Establishing Enterprise Communities*

Constantin Serban, Xuhui Ao, Naftaly Minsky
{serban,ao,minsky}@cs.rutgers.edu

Department of Computer Science
Rutgers University, Piscataway, NJ 08854

03/30/01

Abstract

One of the most important challenges facing the builders of enterprise software is the reliable implementation of the policies that are supposed to govern the various *communities* operating within an enterprise. Such policies are widely considered fundamental to enterprise modeling, and their specification were the subject of several recent investigations. But specification of the policy that is to govern a given community is only the first step towards its implementation—the second, and more critical step is to *ensure* that all members of the community actually conform to the specified policy.

The conventional approach to the implementation of a policy is to build it into all members of the community subject to it. But if the community in question is large and heterogeneous, and if its members are dispersed throughout a distributed enterprise, then such "manual" implementation of its policy would be too laborious and error-prone to be practical. Moreover, a policy implemented in this manual manner would be very unstable with respect to the evolution of the system, because it can be violated by a change in the code of any member of community subject to it.

It is our thesis that the only reliable way for ensuring that an heterogeneous distributed community of software modules and people conforms to a given policy is for this policy to be strictly *enforced*. A mechanism for establishing enterprise communities by formally specifying their policies, and by having these policies enforced is the subject of this paper.

1 Introduction

One of the most important challenges facing the builders of enterprise software is the reliable implementation of the policies that are supposed to govern the various *communities*[1] operating within an enterprise. Such policies are widely considered fundamental to enterprise modeling, and their specification were the subject of several recent investigations [3, 12, 7, 4, 5]. But specification of the policy that is to govern a given community is only the first step towards its implementation—the second, and more critical step is to *ensure* that all members of the community actually conform to the specified policy.

The conventional approach to the implementation of a policy is to build it into all members of the community subject to it.[2] But if the community in question is large and heterogeneous, and if

*Work supported in part by an "Excellence Award" by the NJ Commission on Science and Technology, and by NSF grants No. CCR-9710575 and No. CCR-98-03698

[1]We use here the term "community" in its sense in the *enterprise language* currently under development within ISO [6]. It is a collection of agents (or, objects), formed to meet certain objectives within an enterprise, and which operate under a distinct policy.

[2]This, for community members that are program modules, which can be constructed at will. The conventional approach towards human members of a community is to inform them of the policy, and to train them for observing it.

its members are dispersed throughout a distributed enterprise, then such "manual" implementation of its policy would be too laborious and error-prone to be practical. Moreover, a policy implemented in this manual manner would be very unstable with respect to the evolution of the system, because it can be violated by a change in the code of any member of community subject to it.

It is our thesis that the only reliable way for ensuring that an heterogeneous distributed community of software modules and people conforms to a given policy is for this policy to be *enforced*. Such an enforcement is, indeed, possible under the concept of law-governed interaction (LGI) [8, 10]. LGI is a message-exchange mechanism that enables an heterogeneous collection of distributed agents to engage in a mode of interaction *governed* by an explicitly specified and scalable enforced policy, called the *law* of this collection. A collection of agents thus interacting via LGI under a given law $\mathcal{L}$, is called an $\mathcal{L}$-community.

But there is a difficulty with the current concept of a community under LGI, which is addressed in this paper. The difficulty is that due to the strictly decentralized deployment technique used by LGI [1], an $\mathcal{L}$-community has an *indefinite membership*. That is to say, the membership of an $\mathcal{L}$-community is implicitly defined as the set of all agents that operate under law $\mathcal{L}$; and there is no way to know what is the total membership of such a community, or to restrict this membership in any way. This is considered an advantage for open communities operating over the internet[1], and is often appropriate for enterprise communities as well, as has been demonstrated in [9, 2]. But more often than not, it is necessary for the membership of an enterprise community to be *known*, and to be *controllable*. In this paper we introduce the concept of an $\mathcal{L}$-group under LGI, which is a subset of an $\mathcal{L}$-community that has a definite membership, whose changes can be controlled, and which provides other useful services as well.

The rest of this paper is organized as follows: We start, in Section 2, with a motivation of our concept of an $\mathcal{L}$-group, by considering a policy governing *distributed committees* operating within an enterprise. In Section 3 we provide an overview of the concept of law-governed interaction, which is the basis for this paper. In Sections 4 we introduce the concept of $\mathcal{L}$-group under LGI, and in Sections 5 we demonstrate this concept by formalizing our example policy of committees into a law. We make some concluding remarks in Section 6.

2 Distributed Committees—a Motivating Example

To illustrate the nature of communities in modern computerized enterprises, and to demonstrate the need to enforce their underlying policies, we will consider the case of distributed committees. A committee, in general, is a group of people whose function is to deliberate and make communal decisions about certain issues. Traditionally, such a group meets periodically in a conference room, to deliberate and to vote on their resolutions, observing semi informal rules of engagement, often modeled roughly after the *Robert's Rules of Order*.

But the traditional *modus operandi* of committees is changing in many cases, due to the increasing use of internet communication within enterprises, particularly when the members of a committee are geographically distributed. Instead of gathering in a conference room for their deliberations, the members of a committee might exchange views, and even vote on their resolutions, via internet messages.

Such distribution of committee work poses serious risks to its integrity. One such risk is that messages ostensibly exchanged between committee members would be forged. Another, even more serious risk, is that the voting process would be undermined, due to forged votes, incorrect counting of ballots, and the public disclosure of individual votes, when the committee is supposed to engage in a secret ballot. To minimize these risks it is necessary to carefully formulate an appropriate policy regarding the official interactions between committee members, and it is vital for this policy to be enforced, to ensure conformance with it.

We now introduce an example policy for distributed committees, under the following assumption about the enterprise at hand: The enterprise employs a *certification authority* (CA) called "*admin*" that provides digital certificates that authenticate the identity of individual employees, and the roles they play within the enterprise. Such a certificate might, for example, assert that an employee identified by a certain public-key (see [11])) is called "john Smith", and that he serves as the chair of certain committee. Our example policy for distributed committees, which we call *DC*, is specified, informally, as follows:

1. The membership of each committee is pre-specified, and can be authenticated via a certificate issued by the CA *admin*, with one member designated as its chair.
2. In every official message-exchange between committee members, each member should be provided with the identity of his interlocutor, as authenticated by the *admin* CA.
3. Any member can offer a resolution, which can be put to a vote once it is seconded by another member.
4. The voting is performed via a secret ballot, which must satisfy the following conditions:
   (a) The duration of the ballot is as specified when the resolution is initially offered.
   (b) Members cannot see each other's vote, but the copy of each vote is sent to a specified audit trail server.
   (c) The result of the ballot, and whether it is binding, is broadcasted to all member of the committee, once the balloting is over. (The result of the vote is considered binding only if the number of submitted votes is no smaller than the predefined *quorum* of this committee).
5. The chair of a committee can suspend any of its members from voting and from offering new resolutions, for a specified period of time.

In Section 5 we formalize this policy, as an enforceable law under LGI.

3 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 here no assumptions about the structure and behavior of the agents[3] that are members of a given community $\mathcal{C}_{\mathcal{L}}$, which might be software processes, written in an arbitrary languages, 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 (via TCP/IP, say) 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 dynamic changes in their state. These control-states, whose semantics for a given community is defined by its law, could represent such things as the role of an agent in this community, privileges and tokens it carries, and dynamic identification of the state of computations in which the agent is involved.

We now elaborate on several aspects of LGI, focusing on (a) its concept of law, (b) how agents can join an $\mathcal{L}$-community (c) its treatment of digital certificates and (d) interoperability between communities. Due to lack of space, we do not discuss here several important aspects of LGI, including

[3]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.

the treatment of *exceptions*, the expressive power of LGI, its implementation, and its efficiency. For these issues, and for a more complete presentation of the rest of LGI, the reader is referred to [10, 1, 13].

3.1 The Concept of Law

Generally speaking, the law of a community $\mathcal{C}$ is defined over a certain types of events occuring 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; the *coming due of an obligation* previously imposed on a given object; and the submission of a digital certificate. 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 occured (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. The locality of law enforcement, however, critically enables LGI to scale with community size.

On the Structure and Formulation of 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 location of the event from which the ruling was derived (called 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.)

Concretely, the law is defined by means of a Prolog-like program[4] `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 have special roles to play 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 each 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. A sample of primitive operations is presented in Figure 1.

[4]Note, however, that Prolog is incidental to this model, and can , in principle, be replaced by a different, possibly weaker, language; a restricted version of Prolog is being used here.

Operations on the control-state	
`t@CS`	returns true if term `t` is present in the control state, and fails otherwise
`+t`	adds term `t` to the control state;
`-t`	removes term `t` from the control state;
`t1`←`t2`	replaces term `t1` with term `t2`;
`incr(t(v),d)`	increments the value of the parameter `v` of term `t` with quantity `d`
`dcr(t(v),d)`	decrements the value of the parameter `v` of term `t` with quantity `d`

Operations on messages	
forward(x,m,y)	sends message `m` from `x` to `y`; triggers at `y` an `arrived(x,m,y)` event
deliver(x,m,y)	delivers the message `m` from `x` to agent `y`

Figure 1: Some primitive operations in LGI

The Concept of Enforced Obligation: Informally speaking, an obligation under LGI is a kind of *motive force*. Once an obligation is imposed on an agent—generally, 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*, and 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.

Specifically, an obligation can be imposed on a given agent `x` at time `t0` by the execution at `x` of a primitive operation

```
imposeObligation(oType,dt),
```

where `dt` is the time period, after which the obligation is to come due, and `oType`—the *obligation type*—is a term that identifies this obligation (not necessarily in a unique way). The main effect of this operation is that unless the specified obligation is *repealed* before its due time $t=t_0+dt$, the *regulated event*

```
obligationDue(oType)
```

would occur at agent `x` at time `t`. The occurrence of this event would cause the controller to carry out the ruling of the law for this event; this ruling is, thus, the *sanction* for this obligation. Note that a pending obligation incurred by agent `x` can be *repealed* before its due time by means of the primitive operation

```
repealObligation(oType)
```

carried out at `x`, as part of a ruling of some event. (This operation actually repeals *all* pending obligations of type `oType`).

The Basis for Trust between Members of A Community: 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 2) 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. Although several agents can share a single controller, if such sharing is desired. (The efficiency of this mechanism, and its scalability, are discussed in [10].) 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 can maintain a set of active controllers.

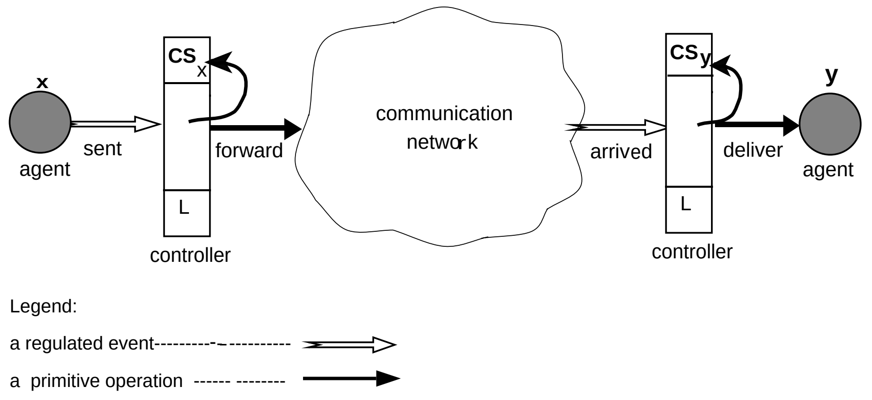

Figure 2: Enforcement of the law

3.2 Engaging in an $\mathcal{L}$-Community

For an agent x to be able to exchange $\mathcal{L}$-messages with other members of an $\mathcal{L}$-community, it must: (a) find an LGI controller, and (b) notify this controller that it wants to use it, under law $\mathcal{L}$. We will discuss these two steps below.

Locating an LGI Controller: As we discussed, LGI have implemented a *controller-service*, which maintains a set of active controllers and provides the address of the available controllers to any agent that wished to engage in LGI. The controllers can be dispersed geographically over the Internet or distributed enterprise, so one agent can select a controller reasonably close to it.

Adopting a Law: Upon selecting a controller T, x would send T the message

```
adopt(law,id),
```

where `law` is the law that it wants to adopt, and `id` is the name that it wants to be known by within this community. When controller T receives the `adopt` message, it checks the supplied law for syntactic validity, and the chosen id for uniqueness among the identifiers of all current agents handled by T. If these two conditions are satisfied, and if T is not already loaded to capacity, it will set up a starting control-state for agent x, as specified in the preamble of the law adopted by x, allowing x to start operating under this law[5].

3.3 The Treatment of Certificates under LGI

Under LGI, *all agents are made equal* at the time they join an $\mathcal{L}$-community. This is because the control-state of all new members is identical—and control-states provide the only means for a law to make distinctions between agents. We now explain how an agent can acquire extra privileges, thus becoming *more equal than others*, by submitting appropriate certificates.

The submission by an agent x, operating under law $\mathcal{L}$, of a certificate `cert` to its controller, has the following effect: An attempt is made to confirm that `Cert` is a valid certificate, duly signed by an authority that is acceptable to law $\mathcal{L}$, i.e., an authority that is represented by one of the `Authority-clauses` in the preamble to the law (See Figure 5 for an example). If this attempt is successful[6], then a *certified-event* is triggered. This event, which is one of the *regulated-events* under LGI, has as its argument the following representation of the submitted certificate:

```
[issuer(I), subject(S), attributes(A)].
```

[5]If any one of these conditions is not satisfied, then `x` would receive an appropriate diagnostic, and will be able to try again.

[6]If the the certificate is found invalid then an *exception-event* is triggered.

Here I and S are internal representations of the public-keys of the CA that issued this certificate, and of its subject, respectively; and A is what is being certified about the subject. Structurally, A is a list of `attribute(value)` terms. For example, the attributes of a certificate might be the list `[name(John Smith), role(chair,c1)]`, asserting that the name of the subject in question is John Smith and his role in one committee named `c1` is chair. What happens when the `certified` event is triggered depends, of course, on the law.

3.4 Interoperability Between Communities

LGI also support the interoperability between different communities. By "interoperability" we mean, the ability of an agent x operating under law $\mathcal{L}_x$ to exchange messages with agent y operating under different law $\mathcal{L}_y$, such that the following properties are satisfied: (a)`consensus:` An exchange between a pair of laws is possible only if it is authorized by both. (b)`autonomy:` The effect that an exchange initiated by x operating under law $\mathcal{L}_x$ may have on the structure and behavior of y operating under law $\mathcal{L}_y$, is subject to law $\mathcal{L}_y$. (c)`transparency:` Interoperating parties need not to be aware of the details of each other law.

To support such an interoperability between communities, LGI has such primitive operation–`export` and event–`imported`, as follows:

- `export(x,m,[y,Ly])`: invoked by agent x under law $\mathcal{L}_x$, initiates an exchange between x and agent y operating under law $\mathcal{L}_y$. When the message carrying this exchange arrives at y it would invokes at it an `imported` event under $\mathcal{L}_y$.
- `imported([x,Lx],m,y)`: occurs when a message m exported by x under law $\mathcal{L}_x$ arrives at agent y operating under law $\mathcal{L}_y$.

With which law one law can interoperate is defined by the `Portal` clause in the law preamble. For example, the `Portal-clause` in Figure 5 allows the agents under committee-law to exchange messages with those agents under secretary-law.

4 The Concept of an $\mathcal{L}$-group

As has already been pointed out, an $\mathcal{L}$-community is defined as the set of all agents that *happen* to be operating under law $\mathcal{L}$. Such a community is never formally established and there is no formal admission into it. Anybody can become a member of this community, simply by adopting the law $\mathcal{L}$ when using LGI. One can draw an analogy between such a community and, say, the community of French speaking people. Just as in the latter case, it is not possible to determine the total membership of a given $\mathcal{L}$-community under LGI, or to regulate this membership in any way.

Such an open concept of a community is very suitable for many internet applications, and it is often appropriate in the enterprise context as well. For example, to regulate its interactions with its customers, a given enterprise might define a customer-law, to be adopted by anybody who wishes to interact with this enterprise. The resulting customer community is clearly indefinite and open. But in many situations, exemplified by our committee example, one needs a more definite, and controllable membership, which is provided by the concept of $\mathcal{L}$-group introduced below.

Broadly speaking, an $\mathcal{L}$-group is a subset of the members of the $\mathcal{L}$-community that has the following properties:

1. The membership of an $\mathcal{L}$-group is known, at every moment of time, and it can be regulated.
2. There can be any numbers of arbitrarily overlapping $\mathcal{L}$-groups in a given $\mathcal{L}$-community. (That is, a given member of an $\mathcal{L}$-community can belong to any number of different $\mathcal{L}$-groups, or to none.)

We will introduce here one implementation of this concept of a group, under LGI.

Every $\mathcal{L}$-group G is anchored on an agent $S(G)$ (or, simply S) called the *secretary* of G. Each such secretary serves a single group, providing it with various generic group-services, which are

Roles	
`role(chair)`	the role records for a chairman join
`cardinality(1)`	only one chairman can join the secretary
`CS([active(S),quorum(N)])`	specific terms for the chairman CS
`role(member)`	the role records for a member join
`cardinality(-1)`	the number of member joins is not specified
`CS([active(S),quorum(N)])`	general terms for the member CS

Figure 3: The secretary configuration file

independent of the law $\mathcal{L}$ under which this group operates. When a particular secretary $S(G)$ is activated, it is primed with the law under which the members of G operate, and with a *configuration file*, which, as we shall see below, defines initial structure of the members of G. Once activated, S serves as a gateway through which members of the $\mathcal{L}$-community join group G, and it provides these members with the following services:

- *Membership information*, of the following kinds: (a) the secretary provides the entire membership list, upon request from any of its members; and (b) it allows individual members to publish information, such as their control-state, making it available to all other group members.
- *Membership control*, via limits imposed on the total membership, and via removal of existing members.
- *Broadcasting messages* to all group members.
- *Persistent storage* of the control-states of members.

For a detailed discussion of these services, the reader is referred to "www.cs.rutgers.edu/moses/"; here we will elaborate only on certain aspects of this facility, and then illustrate it via groups operating as committees.

The secretaries serving various groups all operate under their own secretary-law that allows them to import commands of the form `groupCommand(cm)` from agents operating under arbitrary laws, and to export to them replies of the form `reply(rep)`. A secretary serving group G provides to all its client a generic API, which includes such operations as `joinAs` (allowing an agent to join group G, and to occupy a specified role in it), `who` (asking for the membership list of the group), and `broadcast(m)`. Yet, the groups served by such generic secretaries may differ profoundly from each other, due to two factors: (a) the configuration files with which the different group-secretaries have been primed, and (b) the laws under which the members of the group operate.

The configuration file used to prime a secretary, for service of a specific group, has the form exemplified by the configuration file we use for committees displayed in Figure 3. This files contains two "role-entries", the first for the role of committee chair, and the other for a regular members. Beside the name of the role, a role-entry specifies the following: (a) the *cardinality* of the role, which is the maximal number of members allowed to hold this role at any given moment (it is 1 for the chair in this case, and unlimited (which is what "-1" means here) for a regular member); and (b) a collection of terms to be added to the control state of anybody joining the group in that role. In general, a configuration file, which may have any number of such role-entries, can determine some aspects of the composition of the group.

Although, as we have said, all secretaries provide a generic set of commands, the manner in which these commands can be used by the members of a given $\mathcal{L}$-community is determined by law $\mathcal{L}$. For example, under the committee-law to be introduced in the following section, for an agent to join a given committee he must present an appropriate certificate, which authenticates him as either the chair of this committee or a regular member. As another example, secretaries provide a command `remove` that allow one agent to remove another from their group. But the law under

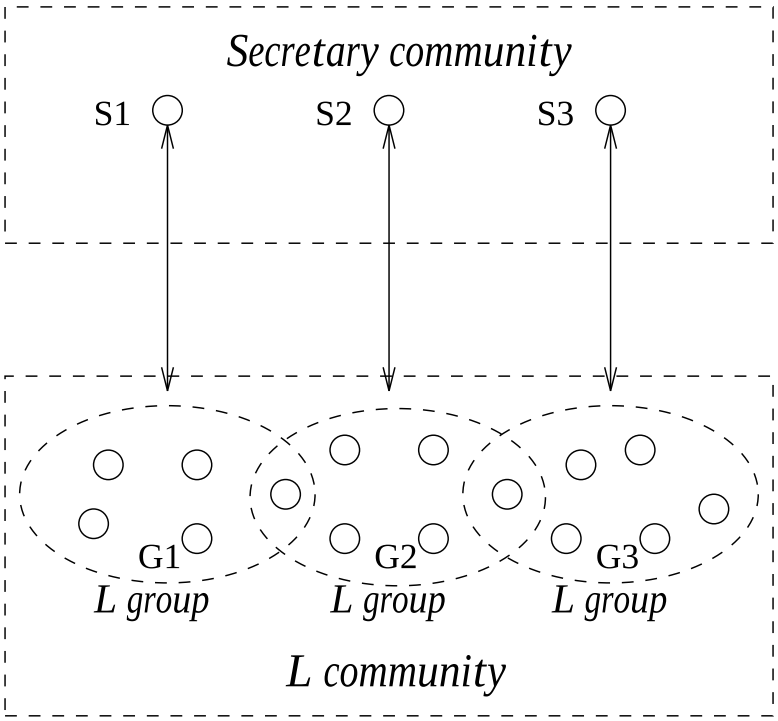

Figure 4: $\mathcal{L}$-community, $\mathcal{S}$-community and $\mathcal{L}$-group

which a given group operates may regulate the use of this powerful command. For example, the committee-law does not allow this command to be issued at all. (The interplay between $\mathcal{L}$-groups and their secretaries is illustrated in Figure 4).

5 The Committee Law

In this section we present the formal expression of the policy introduced in Section 2 via a law under LGI. This law, displayed in Figures 5 and 6 , is composed of a preamble and a set of rules, each followed by a comment (in italic), which, together with the explanation bellow, should be understandable even for a reader not well versed in our Prolog-based language of laws.

The preamble of the law provides the following information. The first line specifies the name of the current law, *committee* in this case. The second line declares a portal that enables our law to exchange messages with another law via *export* and *import* primitives, as discussed in Subsection 3.4. In this case, the portal for the secretary-law is given (the secretary-law itself is not described in this paper). The next line specifies the public key of the Certification Authority that certifies agents under this law. The fourth line defines "auditor" as an alias of the address of the server that maintains an audit trail for operations of committees. The last line of the preamble specifies the initial Control State of every agent under this law—it is empty in this case. We now discuss in some details how this law deals with various aspects of the structure and behavior of committees, starting with the manner that agents operating under this law certify their names and roles.

Certification: As mentioned in Section 2, the enterprise employs a *certification authority* (CA) called "*admin*" that provides digital certificates that authenticate the identity of individual employees, and the roles they play within the enterprise. The attribute of a committee membership certificate issued by *admin* has the following form:

```
[name(N), role(R,S)].
```

Preamble:

```
Law(committee)
Portal(secretary,hash_of_secretary_law)
Authority(admin,public_key_of_CA)
Alias(auditor,auditor@enterprise.com)
InitialCS([])
```

$\mathcal{R}1$.
```
certified([issuer(admin),subject(self),attributes(A)]) :-
        name(N)@A,role(R,S)@A, do(+name(N)),do(+role(R,S)).
```

when receiving a self-certificate, match the role, the name and the committee name in the certificate and add the terms `name(name)` *and* `role(role,cname)` *into CS*

$\mathcal{R}2$.
```
sent(X,groupCommand(joinAs(N,R)),S) :-
        role(R,S)@CS,name(N)@CS,
        do(export(X,groupCommand(joinAs(N,R)),[S,secretary])).
```

the agent sends a joinAs message to the secretary. The arguments are the proposed name and the targeted role

$\mathcal{R}3$.
```
imported([S,secretary],reply(joinAs,accepted(N,deltaCS)),X) :-
        do(deliver(S,reply(joinAs,accepted(N,deltaCS)),X)), do(+addCS(deltaCS)).
```

if it is an acceptance message, deliver the message and add the role corresponding terms to the CS. The terms are specified in the secretary role file

$\mathcal{R}4$.
```
sent(X,cmessage(M,S,Nm),Y) :-
        role(R,S)@CS,name(N)@CS, do(forward(X,cmessage(M,S,N,Nm),Y)).
```

append the name of the sender to each official message within the committee

$\mathcal{R}5$.
```
arrived(X,cmessage(M,S,N,Nm),Y) :-
        role(R,S)@CS,name(Nm)@CS, do(deliver).
```

deliver each committee official message if the receiver name Nm matches the target name Nm in the message

$\mathcal{R}6$.
```
sent(X,suspend(S,T),Y) :-
        role(chair,S)@CS, do(forward(X,suspend(S,T,N),Y)).
```

only the chair may send a suspension message to another member of the same committee

$\mathcal{R}7$.
```
arrived(X,suspend(S,T,N),Y) :-
        role(member,S)@CS,name(N)@CS, do(-active(S)),
        do(imposeObligation(enable(S),T)). do(deliver).
```

when the suspension message arrives, remove the term `active(S)` *from CS and impose an obligation to come due when the suspension expires*

$\mathcal{R}8$.
```
obligationDue(enable(S)) :-
        do(+active(S)).
```

add the term `active(S)` *back to CS when the suspension obligation comes due*

Figure 5: Committee Law

This certificate establishes that the agent identified by it has the name N and it has the role R in the committee S. (Since each committee has exactly one secretary we use, for simplicity, the id of the secretary as the name of the committee.) In Rule $\mathcal{R}1$, when an agent presents a certificate with the attributes as above, the terms *name(N)* and *role(R,S)* are added in the control-state. The role can be either *chair* or *member*. Note that an agent may end up having more than one role term since he may be member in several committees.

$\mathcal{R}$9.
```
sent(X,groupCommand(broadcast(resolution(I,T))),S) :-
        role(R,S)@CS,name(N)@CS,active(S)@CS, do(+resolution(I,T,S)),
        do(export(X,groupCommand(broadcast(resolution(I,T,S,N))),[S,secretary])).
```
only an active member may broadcast a resolution under a committee name

$\mathcal{R}$10.
```
imported([S,secretary],broadcast(resolution(I,T,S,N),from(Y)),X)
     :- do(deliver(S,broadcast(resolution(I,T,S,N),from(Y)),X)).
```
every resolution is delivered

$\mathcal{R}$11.
```
sent(X,second(I,S),Y) :-
        role(R,S)@CS,name(N)@CS,active(S)@CS, do(forward(X,second(I,S,N),Y)).
```
only an active member my second a resolution

$\mathcal{R}$12.
```
arrived(X,second(I,S,N),Y) :-
        role(R,S)@CS,name(Nm)@CS,active(S)@CS, resolution(I,T,S)@CS,
        do(+result(0)),do(+count(0)),do(+vote(I,S,Nm)),
        do(export(Y,groupCommand(broadcast(startnow(I,T,S,Nm))),[S,secretary])),
        do(imposeObligation(deadline(I,S),T)), do(deliver).
```
when the second message arrives at the host, add the voting terms into CS, broadcast the official start of the vote and set the ballot deadline obligation

$\mathcal{R}$13.
```
imported([S,secretary],broadcast(startnow(I,T,S,Nm),from(Y)),X) :-
        role(R,S)@CS,active(S)@CS, do(+vote(I,S,Nm)),
        do(deliver(S,broadcast(startnow(I,T,S,Nm),from(Y)),X)).
```
receive one vote token when notified of the start of the ballot

$\mathcal{R}$14.
```
sent(X,Z,Y) :-
        role(R,S)@CS,name(N)@CS,active(S)@CS, Z=vote(I,S,Nm,Op),Op@[-1,0,1],
        vote(I,S,Nm)@CS,do(-vote(I,S,Nm)), do(forward(X,vote(I,S,Nm,N,Op),Y)),
        do(deliver(X,Z,auditor)).
```
allow a member to vote only once, the vote is forwarded to the host of the vote and delivered to audit trail server

$\mathcal{R}$15.
```
arrived(X,vote(I,S,Nm,N,Op),Y) :-
        role(R,S)@CS,name(Nm)@CS, result(Res)@CS,count(Cnt)@CS, NRes is Res + Op,
        do(result(Res) <- result(NRes)), do(incr(count(Cnt),1)).
```
when the vote arrives at the host, compute the partial results and increment the counter of total votes

$\mathcal{R}$16.
```
obligationDue(deadline(I,S)) :-
        role(R,S)@CS,name(N)@CS, result(Res)@CS,count(Cnt)@CS,quorum(Cv)@CS,
        do(-result(Res)),do(-count(Cnt)),do(-resolution(I,T,S)),
        (Cv > Cnt -> Ballot is notbinding ; Ballot is binding),
        do(export(Self,groupCommand(broadcast(ballot(I,S,N,Res,Ballot))),[S,
        secretary])), do(deliver(Self,ballot(I,S,N,Res,Ballot),Self)).
```
when the deadline occurs, decide the validity of the ballot based on the quorum, broadcast and deliver the results of the ballot and clean up CS

$\mathcal{R}$17.
```
imported([S,secretary],broadcast(C,from(Y)),X) :-
        C = ballot(I,S,N,Res,Ballot), do(deliver(S,broadcast(C,from(Y)),X)).
```
deliver the results of the ballot to each member of the committee

Figure 6: Committee Law - continuation

Joining a Committee: By Rule $\mathcal{R}2$, an agent x that has the terms *role(R,S)* and *name(N)* in its control-state can send a group-command *joinAs(N,R)* to secretary S. Now, each secretary is built to approve such request, admitting the requester in the group, if the following conditions are satisfied: (a) the current membership of the committee does not already include anybody with name N; and (b) if the cardinality of the requested role, as specified in the configuration file, is not exceeded. Note that condition (a) above prevents the same agent from having duplicate membership in the same committee—which is a very important assurance.

If x is accepted in the committee, the secretary will send it an appropriate reply, which includes the set of CS terms mandated by the configuration file. By Rule $\mathcal{R}3$, when this reply arrives at x, this set of terms called `deltaCS` are appended to the CS of x. In this case, the `deltaCS` consists of the terms *active(S),quorum(N)*, and would have the following effects. The term *active(S)* enables an agent to participate actively in voting, while the term *quorum(N)* specifies the minimum number of votes required for a vote to be declared as binding (condition 4(c) of the policy).

Free exchanges between committee members: Rules $\mathcal{R}4$ and $\mathcal{R}5$ regulate the official, but arbitrary, messages exchanged among the members of the committee. By Rule $\mathcal{R}4$, whenever an agent sends a committee message - *cmessage(M,S,Nm)*, the name of the sender is appended to the message. M represents the informal message to be sent, S is the id of secretary (and of the committee) under which the message should be sent, and *Nm* is the name of the agent to receive the message. The authenticated names of the sender and the receiver travel along with the message, thus providing the peer of the communication with the real identity, as stated in the condition 2 of the policy.

Suspension of voting rights: By Rule $\mathcal{R}6$, a chair can send a suspension message to any member z of his committee, suspending z from voting, or from offering new resolution, for a specified period of time. Technically, this suspension is carried out when the suspension message arrives at z, as follows: First, by Rule $\mathcal{R}7$, the term *active(S)*, which is required for all voting activities, is removed from the control-state of z; second an obligation is imposed on z to restore this term, making z active again, after the suspension period is over (see Rule $\mathcal{R}8$.)

The voting process: Figure 6 presents the rules that deal with the initiation of a vote on a specific resolution and the process of voting on this resolution. Rules $\mathcal{R}9$ and $\mathcal{R}10$ enable an agent to offer a resolution, as required by condition 3 of the policy. The member that initiates the offer is subsequently called the *host* of this resolution. The secretary broadcasts the offer to all committee members. Rule $\mathcal{R}9$ verifies whether the sender belongs to the same committee and adds the term *resolution(I,T,S)* in its Control State. Here, I represents the issue to be voted, T is the voting period once the ballot has started (policy point 4(a)), and S is the secretary id that identifies the committee uniquely. The terms I and S mentioned before along with the name of the host of the offer identify uniquely each resolution and they are used for voting on a resolution.

As discussed in Section 2, in order to start a ballot, another member has to second the resolution. By Rules $\mathcal{R}11$ and $\mathcal{R}12$ an active member may second any resolution by sending a *second(I,S)* message to the host of that resolution. When the second message arrives at the host ($\mathcal{R}12$), a number of steps are followed in order to start the ballot officially. First, the law verifies that the second message has arrived at the right place, the host of the same resolution. If this is the case, the terms *result(0),count(0)* added in the Control State, allow the host of the resolution to compute the results of the ballot at a later moment of time. The host also receives a vote token that enables it to express its vote. The next step is to broadcast the official start of the ballot to all the members of the committee. As a last step, the host imposes an obligation to come due at the end of the ballot period (Rule $\mathcal{R}16$).

By Rule $\mathcal{R}13$, when the message from secretary initiating the official start of the ballot arrives, each member gets a vote token that allows it to vote exactly once, as we shall see.

By Rule $\mathcal{R}14$, anybody who has a vote token can send its vote to the issuer host. After sending the vote, the vote token is taken away, and the message is forwarded to the host. The vote has the form *vote(I,S,Nm,N,Op)* where *I*, *S*, *N* identifies the resolution to be voted, *Nm* reveals the name of the sender of the vote and *Op* is the vote itself. The vote is an integer with the value 1,-1 or 0 with the straightforward meaning of positive, negative or neutral vote.

The vote is only forwarded to the host of the offer (point4(b)) and delivered to the audit trial server that records an official copy of the committee activity.

By Rule $\mathcal{R}15$, when the vote arrives at the host of the resolution, it is counted but not delivered to the host itself.

Finally, Rule $\mathcal{R}16$ deals with the timeout of the ballot. The timeout is triggered after a time period T that has been set by the host of the resolution. The rule first checks whether the ballot is binding or not based on the *quorum(Cv)* term, as discussed in the configuration file (condition 4(c)of the policy) . A message containing the complete result of the ballot is broadcasted through the secretary to all the members of the committee, and it is delivered to them by Rule $\mathcal{R}17$ (the results are also delivered to the host itself).

6 Conclusions

This paper converts the modeling concept of enterprise community, into a system component. That is, under LGI, the actual community members, wheather they are software components or people, must conform to law of that community, and they can trust their interlocutors to be similarly governed by this law. What has not been addressed in this paper is the ability to form hierarchies of laws under LGI, to support the community hierarchies required for the modeling of enterprises according to [7]. A paper about such hierarchies is forthcoming.

It should be pointed out, however, that the concept of law under LGI has a fundamental limitation: It can regulate only interaction between agents, having no sway over the structure and internal behavior of individual agents. This means that various aspects normally included on the policy of a community, like its goals, cannot be directly supported under LGI.

References

[1] X. Ao, N. Minsky, T. Nguyen, and V. Ungureanu. Law-governed communities over the internet. In *Proc. of Fourth International Conference on Coordination Models and Languages; Limassol, Cyprus; LNCS 1906*, pages 133–147, September 2000.

[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 (to be published)*, May 2001.

[3] X. Blanc, M.P. Geravis, and R. Le-Delliou. Using the UML language to express the ODP enterprise concepts. In *Proceedings of the Third Internantional Enterprise Distributed Object Computing (EDOC99) Conference*, pages 50–59. IEEE, September 1999.

[4] J. Cole, Derrick J., Z. Milosevic, and K. Raymond. Policies in an enterprise specification. In Morris Sloman, editor, *Proc. of Policy Worshop, 2001, Bristol UK*, January 2001.

[5] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In Morris Sloman, editor, *Proc. of Policy Worshop, 2001, Bristol UK*, January 2001.

[6] ISO. Open distributed processing—enterprise language. Technical report, ISO/IEC CD 15414, January 1998.

[7] P.F. Linington, Z. Milosevic, and K. Raymond. Policies in communities: Extending the odb enterprise viewpoint. In *Proceedings of the Second Internantional Enterprise Distributed Object Computing (EDOC98) Conference*. IEEE, November 1998.

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

[9] N.H. Minsky. A scalable mechanism for communal access control in distributed systems. Technical report, Rutgers University, October 2000.

[10] 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.

[11] B. Schneier. *Applied Cryptography.* John Wiley and Sons, 1996.

[12] M.W.A. Steen and J. Derric. Formalizing ODP enterprise policies. In *Proceedings of the Third Internantional Enterprise Distributed Object Computing (EDOC99) Conference*, pages 84–94. IEEE, September 1999.

[13] 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.