

# Scalable Regulation of Inter-Enterprise Electronic Commerce\*

Naftaly H. Minsky\*\* and Victoria Ungureanu\*\*\*

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

**Abstract.** In the current electronic-commerce literature, a commercial transaction is commonly viewed as an exchange between two autonomous principals operating under some kind of contract between them—which needs to be formalized and enforced. But the situation can be considerably more complex in the case of *inter-enterprise* (also called business-to-business, or B2B) commerce. The participants in a B2B transaction are generally not autonomous agents, since their commercial activities are subject to the policies of their respective enterprises.

It is our thesis, therefore, that a B2B transaction should be viewed as being governed by three distinct policies: the two policies that regulate the activities of the two principals, while operating as representatives of their respective enterprises, and the policy that reflects the contract between the two enterprises. These policies are likely to be independently developed, and may be quite heterogeneous. Yet, they have to *interoperate*, and must all be brought to bear in regulating each B2B transaction. This paper presents a mechanism for formulating such interoperating policies, and for their scalable enforcement, thus providing for regulated *inter-enterprise* electronic commerce.

## 1 Introduction

In the current electronic-commerce literature, a commercial transaction is commonly viewed as an exchange between two autonomous principals operating under some kind of contract between them. The formulation and enforcement of such contracts are among the main problems presently facing this field. But the situation can be considerably more complex, and more challenging, in the case of *inter-enterprise* commerce (also called business-to-business, or B2B commerce). This is because the participants in a B2B transaction are generally not autonomous agents but are subject to the policies of their respective enterprises.

It is our thesis, therefore, that a B2B transaction should be viewed as being governed by three distinct policies: the two policies that regulate the activities

---

\* In Proc. of the Second International Workshop on Electronic Commerce, Heidelberg, Germany, November 2001.

\*\* Work supported in part by NSF grants No. CCR-9710575 and No. CCR-98-03698

\*\*\* Work supported in part by DIMACS under contract STC-91-19999 ITECC, and Information Technology and Electronic Commerce Clinic, Rutgers University

of the two principals, while operating as representatives of their respective enterprises, and the policy that reflects the contract between the two enterprises. This state of affairs can be illustrated by the following example: Consider a pair of enterprises  $E_1$  and  $E_2$  that trade with each other under a contract defined by policy  $P_{12}$  below:

**Policy  $P_{12}$ :** *A business transaction between  $E_1$ , the client enterprise in this case, and  $E_2$ , the vendor, is initiated by a purchase order (PO) sent by some agent  $x_1$  of  $E_1$  to an agent  $x_2$  of  $E_2$ , specifying the merchandise and the desired delivery time  $t$ . The exchange of merchandise and payment between the two agents is subject to the following provisions:*

- *payment should accompany the PO;*
- *the buyer,  $x_1$ , can cancel his order before the specified delivery time  $t$ , and will be reimbursed 90% of the payment he made, while 10% of it will go to  $x_2$ .*
- *if the merchandise has been supplied by  $x_2$  by time  $t$ , and the order has not been canceled by  $x_1$ , then the seller will get his payment and the buyer will get his merchandise; otherwise, the buyer  $x_1$  would be fully refunded.*

Also, suppose that enterprise  $E_1$  has the following policy,  $P_1$ , regarding such purchases:

**Policy  $P_1$ :** *Agents in  $E_1$  are allowed to purchase from  $E_2$  in accordance with contract  $P_{12}$ , if the following conditions are met:*

- *$x_1$  must have a budget assigned to it by a designated `budgetOfficer`.*
- *a purchase order (PO) can be issued only if the balance in  $x_1$  budget exceeds the scrip amount included in the offer. If this is the case,  $x_1$ 's budget is reduced accordingly.*
- *if the merchandise requested by  $x_1$  is not delivered for whatever reason, its budget is increased by the refunds received under the contract between  $E_1$  and  $E_2$ .*

And suppose that enterprise  $E_2$  has its own policy,  $P_2$ , regarding responses to purchase orders:

**Policy  $P_2$ :** *Agents in  $E_2$  are allowed to respond to purchase orders in accordance with contract  $P_{12}$ , but the arrival of purchase orders and of payments, as well as the delivery of the purchased goods, must be monitored by a designated agent called `auditor`.*

It is clear that any purchase transaction between these two enterprises must conform to all three policies above. The problem is how can such three distinct policies be brought to bear on a single transaction?

One may attempt to deal with this problem by compiling a textual *composition* of the three policies into a single one, and then subject all transactions between the two enterprise to this composition. This technique has been proposed, for access-control policies, by several authors [4,2]. But textual compositions

won't do for B2B commerce, for the following reasons: First, the two enterprises are likely to consider their policies  $P_1$  and  $P_2$  confidential, and may be reluctant to reveal them to a single composer. Second, these policies would be usually formulated separately, by different organizations, and without any knowledge of each other; therefore, it might be difficult to ensure that they are compatible with each other and composable. Third, a hard textual composition would be an impediment to the independent evolution of the policies of individual enterprises.

So, we need a mechanism that allows different enterprises to interoperate dynamically, under a mutually agreed contract, without sacrificing the confidentiality of their internal policies, and without losing their ability to make changes in these policies at will. We already addressed certain aspects of this problem, using a message exchange mechanism called Law-Governed Interaction (LGI) [6, 7]. In particular, a technique for establishing intra-enterprise policies, like  $P_1$  and  $P_2$  above, has been presented in [1]. And the issue of interoperability between enterprises has been addressed in [10].

But these techniques suffer from several difficulties. Chief among them is that although LGI itself is strictly decentralized, the mechanism proposed in [10] relies on a single agent to enforce the contract over all transactions between any pair of enterprises. Such a centralized enforcer might become a bottleneck, and a dangerous single point of failure when the number of the transactions grows. This, and another difficulty with the previous LGI-based mechanism are addressed in this paper.

The rest of the paper is organized as follows. We start, in Section 2, with a brief description of the concept of LGI, and then describe extensions to LGI designed to solve our interoperability problem. In Section 3 we demonstrate the proposed techniques by formulating the three intertwined example policies as *laws* under LGI, and by showing how they are enforced in a scalable manner. We conclude in Section 4.

## 2 Law-Governed Interaction (LGI)—an Overview

Broadly speaking, LGI is a message-exchange mechanism that allows an *open group* of distributed agents to engage in a mode of interaction *governed* by an explicitly specified policy, called the *law* of the group. The messages thus exchanged under a given law  $\mathcal{L}$  are called  $\mathcal{L}$ -messages, and the group of agents interacting via  $\mathcal{L}$ -messages is called a *community*  $\mathcal{C}$ , or, more specifically, an  $\mathcal{L}$ -community  $\mathcal{C}_{\mathcal{L}}$ .

By the phrase “open group” we mean (a) that the membership of this group (or, community) can change dynamically, and can be very large; and (b) that the members of a given community can be heterogeneous. In fact, we make no assumptions about the structure and behavior of the agents<sup>1</sup> that are members of a given community  $\mathcal{C}_{\mathcal{L}}$ , which might be software processes, written in an arbitrary

<sup>1</sup> Given the popular usages of the term “agent,” it is important to point out that we do not imply by it either “intelligence” nor mobility, although neither of these is being ruled out by this model.

language, or human beings. All such members are treated as black boxes by LGI, which deals only with the interaction between them via  $\mathcal{L}$ -messages, making sure it conforms to the law of the community. (Note that members of a community are not prohibited from non-LGI communication across the Internet, or from participation in other LGI-communities.)

For each agent  $x$  in a given community  $\mathcal{C}_{\mathcal{L}}$ , LGI maintains, what is called, the *control-state*  $\mathcal{CS}_x$  of this agent. These control-states, which can change dynamically subject to law  $\mathcal{L}$ , enable the law to make distinctions between agents, and to be sensitive to changes in their state. The semantics of control-states for a given community is defined by its law, and could represent such things as the role of an agent in this community, and privileges and tokens it carries. For example, under law  $\mathcal{L}_1$  to be introduced in Section 3, as a formalization of our example  $P_1$  policy, the term `budget(val)` in the control-state of an agent denotes that this agent has been assigned a budget in amount of `val`.

The rest of this section is organized as follows. We start in Section 2.1 with a brief discussion of the concept of law, emphasizing its local nature, and with a description of the decentralized LGI mechanism for law enforcement. Then, in Section 2.2 we discuss the changes required in LGI for it to support inter-enterprise electronic commerce. We do not discuss here several important aspects of LGI, including its concept of *exceptions*, its treatment of certificates, the deployment of  $\mathcal{L}$ -communities, the expressive power of LGI, and its efficiency. For these issues, and for implementation details, the reader is referred to [7, 1].

## 2.1 Laws, and their Enforcement

Generally speaking, the law of a community  $\mathcal{C}$  is defined over a certain types of events occurring at members of  $\mathcal{C}$ , mandating the effect that any such event should have—this mandate is called the *ruling* of the law for a given event. The events subject to laws, called *regulated events*, include (among others): the *sending* and the *arrival* of an  $\mathcal{L}$ -message; and the *coming due of an obligation* previously imposed on a given agent. The operations that can be included in the ruling of the law for a given regulated event are called *primitive operations*. They include, operations on the control-state of the agent where the event occurred (called, the “home agent”); operations on messages, such as `forward` and `deliver`; and the imposition of an obligation on the home agent.

Thus, a law  $\mathcal{L}$  can regulate the exchange of messages between members of an  $\mathcal{L}$ -community, based on the control-state of the participants; and it can mandate various side effects of the message-exchange, such as modification of the control states of the sender and/or receiver of a message, and the emission of extra messages, for monitoring purposes, say.

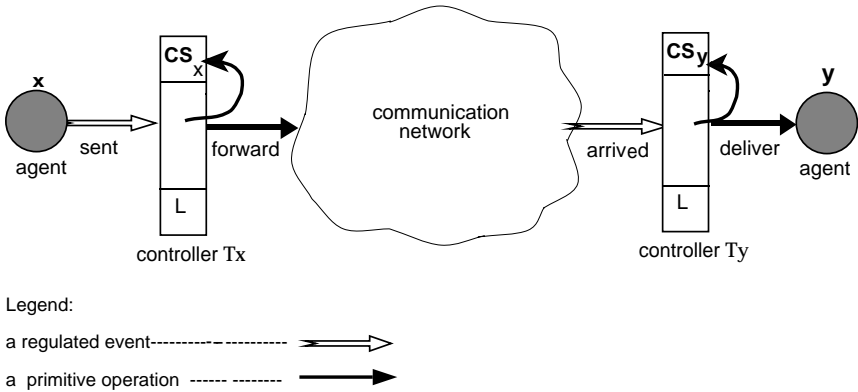
*On The Local Enforceability of Laws:* Although the law  $\mathcal{L}$  of a community  $\mathcal{C}$  is *global* in that it governs the interaction between all members of  $\mathcal{C}$ , it is enforceable *locally* at each member of  $\mathcal{C}$ . This is due to the following properties of LGI laws:

- $\mathcal{L}$  only regulates local events at individual agents,

- the ruling of  $\mathcal{L}$  for an event  $e$  at agent  $x$  depends only on  $e$  and the local control-state  $\mathcal{CS}_x$  of  $x$ .
- The ruling of  $\mathcal{L}$  at  $x$  can mandate only local operations to be carried out at  $x$ , such as an update of  $\mathcal{CS}_x$ , the forwarding of a message from  $x$  to some other agent, and the imposition of an obligation on  $x$ .

The fact that the same law is enforced at all agents of a community gives LGI its necessary global scope, establishing a *common* set of ground rules for all members of  $\mathcal{C}$  and providing them with the ability to trust each other, in spite of the heterogeneity of the community. And the locality of law enforcement enables LGI to scale with community size.

*Distributed Law-Enforcement:* Broadly speaking, the law  $\mathcal{L}$  of community  $\mathcal{C}$  is enforced by a set of trusted agents called *controllers*, that mediate the exchange of  $\mathcal{L}$ -messages between members of  $\mathcal{C}$ . Every member  $x$  of  $\mathcal{C}$  has a controller  $\mathcal{T}_x$  assigned to it ( $\mathcal{T}$  here stands for “trusted agent”) which maintains the control-state  $\mathcal{CS}_x$  of its client  $x$ . And all these controllers, which are logically placed between the members of  $\mathcal{C}$  and the communications medium, as illustrated in Figure 1, carry the *same law*  $\mathcal{L}$ . Every exchange between a pair of agents  $x$  and  $y$  is thus mediated by *their* controllers  $\mathcal{T}_x$  and  $\mathcal{T}_y$ , so that this enforcement is inherently decentralized. However, several agents can share a single controller, if such sharing is desired.



**Fig. 1.** Enforcement of the law.

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 any machine, anywhere in the network. We have implemented a *controller-service*, which maintains a set of active controllers. To be effective in a widely distributed enterprise, this set of controllers need to be well dispersed geographically, so that it would be possible to find controllers that are reasonably close to their prospective clients.

*On the basis for trust between members of a community:* For a member of an  $\mathcal{L}$ -community to trust its interlocutors to observe the same law, one needs the following assurances: (a) messages are securely transmitted over the network; (b) the exchange of  $\mathcal{L}$ -messages is mediated by controllers interpreting the *same law*  $\mathcal{L}$ ; and (c) all these controllers are *correctly implemented*. If these conditions are satisfied, then it follows that if  $y$  receives an  $\mathcal{L}$ -message from some  $x$ , this message must have been sent as an  $\mathcal{L}$ -message; in other words, that  $\mathcal{L}$ -messages cannot be forged.

Secure transmission is carried out via traditional cryptographic techniques. To ensure that a message forwarded by a controller  $\mathcal{T}_x$  under law  $\mathcal{L}$  would be handled by another controller  $\mathcal{T}_y$  operating under the *same law*,  $\mathcal{T}_x$  appends a one-way hash [9]  $H$  of law  $\mathcal{L}$  to the message it forwards to  $\mathcal{T}_y$ .  $\mathcal{T}_y$  would accept this as a valid  $\mathcal{L}$ -message under  $\mathcal{L}$  if and only if  $H$  is identical to the hash of its own law.

As to the correctness of controllers, we assume here that every  $\mathcal{L}$ -community is willing to trust the controllers certified by a given certification authority (CA), which is specified by law  $\mathcal{L}$ . And, every pair of interacting controllers must first authenticate each other by means of certificates signed by this CA. In our case of B2B commerce, for example, it is likely that law  $\mathcal{L}_1$ , which is the formal expression under LGI of policy  $P_1$ , would require the use controllers maintained by enterprise  $E_1$ , and certified by a CA managed by that enterprise—and similarly for law  $\mathcal{L}_2$ . Also, it is likely that law  $\mathcal{L}_{12}$ , which is the formal expression of the contract between the two enterprises, would require the use of controllers maintained by an organization, and certified by a CA, trusted by both enterprises.

## 2.2 Providing for Regulated Interoperability Between Enterprises

We start, in Section 2.2, with the features added to LGI to support direct interoperability between communities operating under different laws, with no contract between them (this work is a refinement of a concept introduced in [10]). Then, in Section 2.2, we introduce the concept of *virtual agent*, which is necessary for the scalable enforcement of inter-enterprise contracts.

**Interoperability Under LGI** By “interoperability” we mean here, the ability of an agent  $x$  operating under law  $\mathcal{L}$  to exchange messages with and agent  $x'$  operating under law  $\mathcal{L}'$ . To support interoperability we introduce the following concepts into LGI: (a) *portal*, which is the specification within a given law  $\mathcal{L}$ , of a different law  $\mathcal{L}'$  that  $\mathcal{L}$  can interoperate with; (b) an *export/import* mechanism used for carrying out interoperation between different laws.

*Portals* To enable interoperability between communities governed by laws  $\mathcal{L}$  and  $\mathcal{L}'$ , each of these laws needs to specify a portal for the other. The portal in law  $\mathcal{L}$ , in particular, would have the form

$$\text{portal}(p', hL', c'),$$

where  $p'$  is the local name of this portal, to be used within law  $\mathcal{L}$ ;  $hL'$  is the one-way hash of law  $\mathcal{L}'$ , which serves as its identifier; and  $c'$  is the CA employed by law  $\mathcal{L}'$  for the certification of the controllers authorized to interpret it. Portals are included in the *Preamble* section of the law, as in Figure 3. The set of all such clauses in a given law is called the initial portal table of the law. This table can be changed dynamically, in a manner left out of this paper due to space limitations.

*Export/Import:* The actual transfer of a message  $m$  from an agent  $x$  operating under law  $\mathcal{L}$  to an agent  $x'$  operating under law  $\mathcal{L}'$ , is carried out via a primitive operation, `export`, and via an event, `imported`, as follows:

An operation

$$\text{export}(x, m, [x', p']),$$

must be invoked by  $x$  under law  $\mathcal{L}$ , where  $p'$  is the local name for the portal of  $\mathcal{L}'$  (defined in  $\mathcal{L}$ ). This operation will initiate a handshake with the controller serving agent  $x'$  under law  $\mathcal{L}'$ ; if the protocol completes successfully<sup>2</sup>, then the following event will be triggered at  $x'$ :

$$\text{imported}([x, p], m, x'),$$

where  $p$  is the name of the portal for law  $\mathcal{L}$ , defined in  $\mathcal{L}'$ .

**Enforcement of Inter-Enterprise Contracts via Virtual Agents** The mechanism proposed in [10] for enforcing contracts over inter-enterprise interactions employed a *single* mediator for *all* interactions between any agent  $x_1^i$  of enterprise  $E_1$  and any agent  $x_2^j$  of  $E_2$ . This mediator, operating under the given contract law  $\mathcal{L}_{12}$ , must be created before the contract is enacted, and must be maintained for the lifetime of the contract. Such centralized enforcement of inter-enterprise contracts can easily become a bottleneck, when the number of transactions between a pair of enterprises grows, and it should not be employed unless it is truly necessary.

Centralized contract enforcement might, indeed, be necessary when the contract imposes dependencies between different inter-enterprise transactions. But no centralization is required if the contract at hand treats each inter-enterprise transaction independently of others—as is the case with the example contract  $P_{12}$  in Section 1.

Such a contract could be enforced scalably under LGI, if different transactions are mediated by different agents trusted by both enterprises. In the case of  $P_{12}$ , in particular, a transaction initiated by a purchase order sent by  $x_1^i$  to  $x_2^j$  could be mediated by an agent  $x^{ij}$  operating under policy  $P_{12}$ , as illustrated in Figure 2.

A mediator such as  $x^{ij}$  must be created dynamically, when a transaction is initiated, possible by agents, such as  $x_1^i$ , which operates under a different law;

<sup>2</sup> The protocol completes successfully if the information provided by controllers serving  $x$  and  $x'$  matches the information given by the portals  $p'$  and respectively  $p$ .

and it needs to be removed when the transaction terminates. We refer to such agents as *virtual agents*.

Virtual agents differ from regular LGI-agents by the way they are created; and by the fact that a virtual agent does not have an actor (such as a person, or a program) to drive it, but is driven by messages from other agents, such as  $x_1^i$  and  $x_2^j$ , that participate in the transaction under its jurisdiction. The creation of such light weight and ephemeral agents is discussed below, and their use is illustrated in the following section.

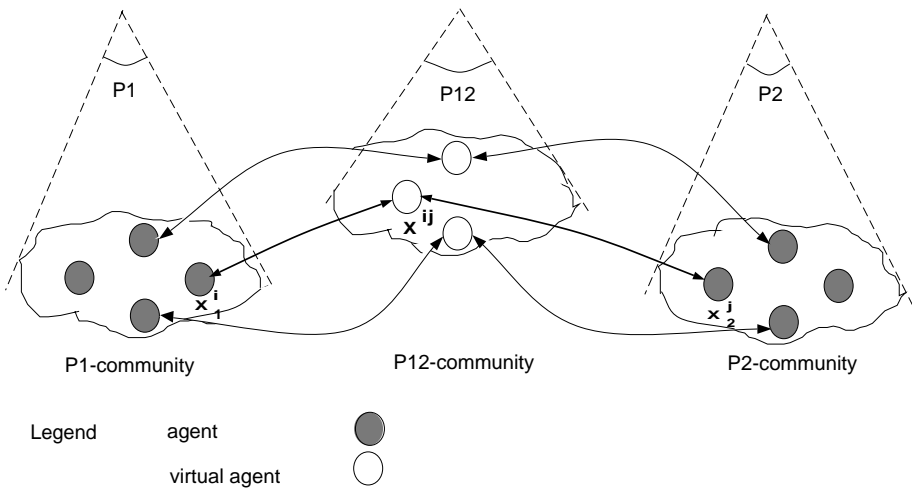


Fig. 2. Interoperation between communities using virtual agents.

*The Creation of Virtual Agents:* An agent  $x$  operating under law  $\mathcal{L}$ , with controllers certified by certification authority  $c$ , can create a virtual-agent  $x'$  under a possibly different law  $\mathcal{L}'$ , with controllers certified by a potentially different certification authority  $c'$ , by invoking the primitive operation:

`create(vAgent( $N'$ ), home( $T'$ ), portal( $P'$ ), arg( $A'$ )).`

Here,  $T'$  is the name of the controller which is to host the new agent;  $N'$  is the “private name” of this particular agent, which will distinguish it from other agents hosted by controller  $T'$ ;  $P'$  is the name of the portal, which identifies: (a) the law  $\mathcal{L}'$  under which this agent is to operate, and (b) the CA to be used for the authentication of controller  $T'$ ; finally,  $A'$  is an arbitrary list of arguments that is to be communicated to the newly created agent.

The very first event in the life of the newborn agent would be:

`created(by(I,P), arg( $A'$ )).`

Here,  $I$  is the id of the creator;  $P$  is the name of the creator’s portal, which must be defined in law  $\mathcal{L}'$  that governs the newborn; and  $A'$  is the argument list contained in the `create` operation that created this object.

What happens when this event occurs depends, of course, on law  $\mathcal{L}'$  that governs the newborn. For example, under law  $\mathcal{L}_{12}$  displayed in Figure 5, it is Rule  $\mathcal{R}1$  (discussed in detail in the following section) which determines the effect of a *created*-event at it.

### 3 A Case Study

We now show how the three policies  $P_1$ ,  $P_2$ , and  $P_{12}$ , introduced in Section 1, can be formalized, and thus enforced, under LGI. We note that  $P_1$  and  $P_2$  do not depend on each other in any way. Each of these policies provides for export to, and import from, the contract policy  $P_{12}$ , but they have no dependency on the internal structure of  $P_{12}$ .

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

*Law  $\mathcal{L}_1$ —the formalization of policy  $P_1$ .* Formally, under LGI, the community regulated by this law consists of the employees allowed to make purchases, and the designated agent `budgetOfficer`, who provides budget to employees. The law of this policy is presented in Figure 3. The law consists of two parts: preamble and rules. The preamble section describes the initial control state of the various agents operating in the community and lists the acceptable communication portals. In this example, the initial control state consists of a term `budget(0)`, denoting that budgets of every agent are initially zero. And law  $\mathcal{L}_1$  provides for communication only via portal `pt12`, the portal for the contract law.

The second part of the law consists of five rules, each followed by an explanatory comment (in italics). We discuss here only the first two rules handling budget allocation. (The rest of the rules will be presented as part of the description of a purchase transaction.) Under law  $\mathcal{L}_1$ , an agent designated as a `budgetOfficer` can increase the budget of any agent  $x_1$  by an arbitrary amount `Amt` by sending a message `giveBudget(Amt)` to it. The sending of such a message is authorized by Rule  $\mathcal{R}1$ , which forwards the message to its destination. By Rule  $\mathcal{R}2$ , when this message arrives at  $x_1$ , the budget of  $x_1$  will be incremented by `Amt`.

*Law  $\mathcal{L}_2$ —the formalization of policy  $P_2$ .* This law, displayed in Figure 4, defines one acceptable portal, `pt12`, and thus allows for interoperability only with the contract community. The community regulated by this law consists of employees allowed to respond to POs and the designated agent `auditor`.

*The contract law  $\mathcal{L}_{12}$ .* This law, which formalizes the contract policy, described informally in Section 1, is displayed in Figure 5. The preamble of  $\mathcal{L}_{12}$  defines

<p><i>Preamble:</i></p> <pre> initialCS([budget(0)]). authority(contractAuthority,publicKey). portal(pt12, hashOfContractLaw,contractAuthority). alias(budgetOfficer," budgetOfficer@e1.com"). </pre> <p><math>\mathcal{R}1.</math> sent(budgetOfficer, giveBudget(Amt), X1) :- do(forward).</p> <p><i>A giveBudget message sent by the budgetOfficer is forwarded immediately.</i></p> <p><math>\mathcal{R}2.</math> arrived(budgetOfficer,giveBudget(Amt),X1) :-              budget(Val)@CS, do(incr(budget(Val),Amt)), do(deliver).</p> <p><i>The arrival of an giveBudget(Amt) message causes the increase by Amt of the value Val held in the budget term; the message is also delivered to the receiver X1, to notify him of the increase.</i></p> <p><math>\mathcal{R}3.</math> sent(X1,M,X2) :- M==purchaseOrder(Specs,Scrip,T),              budget(Val)@CS, Val &gt; Scrip,              do(dcr(budget(Val),Scrip)), select(N,C,X12),              do(create(vAgent(N),home(C),portal(pt12),arg([M,X2]))),              do(+proxy(Specs,X2,X12)).</p> <p><i>A purchase transaction between X1 and X2 is initiated only if Scrip, the amount the X1 is willing to pay for the merchandise, is less than Val, the value of the sender's budget. If this is the case, the budget is decreased accordingly and a request to create a virtual agent N@C is issued.</i></p> <p><math>\mathcal{R}4.</math> sent(X1,M,X2) :- M==cancel(Specs),              proxy(Specs,X2,X12)@CS,do(export(X2,M,[X12,pt12])).</p> <p><i>A cancel message is exported immediately to the virtual agent X12.</i></p> <p><math>\mathcal{R}5.</math> imported([X12,pt12],M,X1) :-              proxy(Specs,X2,X12)@CS, do(deliver(X2,M,X1))              if M==refund(Specs,Amt) then do(incr(budget(Val), Amt)).</p> <p><i>A message, imported from X12 via portal pt12, is delivered to the destination. If the message represents a refund for Amt, then the budget is increased by Amt.</i></p>
--

**Fig. 3.** The law  $\mathcal{L}_1$  of enterprise  $E_1$

two portals  $pt1$  and  $pt2$ , allowing for interoperation with two communities: the  $\mathcal{L}_1$ -community (via  $pt1$ ), and the  $\mathcal{L}_2$ -community (via  $pt2$ ). And we will describe in the next paragraph how the rules of this law materialize the contract.

The community which operates under this law consists of virtual agents only. Each such virtual agent is created when a PO is emitted by a client, and is maintained for the duration of the purchase transaction. As we shall see, a virtual agent  $x_{12}$  acts as a trusted mediator between a client agent  $x_1$  and a vendor agent  $x_2$ , records the terms of the PO, and carries out the provisions of the contract.

<p><i>Preamble:</i></p> <pre> initialCS([]). authority(contractAuthority,publicKey). portal(pt12,hashOfContractLaw,contractAuthority)). alias(auditor,"auditor@trust.com"). </pre> <p><math>\mathcal{R}1.</math> <code>imported([X12,pt12],M,X2) :-</code>  <code>if M==[purchaseOrder(Specs,Price,T),X1] then</code>  <code>do(+proxy(Specs,X1,X12))</code>  <code>do(deliver),</code>  <code>do(deliver(X2,M,auditor)).</code></p> <p><i>A message imported from the contract portal is delivered to the auditor and to the home agent.</i></p> <p><math>\mathcal{R}2.</math> <code>sent(X2,M,X1) :- M==supply(Specs,Ticket),</code>  <code>do(deliver(X1, M, auditor)),proxy(Specs,X1,X12)@CS,</code>  <code>do(export(X2,M,[X12,pt12])).</code></p> <p><i>When a message is sent by an agent from the vendor enterprise to a client agent, the message is delivered to the designated auditor. A copy of the message is exported to X12 under the contract portal.</i></p>
--

**Fig. 4.** The law  $\mathcal{L}_2$  of enterprise  $E_2$

*The Progression of a Purchase Transaction* We explain now how these three laws function together, by means of a step-by-step description of the progression of a purchase transaction initiated by a

`purchaseOffer(specs, scrip, t)`

message sent by agent  $x_1$  of enterprise  $E_1$  (the vendor) to an agent  $x_2$  of  $E_2$  (the client).

1. The sending by  $x_1$  of a PO to  $x_2$  denotes the beginning of a new purchase transaction between the two parties and is handled by law  $\mathcal{L}_1$  (see Rule  $\mathcal{R}3$  of  $\mathcal{L}_1$ ) as follows. If the budget of  $x_1$  is smaller than the amount specified by `scrip`, then this PO is simply ignored. Otherwise a request is issued to create a virtual agent under the contract-law  $\mathcal{L}_{12}$ . The arguments passed to the newly created agent are the original PO, together with  $x_2$ , the identity of the vendor agent.
2. As a result of this request, a new virtual agent, to be called here  $x_{12}$ , is spawned. And immediately, a `created` event, which is handled by Rule  $\mathcal{R}1$  of  $\mathcal{L}_{12}$ , is triggered at  $x_{12}$ . This rule distinguishes between two cases. If the creation request did not come through `pt1`, the authorized portal of  $\mathcal{L}_1$ , then the ruling calls for the immediate removal of the virtual agent. Otherwise, the agent is maintained and will serve, as we shall see, as a trusted intermediary between the two parties. In the latter case, the following additional actions are called for: (a) the terms of the order are recorded in the control state of the virtual agent by a term `order`; (b) an obligation to end the transaction

*Preamble:*

```

initialCS([]).
authority(e1Authority,pk1).
authority(e2Authority,pk2).
portal(pt1, hashOfE1Law, e1Authority).
portal(pt2, hashOfE2Law, e2Authority).

```

$\mathcal{R}1$ . `created`(by([X1,PT1]),arg([purchaseOrder(Specs,Scrip,T),X2])) :-  
     if (PT1 !=pt1) then do(remove)  
     else  
         (do(+order(X1,Specs,Scrip,X2)),  
         do(imposeObligation(endTransaction,T)),  
         do(export(X1,M,[X2,pt2]))).

*A virtual agent is maintained only if the request comes through pt1, the portal of enterprise E<sub>1</sub>. If this is the case, the arguments are processed as follows: (a) the terms of the order are recorded in the control state of the virtual agent; (b) an obligation to end the transaction is set to fire at delivery time; and (c) the original purchaseOrder is exported to the vendor X2.*

$\mathcal{R}2$ . `imported`([X2,pt2],supply(Specs,Ticket),X12) :-  
     order(X1,Specs,Scrip,X2)@CS,  
     do(+goods(Ticket)).

*When a supply message is imported by the virtual agent, X12, the ticket denoting the ordered merchandise is stored in its control state by a term goods.*

$\mathcal{R}3$ . `imported`([X1,pt1],cancel(Specs),X12) :-  
     order(X1,Specs,Scrip,X2)@CS,  
     do(export(X12,refund(Specs,Scrip\*0.9),[X1,pt1])),  
     do(export(X12,cancelled(Specs,Scrip\*0.1),[X2,pt2])),  
     do(remove)

*A cancel imported before the delivery time is processed as follows. The client, X1, is refunded the amount paid minus a 10% penalty. The vendor, X2, is notified and receives scrip amounting to 10% of the order price. Finally, the virtual agent, X12, not longer needed is removed.*

$\mathcal{R}4$ . `obligationDue`(endTransaction) :- order(X1,Specs,Scrip,X2)@CS,  
     if goods(Ticket)@CS then  
         (do(export(X12,supply(Specs,Ticket),[X1,pt1])),  
         do(export(X12,payment(Specs,Scrip),[X2,pt2])))  
     else do(export(X12,refund(Specs,Scrip),[X1,pt1]))  
     do(remove).

*The obligation endTransaction fires at delivery time. The following operations are triggered by this event. If term goods(Ticket) is present in the control state, meaning that the vendor supplied the goods, then a supply message containing the Ticket is exported to the client X1, and a payment message containing Scrip is exported to the vendor X2. Otherwise, the client is fully refunded. In both cases the virtual agent, is destroyed.*

**Fig. 5.** The law  $\mathcal{L}_{12}$  of the contract between  $E_1$  and  $E_2$

- is set to fire at delivery time; and (c) the original PO is exported to the vendor  $x_2$ .
3. When a `purchaseOrder` message exported by  $x_{12}$  is imported into  $x_2$  via `pt12`, the authorized contract portal, it is immediately delivered to the vendor  $x_2$  and to the `auditor` (Rule  $\mathcal{R}1$  of  $\mathcal{L}_2$ ).
  4. According to law  $\mathcal{L}_2$ , agent  $x_2$  can respond to a PO by a `supply(specs,ticket)` message<sup>3</sup>, where `ticket` denotes the requested merchandise. By Rule  $\mathcal{R}2$  of  $\mathcal{L}_2$ , the sending of such a message triggers two operations: (a) the message is exported to  $x_{12}$  and (b) a copy of the message is delivered to the `auditor`.
  5. An import of the `supply(specs,ticket)` response into the virtual agent causes the merchandise to be temporarily stored into the control state in a term `goods` (Rule  $\mathcal{R}2$  of  $\mathcal{L}_{12}$ ).
  6. Under the contract between the two enterprises a client is allowed to cancel an order before the specified delivery time. We will assume for now that the client exercised its right and sent a `cancel` message to the vendor. By Rule  $\mathcal{R}4$  of  $\mathcal{L}_1$  such a message is immediately exported to the virtual agent,  $x_{12}$ , for disposition.
  7. In accordance with the contract law, a `cancel` message imported into the virtual agent *before the delivery time* triggers the execution of the following operations: A `refund` message carrying `scrip` amounting to 90% of the price paid is exported to the client. A `cancelled` message is exported to the vendor to notify him of the outcome. Finally, the virtual agent,  $x_{12}$ , no longer needed is removed (Rule  $\mathcal{R}3$  of law  $\mathcal{L}_{12}$ ).
  8. However, if  $x_1$  does not cancel the order, then the obligation `endTransaction` fires at the virtual agent at the specified delivery time<sup>4</sup>. The ruling for this event mandates the following actions: If term `goods(Ticket)` is present in the control state, meaning that the vendor supplied the goods, then a `supply` message containing the `ticket` is exported to the client  $x_1$ , and a `payment` message containing `scrip` is exported to the vendor  $x_2$ . Otherwise,  $x_1$  is fully refunded. In both cases, the virtual agent, is destroyed. (Rule  $\mathcal{R}4$  of  $\mathcal{L}_{12}$ ).
  9. Finally, the import of a `refund` message into  $x_1$  causes the budget of  $x_1$  to be increased by the amount specified in the message (Rule  $\mathcal{R}5$  of  $\mathcal{L}_1$ ).

*Discussion* This case study makes the following simplifying assumptions: (1) all three policies use the same set of messages, and (2) the enterprises policies  $P_1$  and  $P_2$  allows for interoperation only with  $P_{12}$ . These assumptions are not intrinsic to the proposed model and were adopted only in order to make the example as simple as possible.

---

<sup>3</sup> To keep the example simple we do not describe here the case when the vendor declines a PO

<sup>4</sup> If the client cancelled in time then, by Rule  $\mathcal{R}3$  of law  $\mathcal{L}_{12}$  the virtual agent is removed, and thus the obligation dose not fire any longer

## 4 Conclusion

We have argued that any attempt to regulate inter-enterprise electronic commerce must recognize the fact that such commerce is inherently subject to a combination of several heterogeneous policies: the internal policies of the interacting enterprises, and the policy that expresses the contract between them. To ensure that every inter-enterprise transaction conforms to such a combination of policies, we employed the LGI mechanism—that supports a formal and enforced concept of a policy—extending it in two ways. First, we introduced the *export/import* mechanism, and the concept of *portal*, which allows agents operating under different policies to interoperate. Second, we introduced the concept of *virtual agent*, which can be created dynamically, at the beginning of an inter-enterprise transaction, to handle this transaction subject to the contract-law between the enterprises. What is new about this concept of virtual agent, relative to the conventional concept of a *trusted intermediary* [3, 5, 8], is: (a) its dynamic and ephemeral nature, which contributes to scalability; and (b) while operating under the contract-law, a virtual agent interoperates with agents operating under the laws of the individual enterprises.

## References

1. X. Ao, N. a Minsky, and V. Ungureanu. Formal treatment of certificate revocation under communal access control. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 116–129, Oakland, California, May 2001.
2. 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.
3. 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.
4. L. Gong and X. Qian. Computational issues in secure interoperation. *IEEE Transactions on Software Engineering*, pages 43–52, January 1996.
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. N.H. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, February 1991.
7. N.H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *TOSEM, ACM Transactions on Software Engineering and Methodology*, 9(3):273–305, July 2000.
8. M. Roscheisen and T. Winograd. A communication agreement framework for access/action control. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, May 1996.
9. B. Schneier. *Applied Cryptography*. John Wiley and Sons, 1996.
10. V. Ungureanu and N.H. Minsky. Establishing business rules for inter-enterprise electronic commerce. In *Proc. of the 14th International Symposium on Distributed Computing (DISC 2000)*; Toledo, Spain; LNCS 1914, pages 179–193, October 2000.