policies cannot be in conflict because the exchange of $\mathcal{P}$-messages is governed solely by $\mathcal{L}_{\mathcal{P}}$, regardless of any other policy the agent in question might belong to.

[2]Note that Moses regulates some additional types of events described in [10, 11].

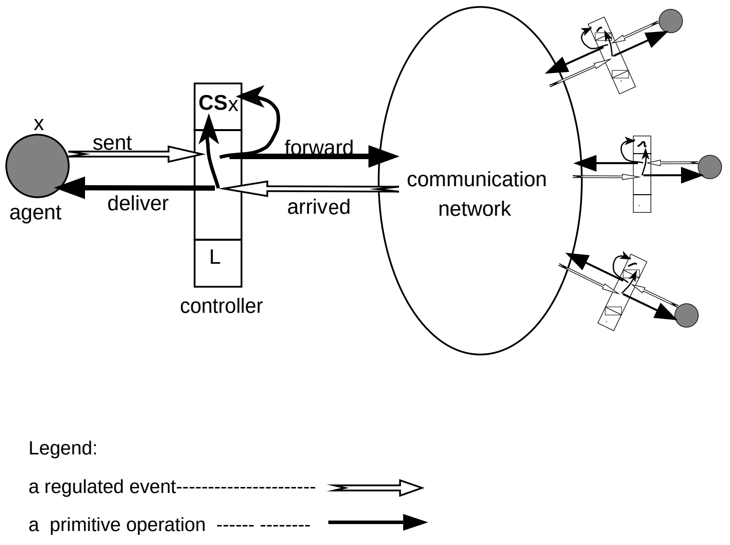
x
sent
CSx
forward
agent
deliver
arrived
communication network
L
controller
Legend:
a regulated event---------------------
a primitive operation ------ --------

Figure 1: Enforcement of the law

Second, although a policy can be formulated without detailed knowledge of the behavior of the prospective participants, an agent involved in a certain policy needs to be familiar with its law, and with the structure of its messages. But since the law is enforced, it cannot be violated even by an agent who does not know the law, or who willfully ignores it.

2.1 The Distributed Law-Enforcement Mechanism

The essential aspect of LGI is that it ensures that the law $\mathcal{L}$ of a policy $\mathcal{P}$ is strictly enforced, providing each agent in the policy-group in question, with the confidence that $\mathcal{L}$ cannot be violated by any of its peers.

The law of a policy is enforced by trusted entities called *controllers.* For any member `x` in a policy-group there is a controller logically placed between `x` and the communications medium, as is illustrated in Figure 1. All controllers have identical copies of the law $\mathcal{L}$—it is in this sense that the law is said to be global to the group—and each controller maintains the control states of the agents under its jurisdiction.

In principle controllers enforce the law in the following manner:

1. First, when a member `x` wishes to send a

$\mathcal{P}$-message `m` to a member `y`, `x` sends `m` to its assigned controller $\mathcal{C}_x$. This will generate a `sent(x,m,y)` event. The controller evaluates the ruling of the law $\mathcal{L}$ for this event and carries it out. If part of the ruling[3] is to forward a message `m'` to `y` then $\mathcal{C}_x$ would send `m'` to the controller $\mathcal{C}_y$ assigned to `y`.

For example, in the $\mathcal{MB}$-policy, if a client `c` sends the message `order(specs,fee,eCash,gp)`, addressed to a vendor `v`, through its assigned controller $\mathcal{C}_c$, it will trigger at $\mathcal{C}_c$ the event:

```
sent(c,order(specs,fee,eCash,gp),v).
```

The ruling for this event calls for storing the `eCash` certificate in the control state of `c`, and forwarding to `v` the message `order(specs,fee,gp)`. (The motivation for this ruling will become apparent in Section 4 where we present the law for the $\mathcal{MB}$ policy in full detail.)

2. Second, when `m'` arrives at $\mathcal{C}_y$, the controller assigned to the destination `y`, it generates an `arrived(x,m,y)` event. $\mathcal{C}_y$ computes and carries out the ruling of the law for this event. The message `m'` is then delivered to `y` if so required by the ruling.

   For example, in the $\mathcal{MB}$-policy when the message `order(specs,fee,gp)` arrives at $\mathcal{C}_v$ the controller assigned to `v`, it will trigger the event:

   ```
   arrived(c,order(specs,fee,gp),v).
   ```

   The ruling for this event calls for the delivery to `v` of the message `order(specs,fee,gp)` only if `v` is a `vendor`.

It should be pointed out that the confidence one has in the correct enforcement of the law at every agent depends on the assurance that all $\mathcal{P}$-messages are mediated by correctly implemented controllers, interpreting policy $\mathcal{P}$. The way to gain such an assurance is a security issue we will address in Section 3.

2.2 The Primitive Operations

The operations that can be included in the ruling of the law for a given regulated event `e` are called *primitive operations*. They are "primitive" in the sense that they can be performed *only* if thus authorized by the law. These operations include:

[3] Of course, this does not have to be the ruling of the law.

1. Operations that change the `CS` of the home agent. Specifically, we can perform the following operations: (1) `+t` which adds the term `t` to the control state; (2) `-t` which removes the term `t`; (3) `t1`$\leftarrow$`t2` which replaces term `t1` with term `t2`; (4) `incr(t(v),x)` which increments the value `v` of a term `t` with some quantity `x`; and (5) `dcr(t(v),x)` which decrements the value `v` of a term `t` with some quantity `x`.

2. Operation `forward(x,m,y)` sends the message `m` to $\mathcal{C}_y$, the controller assigned to `y`. The *sender* of the message is `x`, and the intended destination is `y`. (When a message thus forwarded to `y` arrives at $\mathcal{C}_y$, it would trigger at `y` the event `arrived(x,m,y)`.) The most common use of this operation is in a ruling for event `sent(x,m,y)`, where operation `forward` (with no arguments) simply completes the passing of the intended message.

3. Operation `deliver(x,m,y)` delivers the message `m` to `y`. The most common use of this operation is in a ruling for event `arrived(x,m,y)`, where operation `deliver` (with no arguments) simply delivers the arriving message to the home agent.

2.3 The Formulation of the Law

The function of $\mathcal{L}$ is to evaluate a ruling for any possible regulated-event that occurs at an agent with a given control-state. In our current model, $\mathcal{L}$ is represented by a very simple Prolog-like program. When this program $\mathcal{L}$ is presented with a goal `e`, representing a regulated event, and with a variable `CS`, representing the control-state of the home agent, it produces a list of primitive-operations representing the ruling of the law for this event. For the details of this formulation the reader is referred to [9], here we will only illustrate it with the following example.

Ticket Based Payment Policy—an Example Consider a distributed database that charges a fixed, and small, fee per query. It is well known that e-cash certificates are not suitable for small payments [4], because each such certificate needs to be created by a bank, and each needs to be validated by the issuer, for fear of duplication. Therefore, let the payment be carried out according to the following policy:

> A prospective client `c` can purchase from a designated ticket-seller a non-copyable `q`-valued ticket, where `q` is the number of queries this ticket is good for. A query submitted by `c` is processed only if `c` has a ticket

with non-zero value, and as a result the value of this ticket is reduced by one.

This, to be called $\mathcal{T}$-policy, is formalized by law $\mathcal{L}_\mathcal{T}$, displayed in Figure 2. This law consists of six rules. Each rule in this figure is followed with a comment (in italic), which together with the following discussion, should be understandable even for a reader not familiar with Prolog. We now describe this law by explaining how various operations are regulated by it.

First, by Rule $\mathcal{R}1$, a client `c`, wishing to buy a ticket with value `n` sends to the ticket seller `v`, the message `buyTicket(n,ec)`, where `ec` is a certificate for a sum of dollars in some form of electronic cash, like Ecash [14]. By Rule $\mathcal{R}6$, when such a message arrives at its destination it is delivered, only if the destination is an authorized ticket seller. The ticket seller may check the electronic certificate and if it is satisfied, it is expected to respond with a `giveTicket(n)` message (Rule $\mathcal{R}2$), otherwise it is expected to return the electronic certificate. By Rule $\mathcal{R}3$, when the client receives a `giveTicket(n)` message the term `ticket(n)` which serves as a representation of the ticket prescribed by $\mathcal{T}$-policy, is added to its control state, and the message is delivered to the client.

Next, by Rule $\mathcal{R}4$, a client `c` that has a ticket with positive value in his control state is allowed to present a query to a server of the database; the value of this ticket is automatically decreased by one unit. Finally, when a server responds to the query, the response is delivered to the client without further ado. (Rules $\mathcal{R}5$ and $\mathcal{R}6$)

The important aspects of this law are that it insures that: (1) only a ticket seller can inject tickets into the system, and (2) these tickets cannot be duplicated by clients, because they are kept by the controllers, and the law does not provide for their duplication. This is critical for the scalability of the mechanism, because a server may serve any query it receives *without having to check* the validity of the ticket. If servers are distributed, like in our case, the only solution we are aware of for preventing double spending is to have a centralized ticket checker, which tends to become a bottleneck [15].

Note that under this policy, the clients are at the mercy of both the ticket sellers and database servers. They send their payment with their order with no guarantee for either timely service or for the correct response. This problem can be alleviated by using the money-back policy discussed in Section 2 and implemented in Section 4.

Initially: A member may have in its control state attribute `ticketSeller` which authorizes it to sell tickets.

$\mathcal{R}1$.
```
sent(C,buyTicket(N,EC),V) :- do(forward).
```
A client request to buy `N` *ticket using an electronic certificate* `EC` *is forwarded to the destination.*

$\mathcal{R}2$.
```
sent(V,giveTicket(N),C) :-
      ticketSeller@CS, do(forward).
```
A request to give a ticket for `N` *uses is forwarded only if the sender has in its control state the term* `ticketSeller`.

$\mathcal{R}3$.
```
arrived(V,giveTicket(N),C) :-
      do(+ticket(N)), do(deliver).
```
When a ticket arrives at the destination it is added to the control state of the receiver.

$\mathcal{R}4$.
```
sent(C,query(Q),S) :-
      ticket(N)@CS,
      N>=1,do(dcr(ticket(N),1)), do(deliver).
```
A `query` *message is forwarded only if the sender has in its control state a ticket good for* `N` *uses. Also, the value of the ticket is reduced by one.*

$\mathcal{R}5$.
```
sent(_,M,_) :-
      not((M=giveTicket(_);M=query(_))),
      do(forward).
```
All messages —except `giveTicket` *and* `query`*— are forwarded to their destination.*

$\mathcal{R}6$.
```
arrived(_,M,_) :-
      M=buyTicket(_,_) →
                  (ticketSeller@CS,
      do(deliver)|
                  do(deliver)).
```
A `buyTicket` *message is delivered only if sent to a ticket seller. All other messages are delivered to their destination without further ado.*

Figure 2: Law $\mathcal{L}_\mathcal{T}$ of Ticket Policy

2.4 The Deployment of Policies, and their Scalability

A new policy $\mathcal{P}$ is introduced by creating a server that provides persistent storage for the law $\mathcal{L}$ of this policy, and the control-states of its members. This server is called the *secretary* of $\mathcal{P}$, to be denoted by $\mathcal{S}_\mathcal{P}$.

For an agent `x` to be able to exchange $\mathcal{P}$-messages under a policy $\mathcal{P}$, it needs to engage in a two-step connection protocol, illustrated by Figure 3. First, `x` needs to be associated with a generic controller, that knows nothing about any particular policy. This is

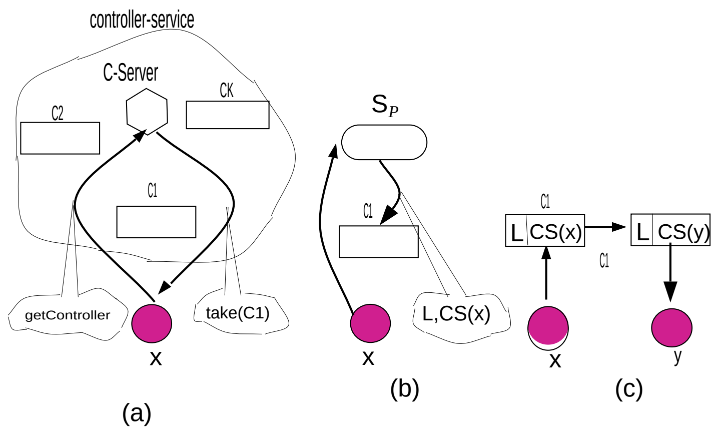

Figure 3: Deployment of a policy: (a) agent x asks a C-server for a trusted controller—it gets C1 in this case; (b) agent x asks $\mathcal{S}_\mathcal{P}$, the secretary of policy $\mathcal{P}$, to become an active member—causing C1 to be loaded with $\mathcal{L}$, the law of policy $\mathcal{P}$, and the control-state of x; (c) x can start communicating with another member y operating under the same policy.

to be done by sending a message to an agent called *C-service* that operates as a trusted name-server for controllers (discussed in the following section).

Second, `x` needs to send a `connect` message to $\mathcal{S}_\mathcal{P}$, asking to be admitted either as one of the existing members of the group, which already has a name and a control state of its own, or as a new member, requesting a generic control state. $\mathcal{S}_\mathcal{P}$ is likely to require authentication, particularly for an attempt to connect to an existing member, using some standard cryptographic techniques. If the secretary agrees to admit `x`, it would feed the controller assigned to `x` with $\mathcal{L}$, the law of the policy in question, and the control-state assigned to `x`.

Once this connection is made, `x` can exchange $\mathcal{P}$-messages with another agent `y` similarly operating under $\mathcal{P}$; such interaction is mediated by the controllers, and does not directly involve the secretary $\mathcal{S}_\mathcal{P}$. *This makes regulated interaction quite scalable.*

Two clarifying points are in order here. First, nothing prevents a member of one policy-group, such as $\mathcal{T}$, to join another one, or to use unregulated messages for part of its communication. Therefore, a given agent may participate in any number of such policy-groups.

Second, a policy may, in principle, be served by a distributed set of secretaries each serving a subgroup of the policy group $\mathcal{G}$, perhaps in a different geographic region of the Internet.

3 Security and Trust

The security of the proposed mechanism is based on the following assumptions: (1) messages are securely transmitted over the network; (2) $\mathcal{P}$-messages are sent and received only via correctly implemented controllers, interpreting law $\mathcal{L}_\mathcal{P}$; and (3) the secretary $\mathcal{S}_\mathcal{P}$ can be trusted by everybody who depends on it. To ensure the first of these conditions, each controller, and potentially each agent in $\mathcal{G}_\mathcal{P}$, has a pair of (RSA) keys: a public key known to a trusted authority and a secret key known only by itself. If the messages sent across the network are digitally signed, their authenticity is guaranteed as long as the private key is not disclosed.

For the sake of the second condition above we postulate the existence of a "public utility" of *trusted controllers* which we call a *controller-service*. It is an organization that maintains a set of trusted hosts that function as controllers. These trusted controllers are generic, and can interpret any law provided to them by a secretary $\mathcal{S}_\mathcal{P}$ of a given policy $\mathcal{P}$. Besides interpreting laws, the controllers are built to communicate with secretaries, with members of a given policy-group, and with each other. Each such communication requires its own specific protocol. The controller-controller authentication protocol, presented in Appendix A, is particularly important. It ensures, in particular, that controllers communicate only if they operate under the same policy.

To be effective, such controllers need to be well dispersed geographically, so that it would be possible to find a controller reasonably close to prospective users. And there needs to be a trusted server that will identify a suitable controller when requested to do so by a secretary $\mathcal{S}_\mathcal{P}$. We call it a *C-server* (C for "controller"). We believe that a controller-service can be established by large, trustworthy financial institutions—like Visa, MasterCard or major banks—or by large Internet providers.

The trustworthiness of controllers is due to several factors, besides the degree to which one is willing to trust the controller service. First, it is difficult to commit fraud by subverting controllers, because they are not designed for any specific application. Second, controllers can be tested anytime by loading them with a policy designed specifically for testing.

Finally, regarding condition 3 above, we point out that a secretary $\mathcal{S}_\mathcal{P}$ may have to be trusted only by some members of its policy-group, because it cannot do much to defraud the others. First, because $\mathcal{S}_\mathcal{P}$ cannot do any harm by supplying altered rules to a controller, because such a controller would be unable to communicate with other controllers that do operate under the correct law—this is due to our controller-controller protocol. Besides, a suspicious client, who should be familiar with the law of the policy under which it operates, can always ask the controller assigned to it to show him the law.

Thus, the only way for a secretary to cause serious harm is by associating incorrect control state to an agent, thus providing him with undeserved powers under the law of the policy in question. So, in the case of policies where members are "equal" under the law, in the sense, that all members start with the same control state, the secretary need not to be trusted by any member. A large collection of policies falls into this category, including the policies presented by Ketchpel and Garcia-Molina [5], and our money-back policy presented in 4. There are cases, however, where some members have special rights under a policy because of their control state. If this is the case, the secretary has to be trusted, at least by agents that might be affected by such special rights. One way to deal with this issue is to offer liability insurance to agents operating under the policy in question, in a manner described in [7].

4 Establishing the Money Back Policy

We introduce here law $\mathcal{L}_{\mathcal{MB}}$, displayed in Figure 4, which implements the money back policy $\mathcal{MB}$ described in Section 2. We describe this law by explaining how various purchase-related operations are regulated by it.

First, consider a client `c` who wants to purchase from vendor `v`, a merchandise defined by `specs`, for which he is willing to pay 9 dollars, denoted by a certificate `ec`, and requests a grace period of 60 minutes. This is accomplished by `c` sending a `order(specs,9,ec,60)` message to `v`. Due to Rule $\mathcal{R}1$ of $\mathcal{L}_{\mathcal{MB}}$, this message would be forwarded to `v` *only* if `c` sends a valid electronic certificate for $9. Note that the electronic certificate is not sent to the vendor, but kept instead in the client's control state by the term `escrow(v,specs,9,ec,60)`, thus effectively placing the certificate in escrow. By Rule $\mathcal{R}2$, when this message arrives at `v`, it will be delivered.

The vendor `v` may respond to the order he received from `c`, with a `supply(specs,m)` message, where `m` is a certificate of some sort for the ordered merchandise (Rule $\mathcal{R}3$). By Rule $\mathcal{R}4$, the arrival of such a message at `c` causes the following operations to be carried out: (1) the merchandise is delivered to the client; and (2) the time `t` when the merchandise was supplied is recorded, so that the grace period can be observed.

The client may ask for a refund, by sending a `refund(specs)` to the vendor `v`. If the vendor has

Initially: A member may have in its control state an attribute `vendor` which characterizes the role it plays.

$\mathcal{R}1$.
```
sent(C,order(Specs,Fee,EC, GP),V) :-
      do(+escrow(V,Specs,Fee,EC,GP)),
      checkMoney(Fee,EC),
      do(forward(C,order(Specs,Fee,GP),V).
```
A message `order(Specs,Fee,GP)` *is forwarded to the destination only if the sender has a valid electronic cash certificate* `EC` *for sum* `Fee`.

$\mathcal{R}2$.
```
arrived(C,M,V) :- vendor@CS,
      (M=order(_,_,_) | M= refund(_) |
      M=payment(_)), do(deliver).
```

$\mathcal{R}3$.
```
sent(V,M,C) :- vendor@CS,
      (M=supply(_,_) | M= paymentRequest(_))
      do(forward).
```
Messages arriving at a vendor, or sent by a vendor are delivered, respectively forwarded without further ado.

$\mathcal{R}4$.
```
arrived(V,supply(Specs,M),C) :-
      clock(T),do(+supplied(V,Specs,T)),
      do(deliver).
```
If the vendor supplied the order, the merchandise is delivered to the client.

$\mathcal{R}5$.
```
sent(C,refund(Specs),V) :-
      escrow(V,Specs,Fee,EC,GP)@CS,clock(T),
      supplied(V,Specs,T')@CS →
             T <= T' + GP | true,
      do(deliver(V,refunded(Fee,EC),C)),
      do(-escrow(V,Specs,Fee,EC,GP)),
      do(forward).
```
If a client asks for a refund either before the merchandise was delivered, or while the grace period is in effect then the electronic certificate with which the client paid is delivered to him and the correspondent `escrow` *term is removed from the control state.*

$\mathcal{R}6$.
```
arrived(V,paymentRequest(Specs),C) :-
      escrow(V,Specs,Fee,EC,GP),clock(T),
      supplied(V,Specs,T'), T > T' + GP,
      do(forward(C,payment(Specs,EC),V)).
```
If the grace period ended a payment request is honored by forwarding the electronic certificate to the vendor.

$\mathcal{R}7$.
```
checkMoney(Fee,EC) :- ...
```
The electronic certificate is checked according to some rules specified by the vendor.

Figure 4: Law $\mathcal{L}_{\mathcal{MB}}$—of Money Back Policy

not yet supplied the merchandise or the grace period has not ended, the certificate `ec` with which the client paid for the merchandise is removed from escrow, by removing the corresponding `escrow` term, and it is delivered to the client. The certificate can now be used in a new transaction.

Finally, to actually get its money, a vendor can ask for payment by the means of a `paymentRequest(specs)` message. When the message arrives at the client, if the grace period is over, it causes the removal from escrow of the electronic certificate that served as payment, and its sending to the vendor (Rule $\mathcal{R}6$).

Discussion In this example, the merchandise was delivered to the client as soon as it was supplied by the vendor. If the merchandise is a certificate of some sort – like a flight ticket – the vendor incurs no losses if a refund is granted because it can always void the certificate. If the merchandise is of a different nature, like for example a digital document, the vendor may require that the actual merchandise is supplied to the client, only after the grace period ended. In this case, the law can be modified so that the merchandise is kept in escrow on the vendor's side and delivered only when the payment is received.

5 Related Work

Generally speaking, this paper is based on, but largely orthogonal to, the large volume of work on cryptographic techniques and on payment mechanisms across the net. It should be enough to mention here the following three payment methods: electronic currency [3, 8, 14], debit/credit instruments [13, 15], and secure credit card. Although our examples used electronic cash, any other payment methods could have been used.

What is directly relevant to this paper is that several researchers recently addressed the need to implement higher level e-commerce policies; a careful analysis of this need has been given by Waidner [18]. All the proposed implementations we are aware of employ *trusted proxies*—like our controllers—to mediate between the principals involved in a given commercial activity, and to make sure that the intended policy is not violated. These proposals differ from each other, and from Moses, in the manner they use these proxies, and in the nature and generality of their concept of a policy.

Ketchpel and Garcia-Molina [5] studied the transactions that occur between a customer who buys items from different vendors through brokers. The integrity of such transactions is ensured by trusted agents placed between every two principals, which operate according to rules generating by a technique called *graph sequencing*. This is an effective technique, but it is limited to a certain class of activities, and is concerned with a certain kind of issues. For example, their technique cannot prevent duplication of certificates, nor can they provide for auditing.

Su and Manchala [16] also deal with a specific class of transactions, using trusted agents to mediate between a client and a vendor. An important difference between this work and ours is that their trusted agents are not generic. Each controller must be programmed to carry out a specific protocol, which may be different from agent to agent. The deployment and management of such heterogeneous agents, and the maintenance of their trustworthiness, may be difficult. Also, Su and Tygar [17] used trusted agents to guarantee the atomicity of electronic commerce transactions.

Examples of other electronic commerce applications that use non-generic trusted agents are Millicent [4] and NetBill [15]. In Millicent, clients pay vendors with vendor's scrip obtained from a broker. The brokers are entities trusted to perform all exchanges correctly. In the latter work, Sirbu and Tygar propose a business model where merchants and clients open accounts with a NetBill server, which is the entity trusted to commit financial transactions.

Finally, Blaze, Feigenbaum and Lacy [2] built a toolkit called PolicyMaker which can interpret security policies. An agent (usually a vendor) receiving a request gives it for evaluation to PolicyMaker together with its specific policy, and the requester's credentials. On this basis the request can be found to be valid, invalid or trust can be deferred to third parties. One of the main differences between this work and ours is that PolicyMaker provides no enforcement. In particular, after asking PolicyMaker for its ruling one can proceed by ignoring it.

6 Conclusion

What makes e-commerce riskier, and thus more difficult, than conventional commerce is that its participants have little reason to trust each other [5]. What we have proposed here is to engender trust between the participants in a given e-commerce activity by imposing a suitable law over the interaction between them. The difference between unregulated e-commerce and one that is regulated by an enforced law, is analogous to the difference between doing business in an anarchic, lawless country, and doing it within a lawful society.

The benefits of regulated e-commerce can be demonstrated with our $\mathcal{MB}$ policy. The confidence that a potential client has, under this policy, of getting his money back, if he asks for it in time, is likely to embolden him to make orders he might otherwise be reluctant to make. This is true, in particular, in the following two cases: First, a potential client c may be reluctant to make an order from a server whose response time is uncertain, fearing that his money will be committed for a long period of time before getting the ordered merchandise—if he will get it at all. Under $\mathcal{MB}$, on the other hand, c can be sanguine about making orders, because he can always get his money back if not serviced quickly enough.

The second benefit to be considered is a simple solution for the following well known problem of coordination in e-commerce (this problem has been discussed, in particular, in [1, 5]). Suppose that c wants to purchase a set of items from different vendors, but he wants these items *only* if he can get them all. (These items might be tickets for a series of flights, planned for a single trip, or a set of related documents). The problem here is to avoid the danger of having c stuck with some of the items he wants, for which he has already paid, and not being able to get the rest of them, for some reason. The conventional solution to this problem is to have this transaction carried out by some kind of coordinator, who must be trusted both by c and by all the vendors involved. But no such coordinator is required under $\mathcal{MB}$. If c purchases all these items with a sufficient period of grace, then he can get refunds for the items he purchased, if he decides that he cannot get all the items he needs.*A little bit of trust can sometimes go a long way.*

It should be pointed out that there is a price to be paid for this gain of trust, which consists of: (a) the run-time cost of having each regulated message go through a pair of controllers; and, (b) the effort of maintaining a collection of identical trusted controllers throughout the Internet. We believe that the first of these costs is quite acceptable, but it is still an open question whether it is practical to deploy such controllers on a truly large scale.

Appendix

A Controller–Controller Authentication Protocol

The controller–controller authentication protocol, displayed in Figure 4, describes the necessary steps that have to be taken when a controller $\mathbf{C_A}$ sends a message **M**, on behalf of a member **A**, to another controller $\mathbf{C_B}$ assigned to a member **B**, the intended destination of the message. The purpose of the protocol is twofold. First, it has to ensure that the message **M** is securely transmitted over the network, which is a problem traditionally solved by authentication protocols. The second, more challenging goal is to authenticate controllers $\mathbf{C_A}$ and $\mathbf{C_B}$ as genuine controllers operating under the same policy.

The protocol we designed assumes that any controller **C** has an associated pair of (RSA) keys $(\mathbf{K_C}, \mathbf{K_C^{-1}})$, where $\mathbf{K_C}$ is the public key and is assumed to be known by the certifying authority, denoted hereby by **T**, and $\mathbf{K_C^{-1}}$ is the private key, and therefore known only by itself. In the description of the protocol we will use the following standard format: a communication step whereby a sender **S** transmit a message **M** to destination **D**, is represented by **S** $\rightarrow$**D**:**M**, and a message **M** encrypted with key **K** is written as $\{\mathrm{M}\}_{\mathbf{K}}$.

$$\begin{array}{ll} (1)\ \mathbf{C_A} \rightarrow \mathbf{C_B} : & \{\{\mathbf{I}, \mathbf{K}\}_{\mathbf{K}^{-1}_{\mathbf{C_A}}}\}_{\mathbf{K_{C_B}}}, \\ & \{\mathbf{B}, \mathbf{M}, \mathbf{md5}(\mathcal{L})\}_{\mathbf{K}}, \\ & \mathbf{C_A}, \{\mathbf{C_A}, \mathbf{K_{C_A}}\}_{\mathbf{K}^{-1}_{\mathbf{T}}} \\ & \\ (2)\ \mathbf{C_B} \rightarrow \mathbf{C_A} : & \mathbf{C_B}, \{\{\mathbf{I}, \mathbf{md5}(\mathcal{L})\}_{\mathbf{K}^{-1}_{\mathbf{C_B}}}\}_{\mathbf{K}} \end{array}$$

Figure 5: Controller–controller authentication protocol

In the first step, $\mathbf{C_A}$ generates a message consisting of an index number **I**, and a random key **K**. To guarantee the authenticity, the message is signed with $\mathbf{K^{-1}_{C_A}}$ its private key, and also, to achieve privacy it is encrypted with $\mathbf{K_{C_B}}$ the public key of $\mathbf{C_B}$. The index belongs to a monotonically increasing sequence and it is kept by both $\mathbf{C_A}$ and $\mathbf{C_B}$. The binding between **K** and **I** assures $\mathbf{C_B}$ that **K** is fresh[4]. Controller $\mathbf{C_A}$ generates another message consisting of **B**, the name of the intended destination, the message **M** and a hash of the law $\mathcal{L}$ of the policy under which it is operating. This message is encrypted with the session key **K**. Finally, $\mathbf{C_A}$ sends $\mathbf{C_B}$ the two messages, its name, and its public certificate signed by the trusted authority.

Controller $\mathbf{C_B}$ recovers $\mathbf{K_{C_A}}$, the public key of $\mathbf{C_A}$, from the certificate. Then, it recovers **K** and index **I** by decrypting with $\mathbf{K^{-1}_{C_B}}$, its private key, and with the public key of $\mathbf{C_A}$. After this step is performed, $\mathbf{C_B}$ is convinced that it is communicating with a genuine

[4]We borrowed the use of index number as proof of freshness from the Needham & Schroder protocol [12]. We note, that their protocol used symmetric cryptography.

controller, because $\mathbf{C_A}$ proved it knows $\mathbf{K_{C_A}^{-1}}$ which is authenticated by certifying authority $\mathbf{T}$. Finally, $\mathbf{C_B}$ decrypts the second part of the message with key $\mathbf{K}$, recovering the message $\mathbf{M}$ and the hash of the law $\mathcal{L}$. Now, if and only if, the two controllers operates under the same law the message $\mathbf{M}$ will be evaluated with respect with set of law $\mathcal{L}$ and the control state of $\mathbf{B}$.

In the second step of the protocol, $\mathbf{C_B}$ acknowledges receiving the message by sending to $\mathbf{C_A}$ the index number, and its own hash of the law encrypted with key $\mathbf{K_{C_B}^{-1}}$ for authenticity and key $\mathbf{K}$ for privacy. After $\mathbf{C_A}$ decrypts the message, it is assured that message $\mathbf{M}$ arrived correctly. Moreover, it trusts that it is talking with a genuine controller because $\mathbf{C_B}$ proved to know key $\mathbf{K_{C_B}^{-1}}$. By comparing the hash of the law received with its own he can decide whether $\mathbf{C_B}$ operates under the same policy.

After the protocol is completed, both controllers are convinced not only that they are communicating with a genuine controller, but also that they are operating under the same policy.

References

[1] J.-M. Andreoli, F. Pacull, and R. Pareschi. XPECT: A framework for electronic commerce. *IEEE Internet Computing*, pages 40–48, July-August 1997.

[2] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust managemnt. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1996.

[3] D. Chaum. Transaction systems to make big brother obsolete. In *Communication of the ACM*, October 1985.

[4] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The Millicent protocol for inexpensive electronic commerce. In *Fourth International World Wide Web Conference Proceedings*, pages 603–618, December 1995.

[5] S. Ketchpel and H. Garcia-Molina. Making trust explicit in distributed commerce transactions. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 270–281, 1996.

[6] S. Ketchpel, H. Garcia-Molina, and A. Paepcke. Shopping models: A flexible architecture for information commerce. In *Digital Libraries*, 1997.

[7] C. Lai, G. Medvinsky, and C. Neuman B. Endoresements, licensing, and insurance for distributed system services. In *Proceedings of the Second ACM Conference on Computer and Communication Security*, November 1994.

[8] G. Medvinsky and C. Neuman. Netcash: A design for practical electronic currency on the internet. In *Proceedings of the 1st ACM Conference on Computer and Communication Security*, 1993.

[9] N.H. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, February 1991.

[10] N.H. Minsky and V. Ungureanu. Regulated coordination in open distributed systems. In David Garlan and Daniel Le Metayer, editors, *Proc. of Coordination'97: Second International Conference on Coordination Models and Languages; LNCS 1282*, pages 81–98, September 1997.

[11] N.H. Minsky and V. Ungureanu. Unified support for heterogeneous security policies in distributed systems. In *7th USENIX Security Symposium*, January 1998.

[12] R. Needham and M. Schroeder. Authentication revisited. *Operating Systems Review*, page 7, January 1987.

[13] C. Neuman. Proxy-based authorization and accounting for distributed systems. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, 1993.

[14] P. Panurach. Money in electronic commerce: Digital cash, electronic fund transfer and ecash. *Communications of the ACM*, 39(6), June 1996.

[15] M. Sirbu and J.D. Tygar. Netbill: An Internet commerce system. In *IEEE COMPCON*, March 1995.

[16] J. Su and D. Manchala. Building trust for distributed commerce transactions. In *17th IEEE International Conference on Distributed Computing Systems(ICDCS)*, May 1997.

[17] J. Su and J.D. Tygar. Building blocks for atomicity in electronic commerce. In *Proceedings of USENIX Security Symposium*, 1996.

[18] M. Waidner. Development of a secure electronic marketplace for Europe. In *Proceedings of Esorics*, September 1996.