Regularity-Based Trust in Cyberspace

Naftaly H. Minsky[*]

Rutgers University, New Bruswick NJ 08903, USA,
minsky@cs.rutgers.edu,
WWW home page: http:www.cs.rutgers.edu/ minsky/

Abstract. One can distinguish between two kinds of trust that may be placed in a given entity e (a person or a thing), which we call: *familiarity-based* trust and *regularity-based* trust. A familiarity-based trust in e is a trust based on personal familiarity with e, or on testimonial by somebody who is familiar, directly or indirectly, with e; or even on some measure of the general reputation of e. A regularity-based trust is based on the recognition that e belongs to a class, or a community, that is known to exhibits a certain regularity—that is, it is known that *all* members of this class satisfy a certain property, or that their behavior conforms to a certain law. These two types of trust play important, and complementary, roles in out treatment of the physical world. But, as we shall see, the role of regularity-based trust in out treatment of the cyberspace has been limited so far because of difficulties in establishing such trust it in this context. It is this latter kind of trust, which is the focus of this paper.
We will describe a mechanism for establishing a wide range of regularity-based trusts, and will demonstrate the effectiveness of this mechanism, by showing how it can enhance the trustworthiness of a certain type of commercial client-server interactions over the internet.

1 Introduction

One can distinguish between two kinds of trust that may be placed in a given entity e (a person or a thing), which we call: *familiarity-based* trust and *regularity-based* trust—or, for short, "F-trust" and "R-trust", respectively. A familiarity-based trust in e is a trust based on personal familiarity with e, or on testimonial by somebody who is familiar with e, directly or indirectly, or even on some measure of the general reputation of e [9]. A regularity-based trust, on the other hand, is based on the recognition that e belongs to a class, or a community, that is known to exhibits a certain regularity—that is, it is known that *all* members of this class satisfy a certain property, or that their behavior conforms to a certain law. Both types of trust play important, and complementary, roles in the physical world. But, as we shall see, the role of regularity-based trust in cyberspace has been limited so far, because of difficulties in establishing it in this context. It is this latter kind of trust, which is the focus of this paper.

To illustrate the nature of regularity-based trust, and its importance, we now consider three familiar examples of regularities (or, approximate regularities, to be exact) in the physical world, and we will discuss the trust they engender:

[*] Work supported in part by NSF grant No. CCR-98-03698

R1: *Everything that goes up must come down.*
R2: *Dollar bills are not forged.*
R3: *Drivers stop at an intersection when they see a red light.*

Although none of these regularities is absolute, they all generate degrees of trust which makes the world we live in simpler and more manageable. The trust engendered by R1 contributes to the predictability of the natural world; this, in spite of the fact that R1 is not satisfied by objects whose speed relative to Earth exceeds the *escape velocity*. The trust engendered by R2 simplifies commerce, in spite of the fact that dollar bills are sometime forged. The occasional violations of R2 does have an effect on the degree of trust emanating from it, as is evident from the fact that while one usually excepts individual dollar bills at their face value, one is likely to check $100 bills more carefully. Finally, it is hard to imagine vehicular traffic without the trust engendered by R3, which allows one to drive through a green light without slowing down. The unfortunately limited validity of R3 is the reason that careful drivers tend to take a quick look at their left and right before crossing an intersection at a green light.

The remarkable usefulness of regularity-based trust is due to the fact that it allows one to deal with a whole class of entities in a uniform manner, without having to be familiar with the individual member at hand—once an entity is recognized as a member. For example, we usually know nothing about the driver we encounter at an intersection, beyond the fact that he or she is driving a car on the road—and thus assumed to satisfy R3. This advantage of R-trust is not shared by familiarity-based trust, which deals with individuals. It is true that *public-key infrastructures* (PKIs), provides for massive and scalable authentication of the identity and roles of principals [19, 8] via distribution of certificates; but each such certificate require a degree of familiarity, by somebody, with the individual being certified. The advantage of F-trust, on the other hand, is that it can provide one with comprehensive information about a given individual; while R-trust provides just the common denominator of a class of entities, and thus cannot be very specific. For example, R3 tells us nothing about where would a given driver go after stopping at a red light, which we might know if we are familiar with him, or with her.

It is also important to note that R-trust often relies on certain F-trust, so that these two types of trusts are not quite independent. For example, our trust in R1 is based, for the non-physicists among us, on familiarity with our physics teachers, who told us about Newton's laws, and on the reputation of the physics textbooks we read. And our trust in R2 relies on the reputation of the bank in charge of printing money. We will see additional example of this phenomenon later.

Now, how are regularities established, and what are the reasons for trusting them? In the physical world, one can distinguish between two kinds of regularities: *natural regularities*, like R1 above, which are due to the laws of nature; and *artificial regularities*, like R2 and R3, induced by societal laws, policies and customs. Of course, passing a social law, or declaring an organizational policy, does not, by itself, generate a regularity. One needs some means for ensuring that this law or policy is actually obeyed. Such means include, self interest, social mechanism of enforcement, and physical constraints. For example, the main reason that traffic laws like R3 are generally complied with, is the personal danger in violating them—dangers from other cars, and from the police. And the reasons that money is generally not being forged is due to the physical difficulty of forging money bills (which is becoming easier lately, due to the improving printing technology), together with the fear of being caught in violating the social laws prohibiting forging.

The problem we are facing in establishing R-trust in cyberspace, is that many of the factors that support regularities in the physical world—like physical constraints, and the fear of being caught violating a law—do not exist, or are less effective, over the internet. Nevertheless, certain types of trust-generating regularities do exist in cyberspace, playing critical roles in it. The most important of these is the natural regularity of *complexity of computation*. That is, the universal difficulty of solving certain computational problems. This regularity gives rise to *cryptography* [20], which is, in turn, the foundation on which much of the current trust-management [4, 13] technology rests. But in spite of its critical importance for generating trust, the range of regularities that can be *directly* established via cryptography is rather limited. For example, although it is possible to represent money via digital certificates, cryptography alone cannot stop such certificates from being duplicated—which means that *physical cash* cannot be directly represented cryptographically. (We will return to this point later in this paper.)

It is also possible to establish *artificial regularities* in cyberspace, simply by means of *voluntary compliance* with declared policies. This is how important regularities such as the use of HTML for writing web pages, and the use of IP for communication, have been established all over the internet. But the effectiveness of voluntary compliance as a means for establishing regularities is severely limited. Indeed, for voluntary compliance with a given policy P to be reliable, the following two conditions need to be satisfied: (a) it must be the vested interest of everybody to comply with P; and (b) a failure to comply with P, by somebody, or somewhere, should not cause any serious harm to anybody else. If condition (a) is not satisfied (as under R2) then there is little chance for P to be observed everywhere, even if it is declared as a standard. And if condition (b) is not satisfied (as it is under R3) than one has to worry about someone not observing P, perhaps inadvertently, and thus causing harm to others.

Clearly, the only way to ensure conformance with a policy that does not satisfy conditions (a) or (b) above, is to *enforce* it. If the community subject to this policy is small enough, then such enforcement can sometimes be carried out by employing a *trusted third party* (TTP) to function as a kind of *reference monitor* [1], which mediates all the interactions subject to it. This has been done by several projects [3, 12], mostly in the context of distributed enterprise systems. But if the community is large, as is often the case over the internet, then the use of a single reference monitor is inherently unscalable, since it constitute a dangerous single point of failure, and could become a bottleneck if the system is large enough. This difficulty can be alleviated when dealing with static (stateless) policies, simply by replicating the reference monitor—as has been done recently in [14].

But replication is very problematic for *dynamic* (or, *stateful*) policies, which are sensitive to the history, or state, of the interactions being regulated. This is because every state change sensed by one replica needs to be propagated, synchronously, to all other replicas of the reference monitor. Such synchronous update of all replica could be very expensive, and is quite unscalable. And dynamic policies are very important for many trust-related applications. In particular, things like *separation of duties* [7, 10], the so called Chinese Wall (CW) policy [5], and the effects of the expiration or revocation of a certificate on the power of an agent holding it [2], are all dynamic. Dynamic policies are particularly important for fostering regularity-based trust in electronic commerce, as we will illustrate via an example in the following section. All such dynamic policies would be very hard to enforce scalably via centralized reference monitors, whether replicated or not.

It is the purpose of this paper to show that it is possible to establish a wide range of regularity-based trusts, dynamic and otherwise, by employing a genuinely decentralized approach to policy enforcement, which is based on the following principles:

1. *Policies are to be formulated* locally*, so that the ruling of the policy regarding operations by a given subject depends dynamically only on the history of* its own *interactions with the rest of the community.*
2. *The enforcement of a policy over a given community is to be carried out by a* set of trusted-third-parties*, one per member of the community, acting as reference monitors, all interpreting the same policy.*

In Section 3 we will outline a mechanism called *law-governed interaction* (LGI), which has been designed according to these principles, and whose current prototype implementation has already been applied to a wide range of distributed and dynamic policies. In Section 4 we present a case study that shows how the this mechanism can be used to support trustworthy e-commerce, by enforcing the example policies introduced in Section 2. We conclude in Section 5, with an examination of the complementary interrelationship between familiarity-based trust and regularity-based trust.

2 Towards Regularity-Based Trust in E-Commerce—an Example

The purpose of this section is to demonstrate the potential usefulness of R-trust in cyberspace, and the difficulty of establishing such trust via conventional techniques. For this purpose, we will postulate a certain form of commercial client-server interaction. And we will propose a couple of highly dynamic policies which should ease this interaction, making it more trustworthy—provided that they are enforced, and thus made into regularities.

Consider a large and open community C, distributed over the internet, whose members provide each other with *prepaid* services. Let the standard service request in this community be a message of a form `order(specs,fee)`, where `specs` specifies the nature of the requested service, and `fee` is the payment for the service requested, which can be very small—few cents, say. To ease the interaction between clients and servers, making it more efficient and more trustworthy, we introduce here two policies: (a) a policy regarding the treatment of e-cash, used as payment for services, and (b) a policy regarding the response by the server to an order. These policies are described informally, and motivated, below; they would be formalized via an LGI law in Section 4.

The Cash Handling (CH) Policy: Given the possibly small size of payments for services, the use of traditional kind of e-cash certificates is not practical. Because, as has been pointed out by Glassman et. al. [11], each such certificate would have to be validated by the issuing bank, whenever used, for fear of duplication. This is far too expensive for small payments. We would like, therefore, payments to be carried out just like cash payment in the physical world. This should be possible if the following cash handling (CH) policy is established as a regularity among all clients and servers.

1. *Each member of the community must be able to withdraw cash from an account he/she/it has in some bank, storing it in its own* electronic wallet*. (Banks used for this purpose must be authenticated by a specified CA).*
2. *The payment included in an order sent by agent x must be taken from the electronic wallet of x, just as physical cash is taken from one's physical wallet.*
3. *When a payment is obtained by server y, it is to be added to its electronic wallet; and can then be used by orders made by y, or for depositing in y's bank account.*

This policy, if enforced, would be an improvement over the Millicent protocol devised by Glassman et. al. [11] for a similar purpose. Under the Millicent protocol clients use vendor-signed *scrips* for payments. Such scrips are copyable, and thus needs to be validated by the vendor, for every query, which requires a centralized database of valid scrips. Moreover, the vendor has to send a new scrip to the client, as his change. All this involves substantial overhead, and is not very scalable because of the centralized database that it employs. Under our CH policy, however, none of this overhead would be necessary—provided that individual agents in community C can be trusted to observe this highly dynamic policy.

A Policy Regarding Server's Response (SR): Clients generally cannot predict the time it would take the server to carry out his order, in particular because the server may have a backlog of prior orders. Moreover, suppose that the server may end up declining the order, because it is too busy, or because it is unable or unwilling to carry it out for some other reasons. This situation presents the following difficulties for clients: First, once an order is sent, the client may be stuck, waiting indefinitely for the service to be performed, with his money tied up in an order that may eventually be declined. Second, the client has no certainty that if the order is declined his money will be ever returned.

To alleviate these difficulties, we would like to provide clients with the assurance that the following (SR) policy regarding response to service orders is satisfied for all their orders:

> *Once a client* `x` *sends a message* `order(specs,fee)` *to a server* `y`*, he is guaranteed to receive one of the following two responses within a* bounded delay `D`*, which is independent of (and generally much smaller than) the time it takes to carry out the requested service itself:*
> 1. *The message* `accept(specs,fee)`*, which signifies that the server agrees to carry out the service, for the specified fee.*
> 2. *The message* `decline(spec,fee),` *which is signifies that the server declines the service order,* returning*, in this message, the* `fee` *sent by the client.*

Note that SR has nothing to say about the time it should take to carry out the requested service, nor does it provide any assurances that the service will be carried out properly, or at all. Yet, if this policy can be established as a regularity, for any order, sent to any server in community C, it would provide the following advantages:

– An `accept(specs,fee)` message from a server `y` can be viewed as a commitment by `y` to carry out this service for the specified fee. If this message can be authenticated as coming from `y`, then it can be used by the client, *in a court of law*, as a proof of this commitment. (Authentication of such messages cab be accomplished via cryptography; the proof in a court of law is not a technical matter, of course.)

– The assurance, provided by SR, that every order will be either accepted or declined within `D` seconds, with the money returned in the latter case, allows a client to plan a "shopping expedition" — by trying out a sequence of servers, one each `D` seconds, until somebody accept the order.

3 The Concept of 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 and strictly enforced policy, called the *interaction-law* (or simply 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 an $\mathcal{L}$-community $\mathcal{C}_{\mathcal{L}}$ (or, simply, a *community* $\mathcal{C}$.)

We refer to members of an $\mathcal{L}$-community as *agents*[1], by which we mean autonomous actors that can interact with each other, and with their environment. An agent might be an encapsulated software entity, with its own state and thread of control, or a human that interacts with the system via some interface. Both the *subjects* and the *objects* of the traditional security terminology are viewed here as agents. A community under LGI is *open* in the following sense: (a) its membership can change dynamically, and can be very large; and (b) its members can be heterogeneous. For more details about LGI, and about its implementation, than provided by this overview the the reader is referred to [18], and to [23] for more recent results.

3.1 On the Nature of LGI Laws, and their Decentralized Enforcement

The function of an LGI law $\mathcal{L}$ is to regulate the exchange of $\mathcal{L}$-messages between members of a community $\mathcal{C}_{\mathcal{L}}$. Such regulation may involve (a) restriction of the kind of messages that can be exchanged between various members of $\mathcal{C}_{\mathcal{L}}$, which is the traditional function of access-control policies; (b) transformation of certain messages, possibly rerouting them to different destinations; and (c) causing certain messages to be emitted spontaneously, under specified circumstances, via a mechanism we call obligations.

[1] Given the currently popular usage of the term "agent", it is important to point out that we do not imply either "intelligence" nor mobility by this term, although we do not rule out either of these.

A crucial feature of LGI is that its laws can be *stateful*. That is, a law $\mathcal{L}$ can be sensitive to the dynamically changing *state* of the interaction among members of $\mathcal{C}_{\mathcal{L}}$. Where by "state" we mean some function of the history of this interaction, called the *control-state* (CS) of the community. The dependency of this control-state on the history of interaction is defined by the law $\mathcal{L}$ itself.

But the most salient and unconventional aspects of LGI laws are their strictly local formulation, and the decentralized nature of their enforcement. To motivate these aspects of LGI we start with an outline of a centralized treatment of interaction-laws in distributed systems. Finding this treatment unscalable, we will show how it can be decentralized.

On a Centralized Enforcement Interaction Laws: Suppose that the exchange of $\mathcal{L}$-messages between the members of a given community $\mathcal{C}_{\mathcal{L}}$ is mediated by a reference monitor $\mathcal{T}$, which is trusted by all of them. Let $\mathcal{T}$ consist of the following three part: (a) the law $\mathcal{L}$ of this community, written in a given language for writing laws; (b) a generic *law enforcer* $\mathcal{E}$, built to interpret any well formed law written in the given law-language, and to carry out its rulings; and (c) the control-state ($\mathcal{CS}$) of community $\mathcal{C}_{\mathcal{L}}$ (see Figure 1(a)). The structure of the control-state, and its effect on the exchange of messages between members of $\mathcal{C}_{\mathcal{L}}$ are both determined by law $\mathcal{L}$. .

This straightforward mechanism provides for very expressive laws. The central reference monitor $\mathcal{T}$ has access to the entire history of interaction within the community in question. And a law can be written to maintain any function of this history as the control-state of the community, which may have any desired effect on the interaction between community members. Unfortunately, this mechanism is inherently unscalable, as it can become a bottleneck, when serving a large community, and a dangerous single point of failure.

Moreover, when dealing with stateful policies, these drawbacks of centralization cannot be easily alleviated by replicating the reference monitor $\mathcal{T}$, as it is done in the Tivoli system [14], for example. The problem, in a nutshell, is that if there are several replicas of $\mathcal{T}$, then any change in $\mathcal{CS}$ would have to be carried out *synchronously* at all the replicas; otherwise x may be able to get information from other companies in set s, via different replicas. Such maintenance of consistency between replicas is very time consuming,and is quite unscalable with respect to the number of replicas of $\mathcal{T}$.

Fortunately, as we shall see below, law enforcement can be genuinely *decentralized*, and carried out by a distributed set $\{\mathcal{T}_x \mid x \mathbf{in} \mathcal{C}\}$ of, what we call, *controllers*, one for each members of community $\mathcal{C}$ (see Figure 1(b)). Unlike the central reference monitor $\mathcal{T}$ above, which carries the CS of the entire community, controller $\mathcal{T}_x$ carries only the *local control-state* $\mathcal{CS}_x$ of x—where $\mathcal{CS}_x$ is some function, defined by law $\mathcal{L}$, of the history of communication between x and the rest of the $\mathcal{L}$-community. In other words, changes of $\mathcal{CS}_x$ are strictly local, not having to be correlated with the control-states of other members of the $\mathcal{L}$-community, However, such decentralization of enforcement requires the laws themselves to be *local*, in a sense to be defined next.

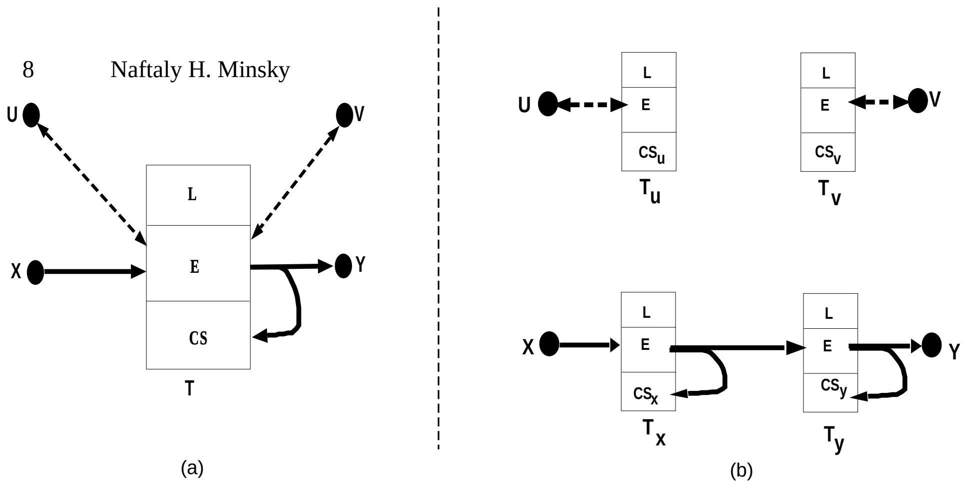

Fig. 1. Law Enforcement: (a) centralized version; (b) decentralized law enforcement under LGI

The Local Nature of LGI Laws: An LGI law is defined over a certain types of events occurring at members of a community $\mathcal{C}$ subject to it, mandating the effect that any such event should have. Such a mandate is called the *ruling* of the law for the given event. The events subject to laws, called *regulated events*, include (among others): the *sending* and the *arrival* of an $\mathcal{L}$-message; the occurrence of an *exception*; and the coming due of an *obligation*. The operations that can be included in the ruling for a given regulated event, called *primitive operations*, are all local with respect to the agent in which the event occurred (called, the "home agent"). They include, operations on the control-state of the home agent; operations on messages, such as `forward` and `deliver`; and the imposition of an obligation on the home agent. To summarize, an LGI law must satisfy the following locality properties:

– a law can regulate explicitly only *local events* at individual agents;
– the ruling for an event `e` at agent `x` can depend only on `e` itself, and on the *local control-state* $\mathcal{CS}_x$; and
– the ruling for an event that occurs at `x` can mandate only *local operations* to be carried out at `x`.

Decentralization of Law-Enforcement: As has been pointed out, we replace the central reference monitor $\mathcal{T}$ with a distributed set $\{\mathcal{T}_x \mid x \textbf{ in } \mathcal{C}\}$ of controllers, one for each members of community $\mathcal{C}$. Structurally, all these controllers are generic, with the same law-enforcer $\mathcal{E}$, and all must be trusted to interpret correctly any law they might operate under. When serving members of community $\mathcal{C}_{\mathcal{L}}$, however, they all carry the *same law* $\mathcal{L}$. And each controller $\mathcal{T}_x$ associated with an agent x of this community carries only the *local control-state* $\mathcal{CS}_x$ of x (see Figure 1(b)).

Due to the local nature of LGI laws, each controller $\mathcal{T}_x$ can handle events that occur at its client x strictly locally, with no explicit dependency on anything that might be happening with other members in the community. It should also be pointed out that controller $\mathcal{T}_x$ handles the events at x strictly sequentially, in the order of their occurrence, and atomically. This, and the locality of laws, greatly simplifies the structure of the controllers, making them easier to use as our *trusted computing base* (TCB).

Note that an LGI law cannot deal *directly* with an exchange of messages between agents. But it can regulate such an exchange, indirectly, by regulating the local events of the sending of a message, and of its arrival, at two different agents. Indeed, as we can see in Figure 1(b), every $\mathcal{L}$-message exchanged between a pair of agents x and y, passes through a pair of controllers: $\mathcal{T}_x$ and $\mathcal{T}_y$. This may seem to be less efficient than using a single central controller $\mathcal{T}$, when $\mathcal{T}$ is not congested. But as has been shown in [18], the decentralized mechanism is actually more efficient in a wide range of its applicability.

Finally, we point out that the LGI model is silent on the placement of controllers *vis-a-vis* the agents they serve, and it allows for the sharing of a single controller by several agents. This provides us with welcome flexibilities, which can be used to minimize the overhead of LGI under various conditions. A preliminary study [18] of such an optimization produced very positive results. The sharing of single controller by several agents, in particular, allows one to optimize law enforcement by devising various sharing strategies, including the use of a single centralized controller to mediate all message exchange among the members of a given community—where such centralized enforcement happen to be optimal.

3.2 A concept of enforceable obligation

Obligations, along with permissions and prohibitions, are widely considered essential for the specification of policies for financial enterprises [15]. The concept of obligation being employed for this purpose is usually based on conventional *deontic logic* [16], designed for the specification of normative systems, or on some elaborations of this logic, such as taking into account interacting agents [6]. These types of obligations allows one to reason about what an agent must do, but they provide no means for ensuring that what needs to be done will actually be done [21]. LGI, on the other hand, features a concept of obligation that can be enforced.

Informally speaking, an obligation under LGI is a kind of *motive force*. Once an obligation is imposed on an agent—which can be done as part of the ruling of the law for some event at it—it ensures that a certain action (called *sanction*) is carried out at this agent, at a specified time in the future, when the obligation is said to *come due*—provided that certain conditions on the control state of the agent are satisfied at that time. The circumstances under which an agent may incur an obligation, the treatment of pending obligations, and the nature of the sanctions, are all governed by the law of the community. For a detailed discussion of obligations the reader is referred to [18].

3.3 The Deployment of LGI

All one needs for the deployment of LGI is the availability of a set of trustworthy controllers, and a way for a prospective client to locate an available controller. This can be accomplished via one or more *controller-services*, each of which maintains a set of controllers, and one or more certification authorities that certifies the correctness of controllers. A prototype of such a controller service has been implemented, and is fully operational at Rutgers University—it is expected to be released by the end of the summer of 1993. This prototype can serve as the basis for the deployment of LGI within an enterprise. However, for effective deployment over the internet, the controller services need to be provided by some reliable commercial or governmental institutions.

Now, for an agent x to engage in LGI communication under a law $\mathcal{L}$, it needs to locate a controller, via a controller-service, and supply this controller with the law $\mathcal{L}$ it wants to employ. Once x is operating under law $\mathcal{L}$ it may need to distinguish itself as playing a certain role, or having a certain unique name, which would provide it with some distinct privileges under law $\mathcal{L}$. One can do this by presenting certain digital certificates to the controller, as we will see in our case study.

3.4 The Language for Formulating Laws:

Abstractly speaking, the law of a community is a function that returns a *ruling* for any possible regulated event that might occur at any one of its members. The ruling returned by the law is a possibly empty sequence of primitive operations, which is to be carried out locally at the *home* of the event. (By default, an empty ruling implies that the event in question has no consequences—such an event is effectively ignored.) Such a function can be expressed in many languages. We are currently using two languages for writing our laws, which are restricted versions of Prolog and of Java. In the rest of this paper we employ the Prolog-based langauge for writing LGI-laws.

3.5 The basis for trust between members of a community

Note that we do not propose to coerce any agent to exchange $\mathcal{L}$-messages under any given law $\mathcal{L}$. The role of enforcement here is merely to ensure that *any exchange of $\mathcal{L}$-messages, once undertaken, conforms to law $\mathcal{L}$*. In particular, our enforcement mechanism needs to ensure that a message received under law $\mathcal{L}$ has been sent under the same law; i.e., that it is not possible to forge $\mathcal{L}$-messages). For this one needs the following assurances: (a) that the exchange of $\mathcal{L}$-messages is mediated by correctly implemented controllers; (b) that these controllers are interpreting the *same law* $\mathcal{L}$; and (c) that $\mathcal{L}$-messages are securely transmitted over the network. This architectural idea has been later adopted by Stefik in his “trusted systems” work [22]—although Stefik’s work may have been done without any knowledge of the previous (1991) work of this author [17]. But Stefik applied this idea only in the limited context of *right management*.

Broadly speaking, these assurances are provided as follows. controllers used for mediating the exchange of $\mathcal{L}$-messages authenticate themselves to each other via certificates signed by a certification authority specified by the clause `cAuthority` of law $\mathcal{L}$ (as will be illustrated by the case study in the following section). Note that different laws may, thus, require different certification levels for the controllers used for its enforcement. Messages sent across the network must be digitally signed by the sending controller, and the signature must be verified by the receiving controller. 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 [20] `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.

Finally, note that although we do not compel anybody to operate under any particular law, or to use LGI, for that matter, one may be *effectively compelled* to exchange

$\mathcal{L}$-messages, if one needs to use services provided only under this law. For instance, if servers, in our case study in Section 4, are accept service-orders only via $\mathcal{SL}$-messages, then the clients will be compelled to send their orders via $\mathcal{SL}$-messages as well. Conversely, once clients commit themselves to using $\mathcal{SL}$-messages for their orders, servers would be effectively compelled to accept these commands only in this manner. Similarly, if a bank wants to serve agents in the $\mathcal{SL}$-community, it will have to do operate under this community itself. This is probably the best one can do in the distributed context, where it is impossible to ensure that all relevant messages are mediated by a reference monitor, or by any set of such monitors.

4 Establishing Trustworthy Client-Server Interaction—a Case Study

In this Section we introduce an LGI law called $\mathcal{SL}$ (for service law), which enforces the cash handling policy (CH) and the server's response policy (SR), introduced in Section 2—thus establishing them as regularities over the community that uses this law. We will also examine, in Section 4.1, the relationship between R-trust and F-trust, as both of these are employed in this example.

Law $\mathcal{SL}$, displayed in Figures 2 and 3, has two parts: *preamble* and *body*. The preamble of this law contains the following clauses: First there is the `cAuthority(pk1)` clause that identify the public key of the certification authority (CA) to be used for the authentication of the controllers that are to mediate $\mathcal{SL}$-messages. This CA is an important element of the trust between the agents that exchange such messages. Second, there is an `authority` clause, which provide the public key of the CA that would be acceptable under the this law for certifying banks; this CA is given a local name—"bankingCA," in this case—to be used within this law. Third, the `initialCS` clause specifies the initial control-state of all members in this community, which are to have the term `cash(0)`, signifying that each member of this community starts with zero balance in it electronic wallet (represented by the term `cash`).

The body of this law is a list of all its rules, each followed by a comment (in italic), which, together with the following discussion, should be understandable, at least roughly, even for a reader not well versed in our language for writing laws. These rules can be partitioned into three groups, dealing with different aspects of the law: (a) rules $\mathcal{R}1$ through $\mathcal{R}5$, which govern banking; (b) rules $\mathcal{R}6$ and $\mathcal{R}7$, which deals with the making of service orders; and (c) rules $\mathcal{R}8$ through $\mathcal{R}12$, which govern server's responses to orders. These three aspects of the law are now discussed in this order.

Banking: By Rule $\mathcal{R}1$, an agent can claim the role of a bank, and thus get the term `role(bank)` into its control-state—by presenting an appropriate certificate issued by `bankingCA`. By Rule $\mathcal{R}2$, such a bank—i.e., any agent that has the term `role(bank)` in its control-state—can send messages of the form `giveMoney(D)` to anybody in this community. When this message arrives at its destination `y`, then, by Rule $\mathcal{R}3$, the value of the `cash` term in the control-state of `y` will be incremented by `D`, and the message would be delivered to `y` itself, to notify him of this event. In this way a bank can provide arbitrary amount cash to any community member.

$\mathcal{P}$reamble:

```
cAuthority(pk1).
authority(bankingCA, pk2).
initialCS([cash(0)]).
```

$\mathcal{R}1$.

```
certified([issuer(bankingCA),subject(X),
      attributes([role(bank)])) :- do(+role(bank))).
```

an agent may claim the role of a bank by presenting an appropriate certificate issued by `ca`*).*

$\mathcal{R}2$.

```
sent(B,giveMoney(D),X) :- role(bank)@CS, do(forward).
```

A `giveMoney` *message can be sent by a bank to anybody*

$\mathcal{R}3$.

```
arrived(B,giveMoney(D),X)
   :- do(incr(cash(Bal1),D), do(deliver).
```

A `giveMoney(D)` *message arriving at any agent* `X` *would increment the cash of* `X` *by* `D`*.*

$\mathcal{R}4$.

```
sent(X,deposit(D),B) :- cash(Bal)@CS, ( D =< Bal),
      do(decr(cash(Bal),D)), do(forward).
```

A message `deposit(D)` *would be forwarded to its destination if the sender has more than* `D` *of cash, after his cash balance is reduced by* `D`*.*

$\mathcal{R}5$.

```
arrived(X, deposit(D),B) :- role(bank)@CS,
      do(deliver).
```

If the reciepient of a `deposit` *message is a bank, the message would be delivered to it without further ado; otherwise it would be blocked (see comment in the text of the paper).*

$\mathcal{R}6$.

```
sent(C,order(Specs,Fee),S) :- cash(Bal)@CS, ( Fee =< Bal),
      do(decr(cash(Bal),Fee)), do(forward).
```

A message `order(_,Fee)` *can be send only by one whose cash balance is larger than* `Fee`*. This balance is decremented by the sending.*

$\mathcal{R}7$.

```
arrived(C,order(Specs,Fee),S) :- do(incr(escrow(Bal),Fee)),
      do(impose_obligation(respond(C,Specs,Fee),60)),
      do(deliver).
```

When an order arrives, its `Fee` *is added to the cash of the reciepient, the order is delivered to him, and an obligation is imposed to respond to the order in 60 seconds (see Rule $\mathcal{R}12$).*

Fig. 2. The Service Law ($\mathcal{SL}$)—Part I

Symmetrically, by Rules $\mathcal{R}4$ and $\mathcal{R}5$ anybody can send any part of his cash balance for deposit in a bank. (It should be pointed out, though, that if a deposit message is sent to a non-bank, the money in it would be lost. Such loss can be easily prevented, at the cost of a slight complication of these rules).

A point of clarification is in order here: this law does not deal with the withdrawal requests sent to the bank, which presumably prompt the bank to send cash—such requests are left unregulated here, so they can be communicated in arbitrary ways, such as via e-mail.

making Service Orders: The sending of service order of the form `order(specs,fee)` is governed by Rule $\mathcal{R}6$, which allows this message to be sent only if the cash balance of the sender is at least as large as `fee`. If the order is sent, then the cash balance of the sender is decremented by `fee`. By Rule $\mathcal{R}7$, the arrival of this order at a server `S` causes the following operations to be carried out: (a) the escrow account of `s` is incremented by `fee`; (b) an obligation `respond(c,specs,fee)` is imposed on `s`, to become due in 60 seconds (used here as an example); and (c) the order is delivered to `s` itself. As we shall see below, the money deposited into escrow will be withdrawn from it when a response to the order in question is given. Note that this money cannot be used by `s` to make his own orders, because, by Rule $\mathcal{R}6$, orders can utilize only money defined as cash.

Responses to Orders: As long as the obligation `respond(c,specs,fee)` is pending, `s` can respond voluntarily to the order he received from `c`, either by accepting or be declining it, as follows:

– To *accept* an `order(specs,fee)` received from `c`, the server `s` sends a message `accept(specs,fee)` to it. This message is governed by Rules $\mathcal{R}8$ and $\mathcal{R}10$. By Rule $\mathcal{R}8$, the sending of an `accept` message causes the following operations to be carried out: (a) `fee` dollars are removed from the escrow and added to the cash value of `s`, thus making this money usable by `s` for making its own orders; (b) the relevant `respond(...)` obligation is repealed, since it is obviously not needed any longer; and (c) the `accept` message is forwarded to `c`.
  By Rule $\mathcal{R}10$, the arrival of a message `accept(specs,fee)` causes the term
  `receipt(accept(specs,fee),Time)`
  to be added to the CS of the receiver `c`, which can serve as a time-stamped record of the promise of the server to carry out the service. Note that the parameter `fee` in this message does *not* cause any transfer of cash.
– To *decline* an `order(specs,fee)` received from `c`, the server `s` sends a message `decline(specs,fee)` to it. This message is governed by Rules $\mathcal{R}9$ and $\mathcal{R}11$. By Rule $\mathcal{R}9$, the sending of an `decline` message causes `fee` dollars to be removed from the escrow, and to be transferred in the message forwarded to `c`; the `respond(...)` obligation is then repealed, as in the previous case, and the declined message is forwarded. By Rule $\mathcal{R}11$, the arrival of a message `decline(specs,fee)` at `c`, causes `fee` dollars to be added to the cash of `c`, thus returning to it the money it send with the request hereby declined.

If server `s` did not respond to `order(specs,fee)` from `c` in time, then obligation `respond(c,specs,fee)` (imposed by the arrival of this order) would come due, triggers the sanction defined by Rule $\mathcal{R}12$.

This sanction is as follows: (a) `fee` dollars are withdrawn from the escrow of `s`; (b) a `decline` message is forwarded to client `c`, with precisely the same effect that a voluntary decline would have. The promise of bounded delay of the response, made by the policy SR in Section 2, is based on fairly standard, even if not entirely reliable,

$\mathcal{R}$8.
```
sent(S,accept(Specs),C)
   :- obligation(respond(C,Specs,Fee),T)@CS,
      do(decr(escrow(Bal),Fee)), do(incr(cash(Bal),Fee)),
      do(repeal_obligation(respond(C,Specs,Fee),_)),
      do(forward).
```
An agent can send an `accept` *message only if it has a matching obligation to* `respond`*; this obligation is automatically repealed by this responce.*

$\mathcal{R}$9.
```
sent(S,decline(Specs,Fee),C)
   :- obligation(respond(C,Specs,Fee),T)@CS,
      do(decr(escrow(Bal),Fee)),
      do(repeal_obligation(respond(C,Specs,Fee),_))),
      do(forward).
```
Similar to the previous rule, except that the `decline` *message carries with it the* `Fee` *found in the obligation to respond, after removing it from the balance.*

$\mathcal{R}$10.
```
arrived(S,accept(Specs,Fee),C) :-
      do(+receipt(accept(Specs,Fee),Time)), do(deliver).
```
Upon its arrival, an `accept` *message is recorded in the CS of agent* `C`*, and delivered to its agent.*

$\mathcal{R}$11.
```
arrived(S,decline(Specs,Fee),C) :-
      do(incr(cash(Bal),Fee)), do(deliver).
```
Get the money and deliver the message

$\mathcal{R}$12.
```
obligationDue(respond(C,Specs,Fee))
   :- do(decr(escrow(Bal),Fee)),
      do(forward(decline(Specs,Fee),C,Self)).
```
The sanction consists of (a) decrementing the balance, to allow for the money to be returned, and (b) forwarding the `decline` *message to the original client, which has the same effect as the* `decline` *message sent voluntarily under Rule $\mathcal{R}$11.*

Fig. 3. The Service Law ($\mathcal{SL}$)—Part II

assumptions of reliable communication, bounded message-transition time, and bounded difference between the rate of clocks.

4.1 On the Relationship between R-trust and F-trust

We conclude this section with an examination of the complementary interrelationship between familiarity-based trust (F-trust) and regularity-based trust (R-trust), as it is reflected in the case study of this paper. Basically, we will show that the two types of trust support and enhance each other, albeit in different ways.

We start by showing that R-trust, as implemented via LGI laws, is based on some F-trust. Indeed the basis for trusting law $\mathcal{SL}$—that is, for trusting that interaction via $\mathcal{SL}$ messages satisfies policies CH and SR formalized by this law—is the correctness of the controllers that mediate this interaction. And the reason to trust the controllers in this case is the F-trust we have in the CA (identified here by its public key `pk1`), which certifies these controllers. And the reason that this CA certifies controllers is, presumably, that it has an F-trust in their correctness.

Moreover, we note that the trust in law $\mathcal{SL}$ does not, by itself, have much practical meaning, without an assurance that the money one is offered as a fee for service originates from a trustworthy bank, and that this money can be deposited in one's bank account. Under law $\mathcal{SL}$ such assurances depends on the trustworthiness of the CA `pk2`, used here to authenticate banks. So, the R-trust established by law $\mathcal{SL}$ depends on the F-trust in CA `pk2`, and on the F-trust that this CA has in the banks it certifies.

On the other hand, the effectiveness of F-trust could be enhanced by an appropriate R-trust. In our case, the F-trust one has in the banks certifies by the CA `pk2` allows members of the $\mathcal{SL}$-community to trust funds obtained from any other members, without having to worry about the origin of these funds. That is, the fee one gets in a service-order under this law, taken from the cash balance of the client, may be the result of direct or indirect contributions of several banks. But one does not need to know the identity of these banks for trusting the validity of the currency. This is the consequence of one's R-trust in the validity of the cash handling policy CH generated by law $\mathcal{SL}$. This is similar to the trust we have in cash in the physical world, which is due to the trust we have that dollar bills are not forged, regardless of the bank that issued them (this is regularity R2, of the introduction.)

5 Conclusion

The type of trust identified in this paper as *regularity-based trust*, is a critical factor in the comprehensibility and manageability of the physical world—in both the natural and artificial aspects of it. But the role of such trust in cyberspace has been limited so far, because of difficulties in establishing sufficiently wide range of useful regularities in this context.

We have demonstrated in this paper that the use of middleware such as LGI can help enhance the role of regularity-based trust in cybersapce, making it a significant factor in activities such as e-commerce. We also discussed the relationship between this kind of trust, and what we have called familiarity-based trust, which is currently more commonly employed in cyberspace.

References

1. J.P. Anderson. Computer security technology planning study. Technical Report TR-73-51, Air Force Electronic System Division., 1972.
2. X. Ao, N. Minsky, and V. Ungureanu. Formal treatment of certificate revocation under communal access control. In *Proc. of the 2001 IEEE Symposium on Security and Privacy, May 2001, Oakland California*, May 2001. (available from `http://www.cs.rutgers.edu/~minsky/pubs.html`).
3. J. Barkley, K. Beznosov, and J. Uppal. Supporting relationships in access control using role based access control. In *Proceedings of the Fourth ACM Workshop on Role-Based Access Control*, pages 55–65, October 1999.

4. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems security. *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, 1603, 1999.
5. D. Brewer and M. Nash. The Chinese Wall security policy. In *Proceedings of the IEEE Symposium in Security and Privacy*. IEEE Computer Society, 1989.
6. M. Brown. Agents with changing and conflicting commitments: a preliminary study. In *Proc. of Fourth International Conference on Deontic Logic in Computer Science (DEON'98)*, January 1998.
7. D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the IEEE Symposium in Security and Privacy*, pages 184–194. IEEE Computer Society, 1987.
8. C. Ellison. The nature of a usable pki. *Computer Networks*, (31):823–830, November 1999.
9. C English, P. Nixon, and S. terzis. Dynamic trust models for ubiquitous computing environemnt. In *Proceedings of the Ubicom 2002 Security Workshop*, 2002.
10. S. Foley. The specification and implementation of 'commercial' security requirements including dynamic segregation of duties. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, April 1997.
11. 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.
12. R.J. Hayton, J.M. Bacon, and K. Moody. Access control in an open distributed enviroment. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, 1998.
13. A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, 2000.
14. G. Karjoth. The authorization service of tivoli policy director. In *Proc. of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*, December 2001. (to appear).
15. P.F. Linington. Options for expressing ODP enterprise communities and their policies by using UML. In *Proceedings of the Third Internantional Enterprise Distributed Object Computing (EDOC99) Conference*. IEEE, September 1999.
16. J. J. Ch. Meyer, R. J. Wieringa, and Dignum F.P.M. The role of deontic logic in the specification of information systems. In J. Chomicki and G. Saake, editors, *Logic for Databases and Information Systems*. Kluwer, 1998.
17. N.H. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, February 1991.
18. 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. (available from `http://www.cs.rutgers.edu/~minsky/pubs.html`).
19. R. Rivest and B. Lampson. SDSI-a simple distributed security infrastructure. Technical report, MIT, 1996. http://theory.lcs.mit.edu/~rivest/sdsi.ps.
20. B. Schneier. *Applied Cryptography*. John Wiley and Sons, 1996.
21. M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 1994.
22. Mark Stefik. *The Internet Edge*. MIT Press, 1999.
23. 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.