Regulating Agent Involvement in Inter-Enterprise Electronic Commerce

Avigdor Gal[*] Naftaly H. Minsky[†]
Victoria Ungureanu
Rutgers University[‡]
Email: avigal@rci.rutgers.edu, {minsky,ungurean}@cs.rutgers.edu

Abstract

Current research on electronic-commerce focused mainly on fair and efficient transfer of money and goods between *a client and a vendor*. Inter-enterprise electronic commerce, though, bestows a more complex setting on the trade by adding a new dimension to the individual-merchant frame. The parties involved in a purchase are no longer *autonomous* entities, but are members of an organization whose rules of doing business they have to obey.

We propose a flexible approach towards regulating agent involvement in inter-enterprise electronic commerce. The method is based on the concept of law-governed interaction (LGI) which makes a strict separation between a declarative, formal statement of a policy and its enforcement.

1 Introduction

Electronic commerce is conceived as one of the major channels for performing commerce on a global scale, and the area is rapidly evolving. Most electronic commerce transactions take place between a client and a vendor, but inter-enterprise commerce is becoming increasingly important as well. There are several reasons for enterprises to engage in electronic commerce. First, electronic commerce is envisioned as an efficient method for decreasing transaction costs. It also allows for the automation of many aspects of the commerce and for the acceleration of transaction execution due to a rapid, almost flawless transfer of information via computer networks. Finally, as more and more trading partners turn to electronic commerce, enterprises are practically forced to join the trend, in order to adapt to the new reality.

Current research on electronic-commerce focused mainly on fair and efficient transfer of money and goods between *a client and a vendor* [13, 2, 8]. Inter-enterprise electronic commerce, though, bestows a more complex setting on the trade by adding a new dimension to the individual-merchant frame. The parties involved in a purchase are no longer

[*]Work supported in part by ITECC, Information Technology and Electronic Commerce Clinic, Rutgers University

[†]Work supported in part by NSF grants No. CCR-98-03698

[‡]Avigdor Gal is with the Department of MSIS. Naftaly H. Minsky and Victoria Ungureanu are with the Department of Computer Science.

autonomous entities, but are members of an organization whose rules of doing business they have to obey. It is the formulation and enforcement of such rules that will concern us here.

Currently, the inter-enterprise commerce is accomplished by using EDI (Electronic Data Interchange) standard, which provides for interchange of business documents in a machine-readable form. The translation to/from the enterprise data representation to an EDI standard is done through COTS (Commercial-Off-The-Shelf) EDI aware software (like eMMT [10], TSIsoft [9], Pro_EDI [3], to name a few). In addition, these EDI-software packages provide for additional aspects of electronic commerce like user authentication, secure transmission over the network, transaction verification and non-repudiation.

The rules of the enterprise are usually built into the code of a *sentry* placed between the EDI-software and the rest of the enterprise, whose job is to validate, and perhaps monitor, the transaction initiated by various agents within the enterprise. Unfortunately, burying enterprise-rules into the code of such a sentry has several serious drawbacks. First, such rules are difficult to understand and to validate. Second, changes of such rules is expensive and error prone. Finally, the sentry that enforces such rules tends to be centralized—perhaps because of the difficulty in changing the code of several distributed sentries, whenever the enterprise-rules change. But such centralized enforcement of enterprise-rules is not scalable, and it becomes a bottleneck, and a dangerous *single point of failure*, for large distributed enterprises.

In this paper we show how the previously developed mechanism for establishing electronic commerce policies [6] may alleviate this difficulties. This mechanism, which is based on "law-governed interaction" (LGI) [4] makes a strict separation between the formal statement of a policy, which is called "law," and the enforcement of this law, which is carried out by a set of policy-independent trusted *controllers*. Under this scheme a new policy is created basically by formulating its law, and thus it is easy to deploy.

As a motivating case study, we examine an enterprise policy in which authorization of performing transactions can be delegated to agents subject to certain constraints. Specifically, we will consider a computer science department which maintains a long-term relationships with various vendors. Each department member has a budget from which (s)he may order merchandise. The department policy regarding the deployment and use of these budgets, to be called the *budgeted payment* ($\mathcal{BP}$) policy, mandates that:

- A member `x` may issue a purchase request to a vendor only if the value of his budget exceeds the fee for the merchandise and once a purchase request is issued `x`'s budget is reduced by the amount of the purchase. If for some reason the vendor does not honor the request, `x`'s budget is restored.
- Any department member may delegate part of his budget to other colleagues but not to students;
- Any order for more than $100 undergoes an authorization process.

As we shall see in Section 3 this policy can be easily formulated and efficiently enforced under LGI.

The rest of the paper is organized as follows. Section 2 introduces the basics of law governed interaction, and in Section 3 we show in detail how the budgeted payment policy can be implemented under LGI; Section 4 discusses some related work, and Section 5 concludes the paper.

2 Basics of Law-Governed Interaction (LGI)

Law governed interaction, first presented in [4], is a concept proposed for enforcing protocols or "policies" in distributed systems. In our view a policy $\mathcal{P}$ is 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 $\mathcal{G}$. Finally, $\mathcal{L}$ represents the "rules of engagements" between the members of group $\mathcal{G}$, formulated to be enforced locally, at each member. The law regulates: (a) the exchange of messages between each member of $\mathcal{G}$ and the rest of this group, and (b) the effect of this exchange on the control-state of each member. We next give a brief description of the policy components and of the distributed mechanism that enforces a policy.

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 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 $\mathcal{L}$.

The 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. Strictly speaking, when an event is said to occur at an agent `x`, it actually occurs at the controller $\mathcal{C}_x$ assigned to `x`. These operations include:

1. `sent(x,m,y)` — occurs when a $\mathcal{P}$-message `m` sent by `x` to `y` arrives at $\mathcal{C}_x$. The sender `x` is considered the *home* of this event.

2. `arrived(x,m,y)` — occurs when a $\mathcal{P}$-message `m` sent by `x` arrives at $\mathcal{C}_y$. The receiver `y` is considered the *home* of this event.

The 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*. It is only through these primitive operations that a regulated event can have any effect on an agent or an agent's control state. These operations include:

1. **Operations on the control-state:** These operations update the control-state of the home agent. They include: (1) `+t` which adds the term `t` to the control state; (2) `-t` which removes a term `t`; (3) `t1`$\leftarrow$`t2` which replaces term `t1` with term `t2`; (4) `incr(t(v),d)` which increments the value of the parameter `v` of a term `t` with quantity `d` (`v` and `d` are assumed here to be integers); and (5) `dcr(t(v),d)` which decrements the value `v` of a term `t` with quantity `d`.

2. **Operations on messages:**

- Operation `forward(x,m,y)` sends to $\mathcal{C}_y$ message `m` addressed to `y`, where `x` identifies the sender of the message (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.)
- Operation `deliver(x,m,y)` delivers the message `m` to the home-agent `y`[1], where `x` is the nominal sender of this message. (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.)

The Law: Abstractly speaking, the law $\mathcal{L}$ of a policy is a *function* that returns a ruling for every possible regulated-event that might happen at a given agent. The ruling returned by the law is a possibly empty sequence of primitive operations, which is to be carried out locally, at the location of the event. By default, an empty ruling implies that the event in question has no consequences—such an event is effectively ignored.

Concretely, under the current implementation of LGI, the law is defined by means of a Prolog-like program[2] `L` which, when presented with a goal `e`, representing a regulated-event at a given agent `x`, is evaluated in the context of the control-state of this agent, producing the list of primitive-operations representing the ruling of the law for this event. In addition to the standard types of Prolog goals the body of a rule may contain two distinguished types of goals that play a special role in the interpretation of the law. These are the *sensor-goals*, which allow the law to "sense" the control-state of the home agent, and the *do-goals* that contribute to the ruling of the law. A *sensor-goal* has the form `t@CS`, where `t` is any Prolog term. It attempts to unify `t` with a term in the control-state of the home agent. A *do-goal* has the form `do(p)`, where `p` is one of the above mentioned primitive-operations. It appends the term `p` to the ruling of the law.

2.1 The Distributed Law-Enforcement Mechanism

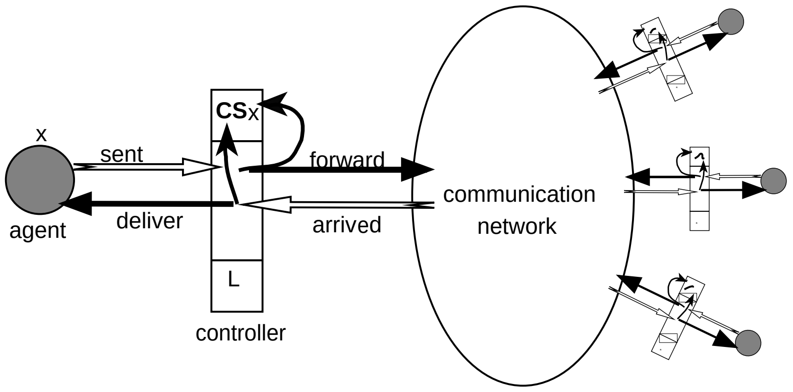
The most critical aspect of a policy, as defined above, is the assumption that its law is observed by all members of the policy-group $\mathcal{G}$. Given the openness and heterogeneity of $\mathcal{G}$, this assumption must be supported by strict enforcement of the law. That is, any exchange of $\mathcal{P}$-messages, once undertaken, satisfies law of $\mathcal{P}$. We next describe how such enforcement is carried out.

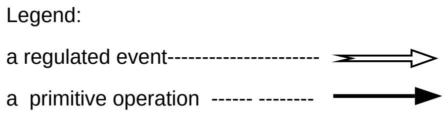
Law $\mathcal{L}_\mathcal{P}$ is enforced by a set of trusted entities called *controllers* that mediate the exchange of $\mathcal{P}$-messages between members of group $\mathcal{G}_\mathcal{P}$. For every active member `x` in $\mathcal{G}_\mathcal{P}$, there is a controller $\mathcal{C}_x$ logically placed between `x` and the communications medium, as illustrated in Figure 1. All these controllers have identical copies of law $\mathcal{L}_\mathcal{P}$, and each controller maintains the control-states of the agents under its jurisdiction.

Consider, for example, an agent `x` sending a $\mathcal{P}$-message `m` to an agent `y`, assuming that both agents have joined the policy-group $\mathcal{G}_\mathcal{P}$.(For a discussion of how does one join a policy-group the reader is referred to [6].) Message `m` is forwarded 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.

[1] `y` must be the home-agent in this case.

[2] Prolog is incidental to this model, and can, in principle, be replaced by a different, possibly weaker, language; for now, a restricted version of Prolog is being used.





Legend:
a regulated event----------------------
a primitive operation ------ --------

Figure 1: Enforcement of the Law

If, for example, this ruling calls for message `m` to be forwarded to `y`, then $\mathcal{C}_x$ would send `m` to the controller $\mathcal{C}_y$ assigned to `y`. Also, if the ruling calls the control-state $\mathcal{CS}_x$ to be updated, such update is carried out directly by $\mathcal{C}_x$.

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

To ensure that exchange of $\mathcal{P}$-messages is mediated by correct controllers interpreting the law of the policy $\mathcal{P}$, the messages sent across the network include a hash of the law, and would be digitally signed, using a pair of (RSA) keys assigned to each controller. This is sufficient to ensure that the law $\mathcal{L}_\mathcal{P}$ is not violated *inadvertently* by agents exchanging $\mathcal{P}$-messages. It is even sufficient to protect against *malicious* violations of the law, provided one can trust the OS-kernel of the hosts, which may be the case within the intranet of a given enterprise. As to the security of this mechanism over the Internet, the reader is referred to [7, 6].

3 Establishing the Budgeted Payment Policy ($\mathcal{P}_{\mathcal{BP}}$)

The components of this particular policy are as follows: the group $\mathcal{G}_{\mathcal{BP}}$ consists of the set of employees allowed to make purchases in the department, and the EDI-gateways for communicating with the vendors. There is often the case in practice that there are several such gateways since different vendors support: (a) different EDI standards (e.g. ASC X12, EDIFACT, and Uniform Communication Council (UCC) agreements, to name a few [14]), (b) different communication protocols [12], (c) different EDI-aware softwares, and (d) different transport mechanisms (either Value Added Networks (VANs) or the Internet). For transparency reasons, a gateway-member has as name the name of the vendor with which it communicates.

The law $\mathcal{L}_{\mathcal{BP}}$ is the set of rules described informally in Section 1. This law regulates three different aspects of the commercial activity in question, which will be described in detail below: (a) budget delegation, (b) purchase transaction per se, and (c) purchase authorization. Finally, the set $\mathcal{M}_{\mathcal{BP}}$ consists of all messages exchanged during these activities.

Budget delegation Under this policy, the value `val` of the attribute `budget(val)` from the control state of a member `x`, denotes the maximum amount `x` can delegate to others or use for purchases[3]. Members may delegate portions of their budget subject to the following constraints: (a) once a member `m` gives away a portion `p` of his budget, the budget is decreased by `p`, and (b) budgets cannot be delegated to students. The law implementing these requirements is presented in Figure 2 and detailed below.

An agent `x`, wishing to delegate a portion `p` of his budget to an agent `y`, sends to `y` the message `delegateBudget(p)`. By Rule $\mathcal{R}1$ the message is forwarded to the destination only if `x` has a term `budget(val)` in his control state and `val` exceeds `p`. Also, his `budget` term is decreased accordingly. Now, by Rule $\mathcal{R}2$, when a `delegateBudget(p)` arrives at the destination, the ruling distinguishes between two cases: first, if the destination agent is not a `student`, his budget is increased by `p`. If, however, the destination is a student the message is forwarded back to the sender `x`. By Rule $\mathcal{R}1$, `x`'s budget is restored to its previous value.

$\mathcal{I}$nitially: A member has in his control state an attribute budget(val), where `val` represents the maximum amount he can delegate to others or use for purchases. Additionally, a member may have in his control state an attribute `role(r)` which denotes his position in the organization; in our example `r` may take values in the set {`chair`,`professor`, `staff`, `researcher`, `student`}.

$\mathcal{R}1$.
```
sent(X, delegateBudget(P),Y) :-
        budget(Val)@CS, Val > P, do(dcr(budget(Val),P)),
        do(forward).
```

A `delegateBudget(P)` *message is forwarded to the destination if the sender has in his control state a term* `budget(Val)` *and* `Val` *is bigger than* `P`*. The* `budget` *of the sender is decreased by* `P`.

$\mathcal{R}2$.
```
arrived(X,delegateBudget(P),Y)) :- role(R)@CS,
        R=student →do(forward(Y,delegateBudget(P),X))
        ;
        (budget(Val)@CS, do(incr(budget(Val),P))).
```

When an `delegateBudget` *arrives at the destination* `Y`*, the value of the* `budget` *term is increased by* `P`*, provided* `Y` *is not a student. If* `Y` *is a student, the message is forwarded back to* `X`*, which will cause the restoration of* `X`*'s budget to its previous value.*

Figure 2: Rules Regulating Budget Delegation

Purchase transaction The rules regulating the purchase, displayed in Figure 3, mandate that: (a) a member may issue a purchase request to a vendor only if he has sufficient funds in his budget, and (b) once a purchase request is issued the client's budget is reduced by the

[3]For simplicity reasons, we do not deal here with the issue of how these budgets are innitially established.

amount of the purchase. For clarity, we do not deal here with how purchases are authorized; this requirement of the $\mathcal{BP}$-policy will be presented in the next paragraph.

Intuitively, a purchase transaction proceeds as follows. Consider a client `c` who wants to purchase from a vendor `v` a merchandise defined by `specs`, for which `c` is willing to pay an amount of 9 dollars. This is accomplished by `c` sending a `purchaseRequest(specs,9,v)` message to `v`. Due to Rule $\mathcal{R}3$, this message would be forwarded to `v` *only* if `c` has in his control state a `budget` term with a value in excess of 9 dollars; the value of the budget is automatically decreased by 9, thus effectively preventing the client to issue purchase orders exceeding his budget. By Rule $\mathcal{R}4$, when this message arrives at `v`, it will be delivered. If the vendor agrees with the terms of the order, she is assumed to send a `supply(specs,ticket)` message to the client, where `ticket` denotes the requested merchandise[4]. Rules $\mathcal{R}5$ and $\mathcal{R}6$ mandate that a reply of a vendor to a client is to be transmitted to the client without further ado. If, however, the vendor cannot supply the merchandise for whatever reason (the fee is too small, the merchandise is out of stock, etc) she is expected to respond with a `denyRequest(specs,9)` message. By Rule $\mathcal{R}7$ when such a message arrives at the client his budget is incremented by 9 dollars[5], thus restoring client's buying privileges.

$\mathcal{R}3$.
```
sent(C, purchaseRequest(Specs,Fee,V),V) :-
        budget(Val)@CS, Val>Fee, do(dcr(budget(Val),Fee)),
```

A `purchaseRequest` *message is forwarded to the destination if* `Fee`*, the amount paid, is less than* `Val`*, the value of the sender's budget.*

$\mathcal{R}4$.
```
arrived(C, purchaseRequest(Specs,Fee,V),V) :-
        do(+request(Specs,Fee,C)), do(deliver).
```

A `purchaseRequest` *message is delivered to the vendor* `V`*, and a term* `request(Specs,Fee,C)` *is added to the control state of the vendor to record that* `V` *received an order denoted by* `Specs` *from* `C`*.*

$\mathcal{R}5$.
```
sent(V,M,C) :- (M=supply(Specs,Ticket);M=denyRequest(Specs,Fee)),
        request(Specs,Fee,C)@CS,do(-request(Specs,Fee,C)),do(forward).
```

A `supply` *or a* `denyRequest` *message is forwarded to the destination only if the sender has a corresponding* `request` *term in the control state.*

$\mathcal{R}6$.
```
arrived(V,supply(Specs,Ticket),C) :- do(deliver).
```

A `supply` *message is delivered without further ado.*

$\mathcal{R}7$.
```
arrived(V,denyRequest(Specs,Fee),C) :-
        budget(Val)@CS,do(incr(budget(Val),Fee)), do(deliver).
```

If the vendor cannot honor the request, the client has his budget increased by `Fee` *upon the receipt of a* `denyRequest` *message.*

Figure 3: Rules Regulating Purchase Transactions

[4]We assume here that the merchandise is in digital form, e.g. an airplane ticket, a document. If this is not the case, the merchandise delivery is outside the scope of the law.

[5]In this example we assumed that the vendor will always respond to a client request. It is possible however, to "oblige" the vendor to send a **`denyRequest`** message, thus precluding the case of a client paying without receiving the merchandise. This feature of the LGI interaction is presented in [5].

Purchase Authorization We will turn now to the last requirement of the to $\mathcal{BP}$-policy namely that purchases of merchandise whose value exceeds a \$100 have to be authorized by a designated purchase officer. This enhancement is implemented by replacing Rule $\mathcal{R}3$ from Figure 3 with Rule $\mathcal{R}3'$ and by adding Rules $\mathcal{R}8$ and $\mathcal{R}9$. The law of the $\mathcal{BP}$-policy consists then of Rules $\mathcal{R}1$, $\mathcal{R}2$, $\mathcal{R}3'$, and $\mathcal{R}4$ through $\mathcal{R}9$.

The rules dealing with purchase authorization, displayed in Figure 4, operate as follows: by Rule $\mathcal{R}3'$, whenever a client `x` issues a purchase request exceeding \$100, his assigned controller will not forward the message to the vendor, but instead ask the `purchaseOfficer` to approve the transaction. Under this policy, the `purchaseOfficer` may send the following messages to the client `x` making the request:

- `authorized(specs,fee,v)` which denotes an approval for buying the merchandise defined by `specs` in amount of `fee` from vendor `v`.
- `notAuthorized(specs,fee,v)` denoting that the purchase is not approved.

By Rule $\mathcal{R}8$, when an `authorized` message arrives at `x`, the `purchaseRequest` will finally be sent to the vendor. If, however, `x` receives a `notAuthorized` message from the `purchaseOfficer` his budget is incremented by `Fee`, the amount `x` was willing to pay for the merchandise.

$\mathcal{R}3'$.
```
sent(X, purchaseRequest(Specs,Fee,V),V) :-
        budget(Val)@CS, Val>Fee, do(dcr(budget(Val),Fee)),
        Fee < 100 →
           do(forward) ;
           do(forward(X,authorizationRequest(Specs,Fee,V),purchaseOfficer)).
```

A `purchaseRequest` *message is forwarded to the destination if* `Fee`*, the amount paid, is less than \$100 and also less than* `Val`*, the value of the sender's budget. If* `Val` *exceeds 100, then an* `authorizationRequest` *message is sent to* `purchaseOfficer`*. In both cases the value of the budget is decreased by* `Fee`*.*

$\mathcal{R}8$.
```
arrived(purchaseOfficer,authorized(Specs,Fee,V),X) :-
        do(forward(X,purchaseRequest(Specs,Fee,V),V)), do(deliver).
```

If the purchase for `Specs` *is authorized by the proper* `purchaseOfficer` *a* `purchaseRequest` *message is forwarded to the vendor* `V`

$\mathcal{R}9$.
```
arrived(purchaseOfficer,notAuthorized(Specs,Fee,V),X) :-
        budget(Val)@CS, do(incr(budget(Val),Fee)), do(deliver).
```

If the purchase for `Specs` *is not authorized, the budget value is increased by* `Fee`*, the amount* `X` *was willing to pay for the merchandise.*

Figure 4: Rules Providing for Purchase Authorization

4 Related Work

The need for a mechanism for specifying security policies as an alternative to hard coding them into an application occurred to several researchers. Theimer, Nichols and Terry [15] introduced a concept of *generalized capabilities*. Such capabilities contain access control programs (ACP) encoding the security policy to be enforced with respect to this capability. When a server receives a request accompanied by such a generalized capability, it executes the ACP to determine whether the request is valid or not.

Blaze, Feigenbaum and Lacy [1] built a toolkit called PolicyMaker which can interpret security policies. An agent 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.
In both these approaches the rights a user has are *static*: they cannot be modified in accordance with its actions. Thus, a large range of policies, like for example the budgeted payment policy, in which the *state* of a user determines his rights, cannot be enforced.

Finally, Roscheisen and Winograd [11] call for an explicit formulation of a policy, called "commpact", which is written in a formal language. Unlike LGI, their work emphasis peer to peer relations, and it is unclear whether their mechanism can support global policies.

5 Conclusion

In this paper we propose a mechanism for regulating agents involvement in inter-enterprise electronic commerce. The mechanism supports intra-enterprise control over its trade activity by providing means for deployment and enforcement of commerce policies aligned with the (frequently modified) enterprise's business rules.

This work is part of a broader project which attempts to regulate the entire process of inter-enterprise commerce. This would include control over internal activities of both enterprises as well as control over inter-enterprise interaction itself. The rationale for regulating the intra-enterprise activities have been discussed in this paper. By extending the control to the inter-enterprise activity, we will be able to offer provisions like goods atomicity, certified delivery [16] and money back guarantee [6] which are currently not supported by the EDI-software.

References

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

[2] 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.

[3] Encomium Data International. Pro_EDI—translation and document management system. website: `http://www.proedi.com`.

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

[5] 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.

[6] 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.

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

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

[9] Trading Partners. Tsisoft. website: `http://www.tsisoft.com`.

[10] RealEDI. eMMT (Electronic Merchandise Management Tool). website: `http://www.realedi.com`.

[11] M. Rosheisen and T. Winograd. A communication agreement framework for access/action control. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1996.

[12] A. Segev, J. Porra, and M. Roldan. Financial EDI over the internet: Case study II. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, 1996.

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

[14] P.K. Sokol. *From EDI to Electronic Commerce—A Bussines Initiative.* Mc Graw-Hill, 1995.

[15] M. Theimer, D. Nichols, and Douglas Terry. Delegation through access control programs. In *Proceedings of Distributed Computing System*, pages 529–536, 1992.

[16] J.D. Tygar. Atomicity in electronic commerce (invited paper). In *ACM/IEEE Conference on Principles of Distributed Computation*, 1995.