Establishing Business Rules for Inter-Enterprise Electronic Commerce*

Victoria Ungureanu** and Naftaly H. Minsky

Rutgers University, New Brunswick, NJ 08903, USA,
{ungurean,minsky}@cs.rutgers.edu

Abstract. Conventional mechanisms for electronic commerce provide strong means for securing transfer of funds, and for ensuring such things as authenticity and non-repudiation. But they generally do not attempt to regulate the activities of the participants in an e-commerce transaction, treating them, implicitly, as *autonomous agents*. This is adequate for most cases of client-to-vendor commerce, but is quite unsatisfactory for *inter-enterprise* electronic commerce. The participants in this kind of e-commerce are not autonomous agents, since their commercial activities are subject to the business rules of their respective enterprises, and to the preexisting agreements and contracts between the enterprises involved. These policies are likely to be independently developed, and may be quite heterogeneous. Yet, they have to *interoperate*, and be brought to bear in regulating each e-commerce transaction. This paper presents a mechanism that allows such interoperation between policies, and thus provides for *inter-enterprise* electronic commerce.

1 Introduction

Commercial activities need to be regulated in order to enhance the confidence of people that partake in them, and in order to ensure compliance with the various rules and regulations that govern these activities. Conventional mechanisms for electronic commerce provide strong means for securing transfer of funds, and for ensuring such things as authenticity and non-repudiation. But they generally do not attempt to regulate the activities of the participants in an e-commerce transaction, treating them, implicitly, as *autonomous agents*.

This is adequate for most cases of client-to-vendor commerce, but is quite unsatisfactory for the potentially more important *inter-enterprise* (also called business-to-business or B2B) electronic commerce[1]. The participants in this kind of e-commerce are not autonomous agents, since their commercial activities are subject to the business rules of their respective enterprises, and to the preexisting

* Appeared in the Proc. of the 14th International Symposium on DIStributed Computing (DISC 2000), LNCS 1914, October 2000, Toledo, Spain

** Work supported in part by DIMACS under contract STC-91-19999

[1] At present, B2B accounts for over 80% of all e-commerce, amounting to $150 billion in 1999 (cf. [5]). It is estimated that by 2003 this figure could reach $3 trillion.

agreements and contracts between the enterprises involved. The nature of this situation can be illustrated by the following example.

Consider a purchase transaction between an agent x_1 of an enterprise E_1 (the client in this case), and an agent x_2 of an enterprise E_2 (the vendor). Such a transaction may be subject to the following set of policies:

1. A policy $\mathcal{P}_1$ that governs the ability of agents of enterprise E_1 to engage in electronic commerce. For example, $\mathcal{P}_1$ may provide some of its agents with budgets, allowing each of them to issue purchase orders only within the budget assigned to it[2].
2. A policy $\mathcal{P}_2$ that governs the response of agents of enterprise E_2 to purchase orders received from outside. For example, $\mathcal{P}_2$ may require that all responses to purchase orders should be monitored—for the sake of internal control, say.
3. A policy $\mathcal{P}_{1,2}$ that governs the interaction between these two enterprise, reflecting some prior contract between them—we will call this an "interaction policy". For example, $\mathcal{P}_{1,2}$ may reflect a *blanket agreement* between these two enterprise, that calls for agents in E_2 to honor purchase orders from agents in E_1, for up to a certain cumulative value—to be called the "blanket" for this pair of enterprises.

Note that policies $\mathcal{P}_1$ and $\mathcal{P}_2$ are formulated separately, without any knowledge of each other, and they are likely to evolve independently. Furthermore, E_1 may have business relations with other enterprises $E_3, ..., E_k$ under a set of different interaction policies $\mathcal{P}_{1,3}, ..., \mathcal{P}_{1,k}$.

The implementation of such policies is problematic due to their *diversity*, their *interconnectivity* and the *large number of participants* involved. We will elaborate now on these factors, and draw conclusions—used as principles on which this work is based.

First, e-commerce participants have little reason to trust each other to observe any given policy—unless there is some enforcement mechanism that compels them all to do so. 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 a "manual" implementation 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. These observations suggest the following principle:

Principle 1 *E-commerce policies should be made* ***explicit****, and be* ***enforced*** *by means of a generic mechanism that can implement a wide range of policies in a uniform manner.*

[2] An agent of an enterprise may be a person or a program.

Second, e-commerce policies are usually enforced by a *central authority* (see for example, NetBill [4], SET [9], EDI [13]) which mediates between interlocutors. For example, in the case of the $\mathcal{P}_{1,2}$ policy above, one can have the blankets maintained by an authority trusted by both enterprises which will ensure that neither party violates this policy.

However such centralized enforcement mechanism is not scalable. When the number of participants grows, the centralized authority becomes a bottleneck, and a dangerous *single point of failure*. A centralized enforcement mechanism is thus unsuitable for B2B e-commerce because of the huge number of of participants involved—large companies may have as many as *tens of thousand of supplier-enterprises* (cf. [6]). The need for scalability leads to the following principle:

Principle 2 *The enforcement mechanism of e-commerce policies needs to be decentralized.*

Finally, a single B2B transaction is subject to a conjunction of several distinct and heterogeneous policies. The current method for establishing a set of policies is to to *combine* them into a single, global *super-policy*. While an attractive approach for other domains[3], combination of policies is not well suited for inter-enterprise commerce because it does not provide for the *privacy* of the interacting enterprises, nor for the *evolution* of their policies. We will now briefly elaborate on these issues.

The creation of a super-policy requires knowledge of the text of sub-policies. But divulging to a third party the internal business rules of an enterprise is not common practice in todays commerce. Even if companies would agree to expose their policies, it would still be very problematic to construct and maintain the super-policy. This is because, it is reasonable to assume that business rules of a particular enterprise or its contracts with its suppliers—the sub-policies in a B2B scenario—change in time. Each modification at the sub-policy level triggers, in turn, the modification of all super-policies it is part of, thus leading to a maintenance nightmare. We believe, therefore, that it is important for the following principle to be satisfied:

Principle 3 *Inter-operation between e-commerce policies should maintain their* `privacy`, `autonomy` *and* `mutual transparency`.

We have shown in [10] how fairly sophisticated contracts between autonomous clients and vendors can be formulated using what we call Law-Governed Interaction (LGI). The model was limited however in the sense that policies were viewed as isolated entities. In this paper we will describe how LGI has been extended, in accordance with the principles mentioned above, to support policy inter-operation.

The rest of the paper is organized as follows. We start, in Section 2, with a brief description of the concept of LGI, on which this work is based; in Section 3

[3] like for example federation of databases, for which it was originally devised

we present our concept of policy-interoperability. Details of a secure implementation of interoperability under LGI are provided in Section 4. Section 5 discusses some related work, and we conclude in Section 6.

2 Law-Governed Interaction (LGI)—an Overview

Broadly speaking, LGI is a mode of interaction that allows an heterogeneous group of distributed agents to interact with each other, with confidence that an explicitly specified set of rules of engagement–called the *law* of the group—is strictly observed by each of its member. Here we provide a brief overview of LGI, for more detailed discussion see [10, 11].

The central concept of LGI is that of a policy $\mathcal{P}$, defined as a four-tuple:

$$\langle \mathcal{M}, \mathcal{G}, \mathcal{CS}, \mathcal{L} \rangle$$

where $\mathcal{M}$ is the set of messages regulated by this policy, $\mathcal{G}$ is an open and heterogeneous group of *agents* that exchange messages belonging to $\mathcal{M}$; $\mathcal{CS}$ is a mutable set $\{\mathcal{CS}_x \mid x \in \mathcal{G}\}$ of what we call *control states*, one per member of group $\mathcal{G}$; and $\mathcal{L}$ is an enforced set of "rules of engagement" that regulates the exchange of messages between members of $\mathcal{G}$. We will now give a brief description of the basic components of a policy.

The Law: The law is defined over certain types of events occuring at members of $\mathcal{G}$, mandating the effect that any such event should have—this mandate is called the *ruling* of the law for a given event. The events thus subject to the law of a group under LGI are called *regulated events*—they include (but are not limited to) the sending and arrival of $\mathcal{P}$-messages.

The Group: We refer to members of $\mathcal{G}$ as *agents*, by which we mean autonomous actors that can interact with each other, and with their environment. Such an agent might be an encapsulated software entity, with its own state and thread of control, or it might be a human that interacts with the system via some interface. (Given popular usage of the term "agent", it is important to point out that this term does not imply here either "intelligence" nor mobility, although neither of these is ruled out.) Nothing is assumed here about the structure and behavior of the members of a given $\mathcal{L}$-group, which are viewed simply as sources of messages, and targets for them.

The Control State: The *control-state* $\mathcal{CS}_x$ of a given agent x is the bag of attributes associated with this agent (represented here as Prolog-like terms). These attributes are used to structure the group $\mathcal{G}$, and provide state information about individual agents, allowing the law $\mathcal{L}$ to make distinctions between different members of the group. The control-state $\mathcal{CS}_x$ can be acted on by the primitive operations, which are described below, subject to law $\mathcal{L}$.

Regulated Events: The events that are subject to the law of a policy are called *regulated events*. Each of these events occurs at a certain agent, called the *home* of the event[4] The following are two of these event-types:

1. `sent(x,m,y)`—occurs when agent `x` sends an $\mathcal{L}$-message `m` addressed to `y`. The sender `x` is considered the *home* of this event.
2. `arrived(x,m,y)`—occurs when an $\mathcal{L}$-message `m` sent by `x` arrives at `y`. The receiver `y` is considered the *home* of this event.

Primitive Operations: The operations that can be included in the ruling of the law for a given regulated event `e`, to be carried out at the home of this event, are called *primitive operations*. Primitive operations currently supported by LGI include operations for testing the control-state of an agent and for its update, operations on messages, and some others. A sample of primitive operations is presented in Figure 1.

Operations on the control-state	
`t@CS`	returns true if term `t` is present in the control state, and fails otherwise
`+t`	adds term `t` to the control state;
`-t`	removes term `t` from the control state;
`t1`←`t2`	replaces term `t1` with term `t2`;
`incr(t(v),d)`	increments the value of the parameter `v` of term `t` with quantity `d`
`dcr(t(v),d)`	decrements the value of the parameter `v` of term `t` with quantity `d`
Operations on messages	
forward(x,m,y)	sends message `m` from `x` to `y`; triggers at `y` an `arrived(x,m,y)` event
deliver(x,m,y)	delivers to agent `y` message `m` (sent by `x`)

Fig. 1. Some primitive operations

The Law-Enforcement Mechanism: Law $\mathcal{L}_\mathcal{P}$ is enforced by a set of trusted entities called *controllers* that mediate the exchange of $\mathcal{P}$-messages between members of group $\mathcal{G}$. For every active member `x` in $\mathcal{G}$, there is a controller $\mathcal{C}_x$ logically placed between `x` and the communications medium. And all these controller carry the *same law* $\mathcal{L}$. This allows the controller $\mathcal{C}_x$ assigned to `x` to compute the ruling of $\mathcal{L}$ for every event at `x`, and to carry out this ruling locally.

Controllers are *generic*, and can interpret and enforce any well formed law. A controller operates as an independent process, and it may be placed on the same machine as its client, or on some other machine, anywhere in the network. Under Moses (our current implementation of LGI) each controller can serve several agents, operating under possibly different laws.

[4] strictly speaking, events occur at the controller assigned to the home-agent.

3 Interoperability Between Policies

In this section we introduce an extension to LGI framework that provides for the interoperability of different and otherwise unrelated policies. This section is organized as follows: in Section 3.1 we present our concept of interoperability. In Section 3.2 we describe an extension of LGI that supports this concept; the extended LGI is used in Section 3.3 to implement a slightly refined version of the motivating example presented in Section 1. We conclude this Section by showing how privacy, autonomy and transparency of interoperating policies are achieved in this framework.

3.1 A Concept of Interoperability

By "interoperability" we mean here, the ability of an agent $x/\mathcal{P}$ (short for "an agent x operating under policy $\mathcal{P}$") to exchange messages with $y/\mathcal{Q}$, were $\mathcal{P}$ and $\mathcal{Q}$ are different policies[5], such that the following properties are satisfied:

consensus: An exchange between a pair of policies is possible only if it is authorized by both.

autonomy: The effect that an exchange initiated by $x/\mathcal{P}$ may have on the structure and behavior of $y/\mathcal{Q}$, is subject to policy $\mathcal{Q}$ alone.

transparency: Interoperating parties need not to be aware of the details of each other policy.

To provide for such an inter-policy exchange we introduce into LGI a new primitive operation—`export`—and a new event—`imported`, as follows:

- Operation `export(x/`$\mathcal{P}$`,m,y/`$\mathcal{Q}$`)`, invoked by agent x under policy $\mathcal{P}$, initiates an exchange between x and agent y operating under policy $\mathcal{Q}$. When the message carrying this exchange arrive at y it would invoke at it an `imported` event under $\mathcal{Q}$.
- Event `imported(x/`$\mathcal{P}$`,m,y/`$\mathcal{Q}$`)` occurs when a message `m` exported by $x/\mathcal{P}$ arrives at $y/\mathcal{Q}$.

We will return to the above properties in Section 3.4 and show how they are brought to bear under LGI.

3.2 Support for Interoperability under LGI

A policy $\mathcal{P}$ is maintained by 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 $\mathcal{S}_{\mathcal{P}}$. In the basic LGI mechanism, the secretary serves as a name server for policy members. In the extended model it acts also as a name server for the policies which inter-operate with $\mathcal{P}$. In order to do so $\mathcal{S}_{\mathcal{P}}$ maintains a list of policies to which members of $\mathcal{P}$ are allowed to export to,

[5] It is interesting to note that x and y may actually be the same agent.

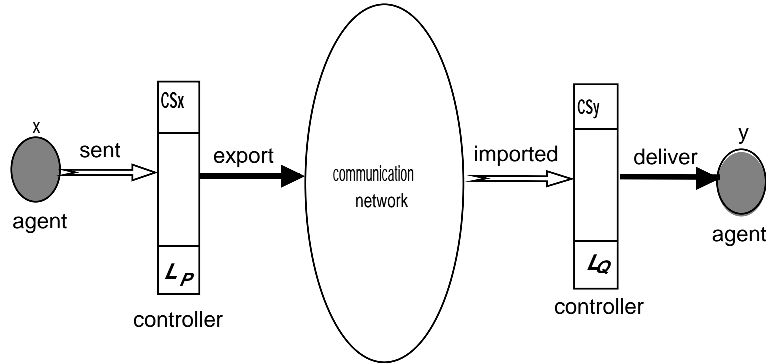


Fig. 2. Policy interoperation

and respectively import from (subject of course to $\mathcal{L}_{\mathcal{P}}$). For each such policy $\mathcal{P}'$, $\mathcal{S}_{\mathcal{P}}$ records among other information the address of $\mathcal{S}_{\mathcal{P}'}$.

For an agent `x` to be able to exchange $\mathcal{P}$-messages under a policy $\mathcal{P}$, it needs to engage in a connection protocol with the secretary. The purpose of the protocol is to assign `x` to a controller $\mathcal{C}_x$ which is fed the law of $\mathcal{P}$ and the control state of `x` (for a detailed presentation of this protocol the reader is referred to [11]).

To see how an `export` operation is carried out, consider an agent `x` operating under policy $\mathcal{P}$, which sends a message `m` to agent `y` operating under policy $\mathcal{Q}$ assuming that `x` and `y` have joined the policy $\mathcal{P}$, respectively $\mathcal{Q}$ (Figure 2). Message `m` is sent by means of a routine provided by the Moses toolkit, which forwards it to $\mathcal{C}_x$—the controller assigned to `x`. When this message arrives at $\mathcal{C}_x$, it generates a `sent(x,m,y)` event at it. $\mathcal{C}_x$ then evaluates the ruling of law $\mathcal{L}_{\mathcal{P}}$ for this event, taking into account the control-state $\mathcal{CS}_x$ that it maintains, and carries out this ruling.

If this ruling calls the control-state $\mathcal{CS}_x$ to be updated, such update is carried out directly by $\mathcal{C}_x$. And if the ruling for the `sent` event calls for the `export` of `m` to `y`, this is carried out as follows. If C_x does not have the address of C_y, the controller assigned to `y`, it will ask $\mathcal{S}_{\mathcal{P}}$ for it. When the secretary responds, $\mathcal{C}_x$ will finalize the `export` and will cache the address. As such, forthcoming communication between `x` and `y` will not require the extra step of contacting $\mathcal{S}_{\mathcal{P}}$.

When the message `m` sent by $\mathcal{C}_x$ arrives at $\mathcal{C}_y$ it generates an `imported(x,m,y)` event. Controller $\mathcal{C}_y$ computes and carries out the ruling of law $\mathcal{L}_{\mathcal{Q}}$ for this event. This ruling might, for example, call for the control-state $\mathcal{CS}_y$ maintained by $\mathcal{C}_y$ to be modified. The ruling might also call for `m` to be delivered to `y`, thus completing the passage of message `m`.

In general, all regulated events that occur nominally at an agent `x` actually occur at its controller $\mathcal{C}_x$. The events pertaining to `x` are handled *sequentially* in chronological order of their occurrence. The controller evaluates the ruling of the law for each event, and carries out this ruling *atomically*, so that the sequence of

operations that constitute the ruling for one event do not interleave with those of any other event occuring at `x`. Note that a controller might be associated with several agents, in which case events pertaining to different agents are evaluated concurrently.

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 messages are mediated by correctly implemented controllers. The way to gain such an assurance is a security issue we will address in Section 4.

3.3 A Case Study

We now show how a slightly refined version of the three policies $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_{1,2}$, introduced in Section 1, can be formalized, and thus enforced, under LGI. We note that $\mathcal{P}_1$ and $\mathcal{P}_2$ do not depend on each other in any way. Each of these policies provides for export to, and import from, the interaction policy $\mathcal{P}_{1,2}$, but they have no dependency on the internal structure of $\mathcal{P}_{1,2}$.

After the presentation of these three policies we will we illustrate the manner in which they interoperate by describing the progression of a single purchase transaction. We conclude this section with a brief discussion.

Policy $\mathcal{P}_1$ Informally, this policy, which governs the ability of agents of enterprise E_1 to issue purchase orders, can be stated as follows:

> For an agent in E_1 to issue a purchase order (PO) it must have a budget assigned to it, with a balance exceeding the price in the PO. Once a PO is issued, the agent's budget is reduced accordingly. If the PO is not honored, for whatever reason, then the client's budget is restored.

Formally, under LGI, the components of $\mathcal{P}_1$ are as follows: the group $\mathcal{G}$ consists of the employees allowed to make purchases. The set $\mathcal{M}$ consists of the following set of messages:

- `purchaseOrder(specs,price,c)`, which denotes a purchase order for a merchandise described by `specs` and for which the client `c` is willing to pay amount `price`.
- `supplyOrder(specs,ticket)`, which represents a positive response to the PO, where `ticket` represents the requested merchandise. (We assume here that the merchandise is in digital form, e.g. an airplane ticket, or some kind of certificate. If this is not the case, then the merchandise delivery cannot be formalized under LGI.)
- `declineOrder(specs,price,reason)` denoting that the order is declined and containing a `reason` for the decline.

The control-state of each member in this policy contains a term `budget(val)`, where `val` is the value of the budget. Finally, the law of this policy is presented in Figure 3. This law consists of three rules. Each rule is followed by an explanatory comment (in italics). Note that under this law, members of $\mathcal{P}_1$ are allowed to interoperate only with members of policy $\mathcal{P}_{1,2}$.

Initially: A member has in his control state an attribute budget(val), where `val` represents the total dollar amount it can spend for purchases.

$\mathcal{R}1$.
```
sent(X1, purchaseOrder(Specs,Price,X1),X2) :-
        budget(Val)@CS, Val>Price,
        do(dcr(budget(Val),Price)),
        do(export(X1/p1,purchaseOrder(Specs,Price,X1),X2/p12)).
```
A `purchaseOrder` *message is exported to the vendor* `X2` *that operates under the inter-enterprise policy* `p12`*—but only if* `Price`*, the amount* `X1` *is willing to pay for the merchandise, is less than* `Val`*, the value of the sender's budget.*

$\mathcal{R}2$.
```
imported(X2/p12,supplyOrder(Specs,Ticket),X1/p1) :-
        do(deliver(X2,supplyOrder(Specs,Ticket),X1)).
```
A `supplyOrder` *message, imported from* `p12`*, is delivered without further ado.*

$\mathcal{R}3$.
```
imported(X2/p12,declineOrder(Specs,Price,Reason),X1/p1) :-
        do(incr(budget(Val),Price)),
        do(deliver(X2,declineOrder(Specs,Price,Reason),X1)).
```
A `declineOrder` *message, imported from* `p12`*, is delivered after the budget is restored by incrementing it with the price of the failed PO.*

Fig. 3. The law of policy $\mathcal{P}_1$

Policy $\mathcal{P}_2$ Informally, this policy which governs the response of agents of E_2 to purchase orders, can be stated simply as follows:

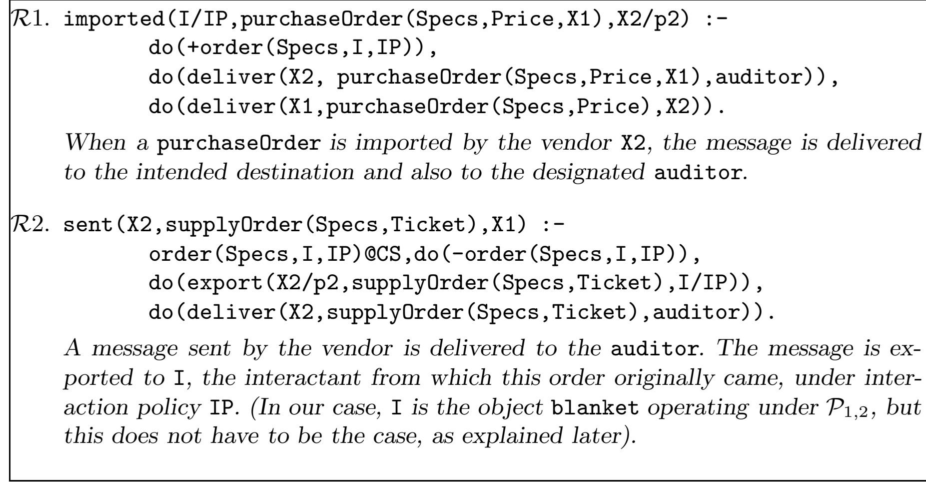
> Each phase of a purchase transaction is to be monitored by a designated agent called `auditor`.

The components of $\mathcal{P}_2$ are as follows: the group $\mathcal{G}$ of this policy consists of the set of employees of E_2 allowed to serve purchase orders, and of a designated agent `auditor` that maintains the audit trail of their activities. For simplicity, we assume here that the set of messages recognized by this policy is the same as for policy $\mathcal{P}_1$—this is not necessary, as will be explained later. The law $\mathcal{L}_2$ of this policy is displayed in Figure 4. Note that unlike $\mathcal{L}_1$, which allows for interoperability only with policy $\mathcal{P}_{1,2}$, this law allows for interoperability with arbitrary policies. (The significance of this will be discussed later.)

Policy $\mathcal{P}_{1,2}$ We assume that there is a *blanket agreement* $\mathcal{P}_{1,2}$ between enterprises E_1 and E_2, stated, informally, as follows:

> A purchase order is processed by the vendor only if the amount offered by the client does not exceed the remaining balance in the blanket.

The components, under LGI, of this policy are as follows: the group $\mathcal{G}$ consists of the set of agents from the vendor-enterprise E_2 that may serve purchase orders, and a distinguished agent called `blanket` that maintains the balance for



```
R1. imported(I/IP,purchaseOrder(Specs,Price,X1),X2/p2) :-
        do(+order(Specs,I,IP)),
        do(deliver(X2, purchaseOrder(Specs,Price,X1),auditor)),
        do(deliver(X1,purchaseOrder(Specs,Price),X2)).
```

When a `purchaseOrder` *is imported by the vendor* `X2`*, the message is delivered to the intended destination and also to the designated* `auditor`*.*

```
R2. sent(X2,supplyOrder(Specs,Ticket),X1) :-
        order(Specs,I,IP)@CS,do(-order(Specs,I,IP)),
        do(export(X2/p2,supplyOrder(Specs,Ticket),I/IP)),
        do(deliver(X2,supplyOrder(Specs,Ticket),auditor)).
```

A message sent by the vendor is delivered to the `auditor`*. The message is exported to* `I`*, the interactant from which this order originally came, under interaction policy* `IP`*. (In our case,* `I` *is the object* `blanket` *operating under* $\mathcal{P}_{1,2}$*, but this does not have to be the case, as explained later).*

Fig. 4. The law of policy $\mathcal{P}_2$

the purchases of the client-enterprise E_1. The law $\mathcal{L}_{1,2}$ of this policy is displayed in Figure 5.

The Progression of a Purchase Transaction We explain now how these policies function together, by means of a step-by-step description of the progression of a purchase transaction initiated by a PO `purchaseOrder(specs,price,x1)` sent by agent x_1 of an enterprise E_1 (the client) to an agent x_2 of E_2 (the vendor).

1. The sending by x_1 of a PO to x_2 is handled by policy $\mathcal{P}_1$ (see Rule $\mathcal{R}1$ of $\mathcal{P}_1$) as follows. If the budget of x_1 is smaller than the specified price, then this PO is simply ignored; otherwise the following operations are carried out: (a) the budget of x_1 is decremented by the specified price; and (b) the PO is *exported* to $x_2/\mathcal{P}_{1,2}$, i.e., to agent x_2 under policy $\mathcal{P}_{1,2}$.
2. The import of a PO into $x_2/\mathcal{P}_{1,2}$ forces the PO to be immediately forwarded to an agent called *blanket*. Agent *blanket*, which operates under $\mathcal{P}_{1,2}$, has in its control-state the term $balance(val)$, where *val* represents the remaining balance under the blanket agreement between the two enterprises (Rule $\mathcal{R}1$ of $\mathcal{P}_{1,2}$).
3. The arrival of a PO at *blanket* agent causes the balance of the blanket to be compared with the price of the PO. If the balance is bigger it is decremented by the price, and the PO is exported to the vendor agent $x_2/\mathcal{P}_2$ (Rule $\mathcal{R}2$ of $\mathcal{P}_{1,2}$); otherwise, a `declineOrder` message is exported back to the client $x_1/\mathcal{P}_1$ (Rule $\mathcal{R}3$ of $\mathcal{P}_{1,2}$). We will assume for now that the former happen; we will see later what happens when a `declineOrder` message arrives at a client.
4. When a PO exported by agent *blanket* (signifying consistency with the blanket agreement) is imported into $x_2/\mathcal{P}_2$, it is immediately delivered to two agents: (a) to the vendor agent x_2 himself, for its disposition; and (b) to to

Initially: Agent blanket has in its control state a term of the form balance(val), where val denotes the remaining amount of money that the client-enterprise E_1 has available for purchases, at a given moment in time.

$\mathcal{R}1$.
```
imported(X1/p1,purchaseOrder(Specs,Price),X2/p12) :-
        do(forward(X2,purchaseOrder(Specs,Price,X1), blanket)).
```
A `purchaseOrder` *message imported by a vendor* `X2` *is forwarded to* `blanket` *for approval.*

$\mathcal{R}2$.
```
arrived(X2,purchaseOrder(Specs,Price,X1),blanket) :-
        balance(Val)@CS, Val>=Price,
        do(dcr(balance(Val),Price)),
        do(+ order(Specs,Price,X1,X2)),
        do(export(blanket/p12, purchaseOrder(Specs,Price,X1),X2/p2)).
```
If `Price`*, the sum* `X1` *is willing to pay for the merchandise, is less than* `Val` *the value of the balance, then the* `purchaseOrder` *message is exported to* `X2`*, the vendor which originally received the request under policy* `p2`*.*

$\mathcal{R}3$.
```
arrived(X2,purchaseOrder(Specs,Price,X1),blanket) :-
        balance(Val)@CS, Val<Price,
        do(export(X2/p12, declineOrder(Specs,Price,''insufficient
        funds''),X1/p1)).
```
If the balance is less than the `Price` *then a* `declineOrder` *message is exported to* `X1`*, the client which originally issued the* `purchaseOrder`*.*

$\mathcal{R}4$.
```
imported(X2/p2,supplyOrder(Specs,Ticket),blanket/p12) :-
        order(Specs,Price,X1,X2)@CS, do(-order(Specs,Price,X1,X2)),
        do(export(X2/p12, supply(Specs,Ticket ),X1/p1)).
```
A `supplyOrder` *message is exported to the client* `X1` *which issued the order.*

Fig. 5. The law of policy $\mathcal{P}_{1,2}$

the distinguished agent *auditor*, designated to maintain the audit trail of responses of vendor-agents to purchase orders (Rule $\mathcal{R}1$ of $\mathcal{P}_2$).

5. According to policy $\mathcal{P}_2$, agent x_2 that received a PO can respond by a `supplyOrder` message[6] which triggers two operations: (a) the message is exported to $blanket/\mathcal{P}_{1,2}$, and (b) a copy of this message is delivered to the *auditor* object (Rule $\mathcal{R}2$ of $\mathcal{P}_2$).
6. An import of the `supplyOrder` response of $x_2/\mathcal{P}_2$ into $blanket/\mathcal{P}_{1,2}$ is automatically exported to the client $x_1/\mathcal{P}_1$ (Rule $\mathcal{R}4$ of $\mathcal{P}_{1,2}$).
7. Finally, the import of a `supplyOrder` message into $x_1/\mathcal{P}_1$ causes this message to be delivered to x_1 (Rule $\mathcal{R}2$ of $\mathcal{P}_1$) , while the import of a `declineOrder`

[6] To keep the example simple we did not describe here the case when the vendor decline the PO.

message into $x_1/\mathcal{P}_1$ causes the budget of x_1 to be restored, before the message is delivered to it (Rule $\mathcal{R}3$ of $\mathcal{P}_1$).

Discussion This case study makes the following simplifying assumptions: (1) all three policies use the same set of messages, and (2) the client-enterprise policy $\mathcal{P}_1$ allows for interoperation only with $\mathcal{P}_{1,2}$. These assumptions are not intrinsic to the proposed model and were adopted only in order to make the example as simple as possible. We will explain now the drawbacks of these assumption and show how they can be relaxed, making this case study far more general and flexible.

First, it is unreasonable to assume that completely different enterprises will use the same *vocabulary* with the same *semantic*. While it is required by the model that interoperating policies—in our example $\mathcal{P}_1$ and $\mathcal{P}_{1,2}$ on one side, and $\mathcal{P}_{1,2}$ and $\mathcal{P}_2$ on the other—"understand" each other messages, policies $\mathcal{P}_1$ and $\mathcal{P}_2$ could have used entirely different messages. The translation from $\mathcal{M}_{\mathcal{P}_1}$ to $\mathcal{M}_{\mathcal{P}_2}$ can generally be done by the intermediate policy $\mathcal{P}_{1,2}$.

Second, it is unrealistic to assume that an enterprise will purchase merchandise only from a single vendor E_2, as is required by our current $\mathcal{P}_1$—which is coded to interoperate only with $\mathcal{P}_{1,2}$, representing the contract between E_1 and E_2. In general, one should provide for an agent in E_1 to purchase from other vendors—say from $E_{2'}$, through an inter-enterprise policy $\mathcal{P}_{1,2'}$ reflecting a pre-agreement between E_1 and $E_{2'}$. An analogous flexibility is inherent in $\mathcal{P}_2$, which does not pose any restrictions on the policy it interoperates with, and thus allows for establishing contracts with *different* clients. A similar technique can be used in $\mathcal{P}_1$ to allow purchasing from any number of vendors.

3.4 Assurances

We are in position now to explain how the three properties of our concept of interoperability, namely `consensus`, `autonomy` and `transparency` are satisfied by LGI mechanism.

The consensus condition stipulated that interoperation between a pair of policies should be agreed by both. This property is satisfied by our implementation because for an agent under $\mathcal{P}$ to send a message to an agent under a different policy $\mathcal{Q}$, $\mathcal{P}$ must have a rule that invokes an *export* operation to $\mathcal{Q}$, and $\mathcal{Q}$ must have a rule that responds to an *imported* event from $\mathcal{P}$.

The *autonomy* condition is satisfied, because the effect on $y/\mathcal{Q}$ of a message imported from elsewhere is determined *only* by the *imported*-rules in $\mathcal{Q}$. Finally, *transparency* is satisfied because, when an agent $y/\mathcal{Q}$ handles a message exported from $x/\mathcal{P}$, it has access only to the message itself and to its source, but not to the policy $\mathcal{P}$ under which it has been produced.

4 A Secure Implementation of Interoperability

To prevent malicious violations of the law, the following conditions have to be met: (1) messages are sent and received only via correctly implemented con-

trollers, and (2) messages are securely transmitted over the network. The first of these conditions can be be handled at different levels of security. First, controllers may be placed on trusted machines. Second, controllers may be trusted when built into *physically secure coprocessors* [12].

To ensure condition (2) above we devised and implemented in Moses toolkit the controller–controller authentication protocol displayed in Figure 6. The purpose of the protocol is twofold. First, it has to ensure that messages are securely transmitted over the network—which is a problem traditionally solved by authentication protocols. The second, more challenging goal is to authenticate communicating controllers as genuine controllers operating under inter operating policies.

This protocol assumes that any controller C has a pair of keys $(\mathbf{K_C}, \mathbf{K_C^{-1}})$, where $\mathbf{K_C}$ is the public key and is assumed to be known by the trusted authority, $\mathbf{T}$[7] and $\mathbf{K_C^{-1}}$ is the private key, and therefore known only by itself. Also the protocol assumes that if C is assigned a member in a policy $\mathcal{P}$, then C maintains a list of the policies which inter-operates with $\mathcal{P}$. For every such policy $\mathcal{P}'$ in this list, C records its identifier $\mathbf{id}(\mathcal{P}')$, the hash of the law $\mathbf{H}(\mathcal{L}_{\mathcal{P}'})$, and the address of the secretary of $\mathcal{P}'$. In the current implementation, this information is given to C by the secretary of $\mathcal{P}$ at the time a member in $\mathcal{P}$ is assigned to C.

$$
\begin{array}{ll}
\mathbf{(1)}\ C_x \rightarrow C_y : & \mathbf{x}, \mathbf{m}, \mathbf{y}, \mathbf{i}, \\
 & \mathbf{id}(\mathcal{P}), \mathbf{H}(\mathcal{L}_{\mathcal{P}}), \\
 & \{\mathbf{controller}, \mathbf{K_{C_x}}\}_{\mathbf{K_T^{-1}}}, \ \mathbf{S}_{\mathbf{K_{C_x}^{-1}}}(\mathbf{x}, \mathbf{m}, \mathbf{y}, \mathbf{i}, \mathbf{H}(\mathcal{L}_{\mathcal{P}}), \mathbf{H}(\mathcal{L}_{\mathcal{Q}})) \\
 & \\
\mathbf{(2)}\ C_y \rightarrow C_x : & \{\mathbf{controller}, \mathbf{K_{C_y}}\}_{\mathbf{K_T^{-1}}} \\
 & \mathbf{S}_{\mathbf{K_{C_y}^{-1}}}(\mathbf{i}, \mathbf{H}(\mathcal{L}_{\mathcal{Q}}))
\end{array}
$$

Fig. 6. Controller–controller authentication protocol

The protocol describes the necessary steps that have to be taken when a controller $\mathbf{C_x}$ sends a message $\mathbf{m}$, on behalf of a member $\mathbf{x}$ operating in policy $\mathcal{P}$, to another controller $\mathbf{C_y}$ assigned to a member $\mathbf{y}$ in policy $\mathcal{Q}$. In the first step of the protocol, C_x sends to C_y a message consisting of `x`, `m`, `y`, and an index number `i`. The index `i` is used to prevent replay attacks and it is maintained by both C_x and C_y. In order to identify to C_y the policy $\mathcal{P}$ to which $\mathbf{x}$ belongs, C_x also transmits $id(\mathcal{P})$ the (unique) identifier of $\mathcal{P}$ and the hash of $\mathcal{L}_{\mathcal{P}}$. To authenticate itself to C_y as a genuine controller, C_x sends to C_y its public certificate along with the signature of a a message consisting of `x`, `m`, `y`, `i`, and the hashes of sender and destination laws.

Now, when controller C_y receives the message it first checks whether `y` is allowed to import messages sent by members in $\mathcal{P}$ policy-group. If this is the case,

[7] For simplicity we assume here a unique certifying authority; the protocol could be easily extended to support a hierarchy of such authorities.

C_y recovers $\mathtt{K}_{C_x}$, the public key of C_x, from the certificate and verifies the signature. If the signature is correct[8] C_y is convinced that it is communicating with a genuine controller, because C_x proved it knows $\mathtt{K}^{-1}_{C_x}$ which is authenticated by the certifying authority. The signature also proves that the message was sent under $\mathcal{L}_\mathcal{P}$ and knowledge of $\mathcal{L}_\mathcal{Q}$. If all conditions are met then an `import(x/`$\mathcal{P}$`,m,y/`$\mathcal{Q}$`)` event will be triggered at $\mathcal{C}_y$.

In the second step of the protocol, C_y acknowledges receiving the message by sending to C_x the signature of the index number `i`, and the hash of the law $\mathcal{L}_\mathcal{P}$, together with its own certificate. After C_x verifies the signature, it is assured that message `m` arrived correctly. Moreover, it trusts that it is talking with a genuine controller because C_y proved to know key $\mathtt{K}^{-1}_{C_y}$. By comparing the hash of the law received with its own C_x can decide whether C_y operates under the law it is expected to.

5 Related Work

The fact that participants in an electronic transaction have different policies, and the importance of finding a common ground between them has been recognized by several researchers. Ketchpel and Garcia-Molina [8] 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. Their role is to generate a transaction protocol which satisfies the policies of the two principals. The protocol is automatically generated using a technique called *graph sequencing*. This is an effective technique, but is limited to individual client-vendor situation, in the sense that a particular client (or vendor) is not bound by an enterprise policy.

Abiteboul, Vianu, Fordham and Yesha [1] propose that the transactions between a client and a vendor be mediated by relational transducers. Generally, such a transducer implements the vendor policy, but their mechanism allows for the modification of the policy. This suggests, that in principle it should be possible that a client may add its own policy. However, such a composition of policies is computationally expensive to enforce—it is undecidable in the general case.

Composition of policies in the context of access control has been studied by several authors: Gong and Qian [7] achieve *policy interoperation* by inferring a composed policy based on (compatible) sub-policies. Another approach, which allows for inter-operation of not necessarily compatible policies is *policy combination* [3]. Finally, we are mentioning the *hierarchical composition* of policies presented in [2]. These approaches rely on the assumption that there is a higher authority which is aware of all sub-policies. Such solutions are not applicable to B2B commerce since there is currently no such authority.

[8] For simplicity, we don't discuss here the case $\mathcal{P}$ is not authorized to export messages to $\mathcal{Q}$ policy-group, or the signature is incorrect. Suffices to say that if this is the case C_y will notify C_x, which in turn will notify `x`.

6 Conclusion

This paper addressed the issue of inter-enterprise electronic commerce, which may be subject to a combination of several heterogeneous policies formulated independently by different authorities. Starting from a mechanism, such as LGI, that supports a formal and enforced concept of a policy, we have argued that such an inter-enterprise commerce requires distinct policies to be able to interoperate, while maintaining mutual transparency, and without loosing their autonomy. We have shown how such a concept of policy-interoperation is implemented in LGI, in a secure and scalable manner, and we have demonstrated the application of this facility for inter-enterprise electronic commerce.

References

1. S. Abiteboul, V. Vianu, B. Forham, and Y. Yesha. Relational transducers for electronic commerce. In *Symposium on Principles of Database Systems*, pages 179–187, June 1998.
2. E. Bertino, F. Buccafurri, E. Ferrari, and P. Rullo. An authorization model and its formal semantics. In *Proceedings of 5th European Symposium on Research in Computer Security*, pages 127–143, September 1998.
3. C. Bidan and V. Issarny. Dealing with multi-policy security in large open distributed systems. In *Proceedings of 5th European Symposium on Research in Computer Security*, pages 51–66, September 1998.
4. B. Cox, J. D. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *First USENIX Workshop on Electronic Commerce*, July 1995.
5. The Economist. E-commerce (a survey). pages 6–54. (February 26th 2000 issue).
6. The Economist. Riding the storm. pages 63–64. (November 6th 1999 issue).
7. L. Gong and X. Qian. Computational issues in secure interoperation. *IEEE Transctions on Software Engineering*, pages 43–52, January 1996.
8. 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.
9. SETCo LLC. Set Secure Electronic Transaction protocol. website: `http://www.ibm.com/security/html/prod_setp.html` .
10. N.H. Minsky and V. Ungureanu. A mechanism for establishing policies for electronic commerce. In *The 18th International Conference on Distributed Computing Systems (ICDCS)*, pages 322–331, May 1998.
11. N.H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology*, July 2000.
12. S.W. Smith and S. H. Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31:831–860, April 1999.
13. P.K. Sokol. *From EDI to Electronic Commerce—A Bussines Initiative*. Mc Graw-Hill, 1995.