DIMACS Technical Report 2003-04
February 2003

Flexible Regulation of Distributed Coalitions

by

Xuhui Ao[1] and Naftaly H. Minsky[2]
{ao,minsky}@cs.rutgers.edu
Department of Computer Science
Rutgers University, Piscataway, NJ 08854

[1]Work supported in part by DIMACS under contract STC-91-19999, and by NSF grants No. CCR-98-03698
[2]Work supported in part by NSF grants No. CCR-98-03698

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, NEC Research Institute and Telcordia Technologies (formerly Bellcore).

DIMACS was founded as an NSF Science and Technology Center, and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

There is a growing tendency for organizations to form coalitions in order to collaborate—by sharing some of their resources, or by coordinating some of their activities. The question addressed in this paper is: how should such coalitions be regulated?

Our approach to this question is based on the following definition of the governance of coalitions:

> A coalition C is a set $\{E_1,..., E_n\}$ of enterprises, which interoperate under an ensemble of policies $[P_C, \{P_i\}]$, where P_C is the *coalition policy* that governs the coalition as a whole, and P_i is the *local* policy of enterprise E_i (for each i), which governs its participation in the coalition.

This means, in particular, that every interaction between an agent x_i of enterprise E_i and an agent x_j of E_j must comply with the local policies P_i and P_j, as well as with the coalition policy P_C.

We also require that the policy-ensemble $[P_C, \{P_i\}]$ of a coalition would satisfy the following *principle of flexibility*:

> Each local policy P_i can be defined and changed independently of other local policies in this ensemble, and without any knowledge of them.

We will describe a regulatory mechanism for coalitions, which provides for efficient and decentralized enforcement of a wide range of policies that might govern a coalition, and which satisfies our principle of flexibility.

1 Introduction

There is a growing tendency for organizations to form coalitions in order to collaborate—by sharing some of their resources, or by coordinating some of their activities. Whether these are business enterprises, educational institutions, or governmental agencies, there is a need for a coalition between them to be regulated, in order to ensure conformance with the policy that, presumably, governs the coalition as a whole, and in order to protect the interest of member organizations. This need triggered a great deal of recent access-control research for coalitions [7, 10, 11, 15, 17, 18], exhibiting several different views of the problem, and employing different techniques for its solution. Our own view of this problem, which is more general than most of the above, is based on the following definition of coalitions.

Definition 1 *A coalition C is a set $\{E_1,..., E_n\}$ of enterprises*[1]*, which interoperate under an ensemble of policies $[P_C, \{P_i\}]$, where P_C is the* coalition policy *that governs the coalition as a whole, and P_i is the* local *policy of enterprise E_i, which governs its participation in the coalition.*

In Section 2 we will explain the need for a coalition to be governed by such an ensemble of policies, and illustrate the nature of such an ensemble. Here we just point out that this definition means that every interaction between an agent x_i of enterprise E_i and an agent x_j of E_j must comply with the local policies P_i and P_j, as well as with the coalition policy P_C.

Two issues need to be addressed when establishing a regulatory framework for such a coalition: (a) the policy-enforcement mechanism, and (b) a mechanism for the specification of the policy-ensemble that is to govern a coalition. Regarding the former issue, we adopt here our own access-control mechanism called Law-Governed Interaction (LGI), which supports a wide range of policies, enforcing them in decentralized manner. In Section 3 we provide an overview of this mechanism—which has been originally introduced in [12], and then implemented and further developed, as described in [13, 14, 2], in particular.

We will be mostly concerned, in this paper, with the structure and specification of the policy-ensemble of a coalition, requiring it to satisfy the following principle of *flexibility*:

Principle 1 (flexibility) *Each local policy P_i, in a policy-ensemble $[P_C, \{P_i\}]$, can be defined and changed independently of other local policies in this ensemble, and without any knowledge of them.*

Such a flexibility is important for several reasons. First, it provides each enterprise with the *autonomy* to define its own policy at will, subject only to the pre-agreed coalition policy P_C. Second, the mutual independence of the local policies of member enterprises simplifies their formation and their evolution. Finally, this principle allows an individual enterprise to keep its own policy *confidential*, since policies of other enterprises do not depend on it.

[1] The term “enterprise” in this definition refers to educational and governmental institutions, as well as to commercial ones.

To appreciate some of the difficulties in satisfying this principle, consider the following question: how does one ensure that an interaction between an agent x_i of enterprise E_i and an agent x_j of E_j conforms to policies P_i, P_j and P_C—which govern it? A seemingly natural answer to this question is to *compose* policies P_i, P_C, and P_j into a single policy, to be enforced by a reference monitor mediating all coalition-relevant interactions between the two enterprises E_i, E_j. This approach has, indeed, been attempted by several researchers [8, 5, 11] concerned with the interoperability between agents subject to different policies. But such composition of policies has several serious drawbacks, particularly when employed in the context of coalitions.

First, composition of policies is computationally hard. According to [11], in particular, such composition is intractable for more than two policies, even for a relatively simple policy language. This is particularly serious problem for a coalition, which would require a quadratic number (in terms of its membership size) of compositions of triples of policies—a truly daunting prospect. Second, composition violates our principle of flexibility. This is because for an enterprise E_i to formulate, or change, its local policy P_i, it will have to be aware of the local policies of every other member-enterprise, say P_j—lest its new policy will prove to be inconsistent, and thus not composable, with P_j. (We will later review other attempts to regulate coalitions, not based on policy composition.)

The Proposed Approach: We adopt a *top-down* approach to the specification of the policy ensemble of a coalition. That is, we propose to start by formulating the global coalition policy P_C, which specifies its constraints over interoperability between different coalition members. We then allow individual members to formulate their own local policies, subject to P_C, but independent of each other.

This approach is supported by an hierarchical organization of policies, recently implemented into LGI. Such an hierarchy is formed via a *superior/subordinate* relation between policies, where a *subordinate* policy is **structurally constrained** to conform to its *superior* one. Under such an organization of policies, the local policies P_i could be defined as subordinate to P_C. A pair of local policies P_i and P_j, thus defined, are consistent with each other *by definition*, although they have been formed with no knowledge of each other, because both are guaranteed to conform to the same global policy P_C. As we shall see, the enforcement of such a hierarchical policy-ensemble can be carried out in a completely decentralized manner, without ever having to compose two independently defined policies, such as P_i and P_j, into a single policy.

The remainder of the paper is organized as follows: Section 2 motivates the proposed approaches to the governance of coalitions, illustrating it via a fairly realistic example; this section also compares the proposed approach to related work. Section 3, provides an overview of the concept of law-governed interaction (LGI). Section 4 shows how policies under LGI can be organized into hierarchies; and Section 5 presents a formalization of the example policy-ensemble introduced in Section 2, via the the LGI hierarchy model. We conclude in Section 6.

2 On the Nature of the Proposed Regulatory Framework for Coalitions

We have just proposed a regulatory framework for coalitions, under which a coalition C of enterprises $\{E_1,..., E_n\}$ is governed by an ensemble of policies $[P_C, \{P_i\}]$, which satisfies the *flexibility principle*. In this section we will attempt: (1) to motivate this coalition model and to illustrate it by means of a detailed example; and (2) to compare our model with other approaches to the governance of coalitions. But we first offer some general remarks about the roles that the various policies in this ensemble are expected to play, and about what they are expected to regulate.

We expect the coalition policy P_C to represent prior agreement, or contract, between its member enterprises—and possible, also, the governmental regulation about the operations of such coalitions. Generally speaking, P_C can expect to specify such things as the coalition membership, and the required interaction protocol between agents of different member enterprises; and it might provide certain coalition officers with the ability to regulate dynamically the interaction between member enterprises. We will not be concerned in this paper with how such a policy is established (This issue has been addressed by others. In particular, a negotiation process between member enterprises to establish a compromised P_C has been presented in [7]).

We expect the local policy P_i of an enterprise E_i to be concerned with internal matters of this enterprise, as they are related to coalition activities. Thus, P_i might specify such things as: (a) which of its *agents* (i.e., people or system-components) are allowed to participate in the coalition activity, either as servers for fellow coalition members, or as clients for services provided by fellow members; (b) the amount of usage a given agent of this enterprise is allowed to make, of services provided by fellow member enterprises, and the amount of services a given server is allowed to provide to the coalition; and (c) the required behavior of agents of this enterprise, when serving requests from other coalition members, or when making such requests—for example, P_i may require certain coalition activities of its agents to be monitored.

2.1 An Example of a Policy-Ensemble

Consider a set $\{E_1,..., E_n\}$ of enterprises that form a coalition C in order to collaborate by using some of each other's services—as is increasingly common under the *grid computing* paradigm [15, 18, 17], in particular. Suppose that the coalition as a whole has a distinguished agent D_C called its *director*, and that each member enterprise E_i has its own director D_i. The function of these directors is, in part, to serve as *certification authorities* (CAs), as follows: D_C would certify the directors of the member enterprises, which, in turn, would certify the names and roles of agents within their respective enterprises. These and other roles of the directors, along with other aspects of this coalition are governed by the global coalition policy P_C, and by the local policies of individual enterprises defined as *subordinate* to P_C, forming a two-level hierarchy depicted in Figure 1. These policies would now be

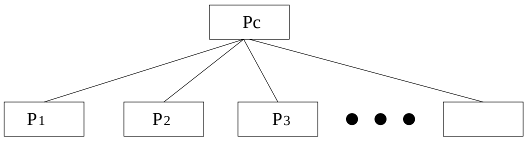

Figure 1: The Policy Ensemble Governing Coalition C.

described informally. They would be formalized as "laws" under LGI in Section 5.

Let the the global policy P_C of this coalition deals with two issues, as described informally below:

1. Director's control over the usage of enterprise-resources:

   (a) The total amount of services offered by an enterprise E_i to other coalition members is determined by director D_i, by offering to the coalition director D_C a budget *B_i for using its services. This budget is expressed in terms of what we call E_i-currency, which can be created (minted, in effect) only by the director D_i, as described above.*

   (b) The coalition director D_C can distribute E_i-currencies he got from D_i among the directors of other member-enterprises.

   (c) E_i-currency can be used to pay for services provided by agents of E_i. Such currency can also be moved from one agent to another within an enterprise, subject to the local policy of that enterprise—but it cannot be forged.

2. Regulation over service requests by agents at enterprise E_i to agents in E_j:

   (a) each such request must carry the name of the sender, as authenticated by director D_i, and must be sent to an agent authenticated as a `server` *by the director D_j.*

   (b) Each such request must carry a payment in E_j-currency, which would be moved by this request from the client to the server. However, if the request is rejected, for whatever reason[2], then the payment will be refunded to the client.

Thus, policy P_C mandates various global aspects of the coalition, including: (a) the authority, and implied responsibilities, of the various directors; (b) the requirement that service budgets minted by directors should be treated as currency, which can be *moved* from one agent to another, in particular as part of a service request, but cannot be copied or forged; and (c) the requirement that service requests must carry the proper authentication of the clients and be accepted by authenticated servers.

[2]The following are among the reasons for a service request to be rejected: (1) the receiver is not an authenticated server; (2)the type of currency used for payment doesn't match the server's enterprise; and (3) the request is not allowed to be served by the server enterprise's local policy.

But policy P_C does not address many aspects of the coalition, leaving them for the local policies of member enterprises to specify. These include, for an enterprise E_i, say, such things as: (a) how is E_j-currency, provided to D_i by the coalition director, to be distributed among the various agents of E_i; (b) how much should servers in E_i charge for their services, and what should they do with payments received; and (c) various conditions that must be satisfied, for agents of E_i to use services of other enterprises, or to provide services to others—such as time constraints and auditing requirements. The following are two example of such local policies, for member enterprises E_1 and E_2.

Policy P_1 of enterprise E_1:

1. *The director D_1 of this enterprise (E_1) can move service currencies obtained from D_C to any other authenticated agent in its enterprise.*

2. *Copies of all service requests sent by agents of this enterprise, and of all requests received by its servers, must be sent to a designated audit-trail server.*

Policy P_2 of enterprise E_2:

1. *Any agent in this enterprise (E_2) can give service currencies it has, to any other agents that has been duly authenticated by the director D_2. (That is, currencies can move freely from one agents to another, unlike under P_1, where currencies can only be granted by the director to regular agents.) Also, every transfer of currency is to be audited by a distinguished audit-trail server.*

2. *Agents of this enterprise are allowed to provide services to other coalition members, or to request such services, only after normal working hours (namely, from 6:00PM to 8:00AM).*

Note that it is an essential aspect of the policy-ensemble of this coalition that it is hierarchical, as depicted in Figure 1. As we shall see in Section 4, this means that all local policies P_i *conform* to P_C, so that none of them would be able to violate any of the provisions of P_C. For example, it would not be possible to write a local policy for an enterprise E_j, say, that allows agents of E_j to forge E_i-currency. This provides an assurance for the director D_i of enterprise E_i that the total amount of E_i-currency circulating among the agents of the various coalition members—and, thus, the total amount of services its agents may be asked to perform—does not exceed the service budget B_i he originally sent to the coalition director D_C.

Therefore, the various member enterprises have the flexibility of being able to write their policies independently of the policies of other members, and they can change their policies at will. And yet, each enterprise can be confident that its interlocutors conform to the same common coalition-policy P_C.

Figure 2 provides an overview of the governance of coalition C by its hierarchical policy-ensemble. The outer box in this figure represents policy P_C, which governs the local policies P_i—represented by boxes nested in it—as well as the coalition director, which operates

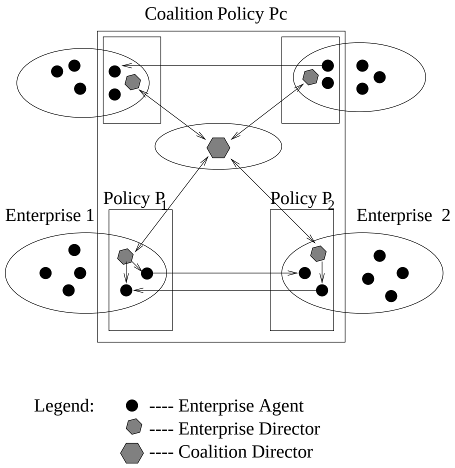

Figure 2: The distributed coalition composed of multiple enterprises and their policies.

directly under P_C. The directors of member-enterprises, represented by shaded shapes, along with other agents of such enterprises, are governed directly by their own policies, but indirectly by P_C. This figure attempts to illustrate two notable aspects of these policies: (1) the coalition director D_C is only involved in the distribution of service currency among the directors of member enterprises. He, she or it is not involved in the actual sharing of services, which is done directly between the agents of different member enterprises. (2) the local policy P_i does not, necessarily, governs the entire enterprise E_i, but only the involvement of E_i in the working of coalition C.

2.2 Other Approaches to the Governance of Coalitions

Our view that coalitions need to be governed by an ensemble of policies $[P_C, \{P_i\}]$ is not the common view of the governance of coalitions—not to speak of our hierarchical structure of such an ensemble, which is quite uniqe. In this section we review several other approaches to this issue—starting with projects that employs only local policies, and ending with those that employ both local and global policies, but in different manners than ours. All the projects to be reviewed below also differ from ours in their enforcement mechanisms, and in their expressive power—which is generally based on stateless RBAC models, and are much weaker than ours. We will comments about these issues, wherever appropriate.

(I) Shands et al. [17], employ only local P_i policies, with no global coalition policy. Moreover, the only control that these policies may have over the ability of agents to issue service requests, is by assigning them to roles. The server enforces his own policy, as well as of his client, for each service request. Thomson et. al. [18] enterprises express their local policies by issuing digitally-signed certificates to define their policy regarding a resource in which they have a stake. Only when all the stakeholders' access requirements are met can the access be performed. Similarly to [17], all policies are enforced by the server asked to perform a service.

There are several problems with this kind of *server-centric* policy enforcement approach in the coalition context: First, the client enterprise needs to trust the distributed heterogenous servers to implement and enforce correctly not only its local enterprise policy, but also that of the client enterprise. Furthermore, the server needs to know the local policy of the client enterprise, which violates the *confidentiality* requirement of our principle of flexibility. Finally, it is virtually impossible for a server to enforce a *stateful* client's policy, such as our policies regarding the movement of currency on the client side.

(II) Several projects take a bottom-up approach. They start with the local P_i policies, but attempt, in various ways, to form a global coalition policy from them. These include the already mentioned work of McDaniel et al. [11], which attempted to compose the local policies automatically into a single common policy, finding this task to be computationaly hard. Another approach to this problem is that of Gligor et al. [7], which tries to establish the common access policy via negotiation among the coalition members.

(III) Finally, we are aware of two projects that, like us, view a coalition as being governed by a global coalition policy as well as by the local policies of individual coalition members. One of these is the project of Pearlman et al. [15], where servers are expected to take into account certificates issued by a centralized enforcer, driven by the coalition policy. The other such project is that of Belokosztolszki et al. [4]. They introduce the concept of *meta policy*, which is expected to be conformed to by all local policies of member enterprises, and is, thus, analogous to our coalition policy P_C. However, they do not propose any computational means for ensuring this conformance, or verifying it—which in our case would be done automatically via the hierarchical structure of policies.

3 Law-Governed Interaction (LGI)–an Overview

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 arbitrary languages, or human beings. Both the *clients* and the *servers* of the traditional distributed system terminology are viewed here as agents. All such members are treated as black boxes by LGI, which deals only with the interaction between them via $\mathcal{L}$-messages, ensuring conformance to the law of the community. (Note that members of a community are not prohibited from non-LGI communication across the

[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.

Internet, or from participation in other LGI-communities.)

We now give a brief discussion of the concept of law, emphasizing its local nature, a description of the decentralized LGI mechanism for law enforcement, and its treatment of digital certificates. We do not discuss here several important aspects of LGI, including its concepts of *obligations* and of *exceptions*, the expressive power of LGI, and its efficiency. For these issues, and for implementation details, the reader is referred to [14, 1].

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, for monitoring purposes, say.

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. For example, under law $\mathcal{P}_{\mathcal{C}}$ to be introduced in section 5, as a formalization of our example P_C policy, the term `budget(`B_i`,`E_i`)` in the control-state of an agent denotes the amount of budget this agent gets from its director or other agents; while `role(server)` in the control-state of an agent denotes that this agent has been certified as a server in the coalition.

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 of 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 3(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 $\mathcal{L}$. For example, under law $\mathcal{P}_{\mathcal{C}}$, an agent can only send a service request with the payment that doesn't exceed the current budget in its CS. A message `giveCurrency(...)` will cause the specific service currency to be reduced from the CS of the sender and added to that of the receiver.

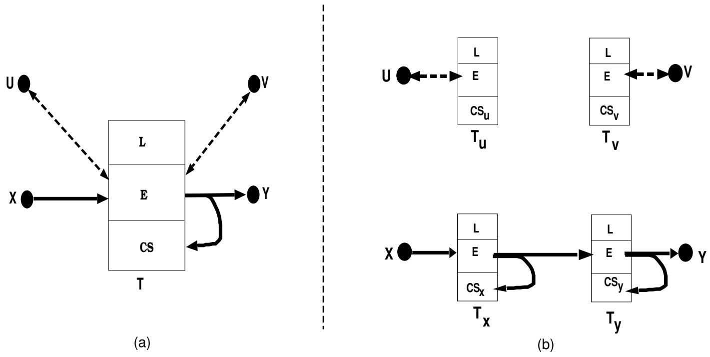

Figure 3: Law Enforcement: (a) centralized version; (b) decentralized law enforcement under LGI

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 [9], for example. The problem, in a nutshell, is that if there are several replicas of $\mathcal{T}$, then any change in $\mathcal{CS}$, like the reduction of the service currency from a sender x of message `giveCurrency(...)`, in the example above, would have to be carried out *synchronously* at all the replicas; otherwise x may be able to send more service currency to other agents than what it actually has, 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 $\{\mathcal{T}_x \mid x \in \mathcal{C}\}$ of, what we call, *controllers*, one for each members of community $\mathcal{C}$ (see Figure 3(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.

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; and the *submission of a digital certificate* (more about the later kind of event, later). 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 and operations on messages, such as `forward` and `deliver`. To summarize, an LGI law must satisfy the following locality properties:

(a) a law can regulate explicitly only *local events* at individual agents; (b) 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 (c) 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 $\{\mathcal{T}_x \mid x \in \mathcal{C}\}$ 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 3(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 3(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 [14], 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.

On the structure and formulation of laws: Broadly 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.)

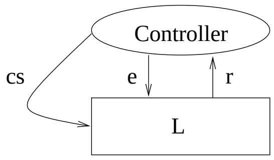

Figure 4: An LGI Law, in Context

More formally, an LGI law $\mathcal{L}$ is a function (called the *ruling* function) of the following form:

$$r = L(e, cs), e \in E, cs \in CS, r \in R \tag{1}$$

where E is the set of regulated events, CS is the set of control states, and R is the set of all possible sequences of operations that constitute the ruling of the law. Figure 4 illustrates the way that this function is used when it is deployed within a controller. When an event e occurs in this controller, it is supplied as an input to the law (function), along with the current control state `cs` maintained by the controller. The law then evaluated the ruling r, and returns it to the controller, which carries it out.

Concretely, such a function can be expressed in many languages. Our middleware currently provides two languages for writing laws: Java, and a somewhat restricted version of Prolog [6]. We employ prolog in this paper. In this case, the law is defined by means of a Prolog-like program `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 5 and the relationship among the message related regulated events (*sent, arrived*) and message related primitive operations (*forward, deliver*) can be illustrated by Figure 6.

On the basis for trust between members of a community: For a member of an $\mathcal{L}$-community to trust its interlocutors to observe the same law, one needs the following assurances: (a) 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. If these conditions are satisfied, then it follows that if `x` receives an $\mathcal{L}$-message from some `y`, this message must have been sent as an $\mathcal{L}$-message; in other words, that $\mathcal{L}$-messages cannot be forged.

Broadly speaking, these assurances are provided as follows: Controllers used for mediating

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` from the control state with term `t2`;

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 `y`

Miscellaneous	
`t@L`	returns true if term `t` is present in list L, and fails otherwise

Figure 5: Some primitive operations in LGI.

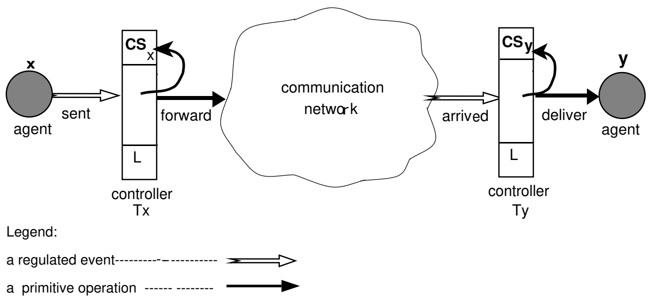

Figure 6: Decentralized enforcement of the law

the exchange of $\mathcal{L}$-messages authenticate themselves to each other via certificates signed by a certification authority specified by the value of the `ca` attribute in the `law` clause of law $\mathcal{L}$ (see, for example, figure 8, in the case of law P_C). Note that different laws may, thus, require different certification levels for the controllers used for its enforcement. Messages sent across the network are digitally signed by the sending controller, and the signature is 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 [16] `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 a server, in some client-server situation, commits itself to accept service requests only via $\mathcal{L}$-messages, then the clients will be compelled to send their requests under law $\mathcal{L}$, if they want their requests to be serviced. Conversely, once clients are used to send their requests via $\mathcal{L}$-messages, the servers would be effectively compelled to operate under law $\mathcal{L}$.

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. 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 explained next.

3.2 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 8 for an example). If this attempt is successful[4], 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:

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

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

Here `I` and `S` are internal representations of the public-keys of the CA that issued this certificate, and of its subject, respectively; `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(johnDoe), role(directorOf(e1))]`, asserting that the name of the subject in question is JohnDoe and his role is the director of a coalition member enterprise $e1$. Additional components of the attributes field could include the expiration time of the certificate, the URL of the server that maintains CRLs for this type of certificates, a certificate id (used to identify it in CRLs), etc. Of course, what happens when the `certified` event is triggered will depends on the law.

Due to lack of space, we don't discuss here how to deal with the certificate expiration and revocation in LGI. For those issues, the reader is referred to [2].

4 The LGI Law-Hierarchy

We will introduce here the concept of law-hierarchy that formalizes the policy-hierarchy described in Section 2. Each such hierarchy, or tree, of laws $t(\mathcal{L}_0)$, is rooted in some law $\mathcal{L}_0$. And each law in $t(\mathcal{L}_0)$ is said to be (transitively) *subordinate* to its parent, and (transitively) *superior* to its descendents. (As a concrete example of such a hierarchy we will use the two-level tree $t(\mathcal{P}_C)$ which is the formalization under LGI of the policy hierarchy depicted in Figure 1.)

Generally speaking, each law $\mathcal{L}'$ in a hierarchy $t(\mathcal{L}_0)$ is created by *refining* a law $\mathcal{L}$, the parent of $\mathcal{L}'$, via a *delta* $\overline{\mathcal{L}'}$, where a delta is a collection of rules defined as a refinement of an existing law (we will, generally denote a delta via an overline above the name of the law they create). The root $\mathcal{L}_0$ of a hierarchy is a normal LGI law, except that it is created to be *open* for refinement, in a sense to be explained below. This process of refinement is defined in a manner that guarantees that every law in a hierarchy *conforms* to all its superior laws—as we shall see later.

For example, in coalition C, the local law $\mathcal{P}_1$ of enterprise E_1 would be created by refining the coalition law $\mathcal{P}_C$ by means of delta $\overline{\mathcal{P}_1}$, defined in Figure 10. Similarly, law $\mathcal{P}_2$ is defined by refining the coalition law via a different delta, $\overline{\mathcal{P}_2}$, defined in Figure 11. This is how the hierarchy $t(\mathcal{P}_C)$—the ensemble of policies governing coalition C—is formed.

To be specific, we will first introduce an abstract—language-independent—model for law-refinement, and thus of the concept of hierarchy. Then we will describe a concrete realization of this model, for laws written in Prolog. Finally, we will discuss the basis for trust between members of different communities under law-hierarchy model. A qualification is in order here: our model of law-hierarchy has been completed, and implemented for a two-level hierarchy, which is what we need for coalitions; the model for hierarchies of arbitrary height has not been completed yet.

4.1 An Abstract Model

Recall that an LGI law $\mathcal{L}$ as described in Section 3 is essentially a *ruling* function:

$$r = L(e, cs), e \in E, cs \in CS, r \in R \tag{2}$$

where E is the set of regulated events, CS is the set of control states, and R is the set of all possible sequences of operations that constitute the ruling of the law.

We will now *open up* this function, by: (a) allowing it to consult a collection of rules designed to refine it—called a *delta*; and (b) by taking the advice returned by this delta into account, when computing its ruling. This is done by replacing the L function defined in Equation 2 with a pair of functions: a *consultation function* L_C, defined in Equation 3, and a *ruling function* L_R, defined in Equation 4. (Variables e and cs in these equations, belong to the domains E and CS, as in Equation 2).

$$pe = L_C(e, cs), pe \in E \tag{3}$$

$$r = L_R(e, cs, pr), pr \in R \tag{4}$$

These two functions are evaluated in succession, as follows (see also Figure 7): when an event e is submitted to law $\mathcal{L}$ for evaluation, the function L_C is evaluated first, producing a *pseudo event pe*, which is passed to delta $\overline{\mathcal{L}'}$ for *consultation*. The delta operates just like the standard LGI law, producing its ruling—called, in this context, a "proposed ruling" *pr*—which is then fed as an input to the ruling function L_R. The result r produced by L_R is, finally the ruling of law $\mathcal{L}$.

Suppose now that unlike the open law $\mathcal{L}$, its delta $\overline{\mathcal{L}'}$ is closed; that is, it is a single function, defined by Equation 5, which computes its proposed ruling *pr* without consulting anything.

$$pr = \overline{L'}(pe, cs), pe \in E, cs \in CS, pr \in R \tag{5}$$

Then the refinement of law $\mathcal{L}$ via delta $\overline{\mathcal{L}'}$, produces a regular LGI law $\mathcal{L}'$—which is not open to further refinements—as illustrated by Figure 7.

Now, note that the ruling function of this *closed* law $\mathcal{L}'$ is a composition of functions 3, 4 and 5 above, which has the following form:

$$r = L'(e, cs) = L_R(e, cs, \overline{L'}(L_C(e, cs), cs)) \tag{6}$$

Law $\mathcal{L}'$ defined by this ruling function is called a *subordinate* to law $\mathcal{L}$.

It should be pointed out that we limited ourselves here to closed deltas, which do not consult anything when computing there rulings. Such deltas can produce hierarchies of depth two, at most—which is all we need in this paper. However, this model can be extended to open deltas that consult other deltas at a lower level, thus producing a cascade of refinements, and a hierarchy of arbitrary depth. Such a model outlined in [3], but is beyond the scope of this paper.

Finally, we can now explain the sense in which a law $\mathcal{L}'$, which is subordinate to $\mathcal{L}$ in a hierarchy, is said to *conform* to its superior law $\mathcal{L}$: Law $\mathcal{L}'$ is created by refining $\mathcal{L}$ via a

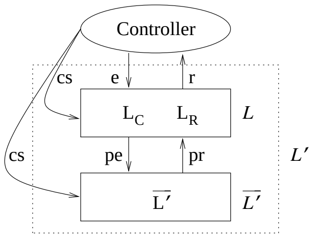

Figure 7: Refinement of law $\mathcal{L}$ via delta $\overline{\mathcal{L}'}$, producing law $\mathcal{L}'$

delta $\overline{\mathcal{L}'}$. As is evident from both Figure 7 and Equation 6, this is done by having the ruling of the delta submitted as an input to the ruling function of $\mathcal{L}$, which finally produces the ruling. Thus, the final decision about the ruling of law $\mathcal{L}'$ is made by its superior law $\mathcal{L}$, leaving to its deltas only an advisory role.

4.2 A Concrete Realization of Law Hierarchy, for Laws Written in Prolog

We now describe how our current Prolog-based language for writing laws has been extended to implement the above abstract model of law hierarchies. We start with the manner that a law can consult its deltas, then we describe the structure of the deltas themselves, and we conclude with the way that a law can decide on the disposition of ruling proposals returned to it by its deltas.

Consulting deltas: Suppose that we are operating under law $\mathcal{L}'$, which is a refinement of law $\mathcal{L}$ via delta $\overline{\mathcal{L}'}$. An arbitrary consultation function L_C (see Equation 3) can be defined into a law $\mathcal{L}$, by inserting a clause of the form `delegate(g)`, anywhere in the body of any rule of $\mathcal{L}$, where `g` is an arbitrary Prolog term. The presence of a `delegate(g)` clause serves to invite refining deltas to propose operations to be added to the ruling being computed.

To see the effect of the `delegate(g)` clause, more specifically, suppose that law $\mathcal{L}$ contains the following rule r:

```
h :- ..., delegate(g), ...
```

If the evaluation of $\mathcal{L}$ gets to the `delegate(g)` clause of rule `r`, then goal `g` is submitted to the delta $\overline{\mathcal{L}'}$ for evaluation, playing the role of the pseudo event *pe* in Equation 3. This evaluation by the delta will produce a list of operations *pr*, which would be fed back to $\mathcal{L}$, as the ruling proposal of the delta, for goal `g`. The operations thus proposed are provisionally added to the ruling, but their final disposition will be determined by $\mathcal{L}$, as we shall see later.

Finally, note that `delegate(g)` clause in a law $\mathcal{L}$ has no effect when $\mathcal{L}$ has no delta.

The structure of law deltas: A refining delta $\overline{\mathcal{L}'}$ of a law $\mathcal{L}$ looks pretty much like the root-law $\mathcal{L}_0$ of the law-tree, with two distinctions:

First, the top clause in the delta is

```
law(name(L'),ca(pk)) refines L,
```

where `L` is the name of the law being refined, `L'` is the name of this delta, and `pk` is the public key of the certifying authority to certify the controllers enforcing law `L'`.

Second, the heads of the rules in $\overline{\mathcal{L}'}$ need to match the goals delegated to it by law $\mathcal{L}$, and not the original regulated events that must be matched by the rules of the root-law $\mathcal{L}_0$. Although the goals delegated to a refining delta can, and often do, take the form of regulated events, like `sent(...)`, and `arrived(...)`, as is the case in our case study in Section 5.

Finally, note that each delta has read access to the entire control-state (CS) of the agent. That is, the rules that constitute a given delta can contain arbitrary conditions involving all the terms of the CS.

The disposition of ruling proposals: A law $\mathcal{L}$ can specify the disposition of operation in the ruling proposal returned to it by any refining delta $\overline{\mathcal{L}'}$. This is done via *rewrite rules* of the form:

```
rewrite(O) :- C,replace(Olist)
```

where `O` is some term, `C` is some condition, and `Olist` is a possibly empty list of operations. The effect of these rules are as follows. First let rp be the set of terms proposed by a refining delta in response to the execution of a delegate clause in $\mathcal{L}$. For each term p in rp, a goal `rewrite(p)` is submitted for evaluation by law $\mathcal{L}$. If this evaluation fails, which happens, in particular, if none of the `rewrite(O)` rules in $\mathcal{L}$ matches this goal, then term p is added to the ruling of law $\mathcal{L}$.

If, on the other hand, the evaluation succeeds by matching one of the `rewrite` rules and condition `C` of this rule evaluates to true, then p is replaced by the list `Olist`. `Olist` is then added to the ruling of $\mathcal{L}$. Note that if `Olist` is empty, then term p would be discarded in spite of its inclusion in the ruling proposal made by the refining delta. For example, the *rewrite rule* of law $\mathcal{P}_{\mathcal{C}}$ in Figure 8, which is the implementation of our example coalition policy, will discard any proposed `forward` operations of `service currency` movement from its refining delta to make sure no service currency can be forged. Further, `C` cannot contain a `delegate` clause so that no further consultation is possible with the refining delta on the disposition of p; we believe that this constraint keeps the model easy to understand without real loss of flexibility.

So, the `rewrite` rules of a law $\mathcal{L}$ determine what is to be done with each operation proposed by a refining delta: whether it should be blocked, included in the ruling, or replaced by some list of operations. Note that each `rewrite` rule is applied to the ruling proposal returned by a refining delta, regardless of which `delegate` clause originally led to the consultation with the refining delta.

Finally, LGI features another technique to regulate the effect of a refining delta on the eventual ruling of the law. It can protect certain terms in the control-state from modification by refining deltas of a given law $\mathcal{L}$. This is done by including the clause

```
protected(T)
```

in the `Preamble` clause of law $\mathcal{L}$ (see Figure 8 for example), where `T` is a list of terms. For example, if the following statement appears in $\mathcal{L}$,

```
protected([name(_),role(_)])
```

then no refinement of $\mathcal{L}$ can propose an operation that modifies the terms `name` and/or `role`. Strictly speaking, such protection of terms in the control state can be carried out via `rewrite` rules, but the `protected` clauses are much more convenient for this purpose.

4.3 On The Basis for Trust Between Members of Different Communities

Recall that a law may require that all the controllers used to interpret it are certified by a specific CA. And different, independent, laws may require different CAs for this purpose (see Section 3.1). The situation with laws related via the *subordinate* relation is as follows: Consider a law $\mathcal{L}$ that requires certification by ca, and suppose that law $\mathcal{L}'$, which is subordinate to $\mathcal{L}$, requires certification by ca'. In this case we require the controllers interpreting law $\mathcal{L}'$ to be certified by both CAs: ca and ca'.

Now, consider agents x and y operating under laws $\mathcal{L}_x$ and $\mathcal{L}_y$, respectively. And suppose that these laws permit these agents to exchange messages, provided that both laws above are *subordinate* to a common law, say $\mathcal{L}$. When these two agents communicate, it would be necessary to ensure that: (a) the exchange is mediated by properly certified controllers; and (b) that these controllers are interpreting laws that have a common superior.

The first assurance above is obtained by having the controller of x (and similarly for that of y) authenticated by the CA required by the common superior law $\mathcal{L}$, as well as by the CA required by law $\mathcal{L}_x$, if any.

The second assurance is obtained as follows: when controller $\mathcal{T}_x$ forwards the message sent by x to the controller $\mathcal{T}_y$ of y, it appends to this message both $H(\mathcal{L})$ and $H(\mathcal{L}_x)$, i.e., the one-way hash H of both laws $\mathcal{L}$ and $\mathcal{L}_y$. Controller $\mathcal{T}_y$ would accept this as a valid message, in this case, if and only if $H(\mathcal{L})$, contained in this message, is identical to the hash of the law that is superior to that it operates under. In our law-language this kind of check is carried out by the predicate

`conforms(`L'`,`L`),`

which is true if and only if law $\mathcal{L}'$ is identical, or subordinate, to law $\mathcal{L}$. (See Figure 8 for an example of the use of this predicate.)

5 A Case Study

We now show how the policy hierarchy described in Section 2 can be specified in LGI, using the mechanisms just introduced. We start by formalizing the coalition policy P_C, making it into an LGI law $\mathcal{P}_\mathcal{C}$. We then formalize policies P_1 and P_2 via deltas that refine law $\mathcal{P}_\mathcal{C}$. And we conclude this section with a brief discussion.

5.1 Establishing the Coalition Policy P_C

Law $\mathcal{P}_\mathcal{C}$, which implements policy P_C, is shown in Figures 8 and 9. This, like any other LGI laws, has two parts: (a) a `Preamble`, which specifies such things as certifying authorities acceptable to this law, and the initial control state of agents operating under it; and (b) the set of rules, whose formal statement is followed by informal comments, in italics.

$\mathcal{P}_\mathcal{C}$'s `Preamble` has several clauses. The `law` clause indicate that this is a root-law, specifying its name, and the public key of the CA that is to be used for certifying the controllers interpreting this law. The `authority` clause specifies the public key of a CA—the `coalition director` D_c, whose certification would be accepted by this law for the authentication of the `member enterprise directors` D_i—as we shall see. The `initialCS` clause specifies that the the initial control state of everybody who adopt this law would be empty. Finally, the `protected` clause specifies four control-state terms that would be protected from change by refining deltas. They are: `myDirector(_,_)`, `budget(_,_)`, `name(_)`, and `role(_)`, which we will discuss later.

Our discussion of the rules of this law is organized as follows: We start with how directors of enterprises, and agents working for them, are authenticated; and how an agent can claim his official names and the role he is to play within its enterprise. Second, we show how service budgets are allocated by enterprise directors, how they are distributed by the coalition director, and service currency is to be moved from one agent to another. Third, we discuss how service request are sent, and what happens if a request is rejected. Finally, we show how the coalition law $\mathcal{P}_\mathcal{C}$ limits the power of the deltas that might be used to refine it.

(a) Coalition membership, and agent authentication: For the director D of enterprise E to claim his role under this law, he needs to present a certificate issued by the coalition director D_C, with the attribute `role(directorOf(E))`. By Rule $\mathcal{R}1$, this would cause the term `directorOf(E)` to be added to his control-state.

For any other agent `x` of the enterprise E to join the coalition, it needs to present two certificates: One, issued by coalition director D_C to certify its director D, and getting its public key ; this is done via Rule $\mathcal{R}1$, which will add the term `myDirector(D,E)` to the CS of x. The other, issued by its director `D` to certify the name and role of `x`; this is done via Rule $\mathcal{R}2$, which will add the attribute `name(N)` and possible `role(R)`, to the CS of x.

Note that the terms added to the CS of various agents as the result of such certification serve as a kind of *seals* that authenticate the role of these agents for their interaction with other agents in the coalition.

(b) Allocation of budgets, and moving currencies: By Rule $\mathcal{R}3$, a director D of enterprise E can send a message `grantBudget(B,E)` to the coalition director D_C, which (by Rule $\mathcal{R}4$) will add the term `budget(B,E)` to the CS of D_C, upon its arrival. This budget for services by enterprise E can be redistributed (by Rules $\mathcal{R}5$ and $\mathcal{R}6$) among the directors of other enterprises.

Note that the term `budget(B,E)` in the CS of any agent represents under this law what

Preamble:

law(name($\mathcal{P}_\mathcal{C}$),ca(publicKey1)).
authority(D_C,publicKey2).
initialCS([]).
protected([myDirector(_,_),budget(_,_),name(_),role(_)]).

$\mathcal{R}$1. `certified([issuer(`D_C`),subject(D),attributes([role(directorOf(E))])]) :- if (D==Self) then do(+role(directorOf(E))) else do(+myDirector(D,E)).`

The director `D` *of enterprise* `E` *needs to be certified by a certificate issued by the* coalition director D_C.

$\mathcal{R}$2. `certified([issuer(D),subject(Self),attributes(A)]) :- myDirector(D,E)@CS, name(N)@A,do(+name(N)), if role(server)@A then do(+role(server)).`

Claiming the name and role within one coalition member enterprise `E` *via certificate issued by the director* `D` *of that enterprise.*

$\mathcal{R}$3. `sent(D,grantBudget(B,E),[`D_C`,ThisLaw]) :- role(directorOf(E))@CS, do(forward).`

Only the director of the member enterprise `E` *can contribute the initial budget* `(B,E)` *to the coalition director* D_C.

$\mathcal{R}$4. `arrived([D,Ld],grantBudget(B,E),`D_C`) :- conforms(Ld,ThisLaw), if (budget(B1,E)@CS) then do(budget(B1,E)<-budget(B1+B,E)) else do(+budget(B,E)), do(deliver).`

The contributed budget `B` *from enterprise* `E` *will be recorded as term* `budget(B,E)` *in the control state of the coalition director.*

$\mathcal{R}$5. `sent(`D_C`,grantBudget(B,E),[D,Ld]) :- conforms(Ld,ThisLaw), budget(B1,E)@CS, B1 >= B, do(budget(B1,E)<-budget(B1-B,E)), do(forward).`

The initial budget assignment from the coalition director to the member enterprise directors.

$\mathcal{R}$6. `arrived([`D_C`,ThisLaw],grantBudget(B,E),D) :- role(directorOf(Ei))@CS, if (budget(B1,E)@CS) then do(budget(B1,E)<-budget(B1+B,E)) else do(+budget(B,E)), do(deliver).`

Only the member enterprise's director can receive the initial budget from the coalition director.

Figure 8: Law $\mathcal{P}_\mathcal{C}$

$\mathcal{R}7$.
```
sent(X,giveCurrency(B,E),[Y,Ly]) :- conforms(Ly,ThisLaw), budget(B1,E)@CS,
        B1>=B, delegate(ThisGoal), if (permitS@Ruling) then
        do(budget(B1,E)<-budget(B1-B,E)), do(forward).
```

An agent can give part of its service currency to others if it is authorized by the local law delta of its enterprise.

$\mathcal{R}8$.
```
arrived([X,Lx],giveCurrency(B,E),Y) :- conforms(Lx,ThisLaw),
        if (budget(B1,E)@CS) then do(budget(B1,E)<-budget(B1+B,E)) else
        do(+budget(B,E)), do(deliver).
```

The service currency given by other agents will be added into the receiver's account.

$\mathcal{R}9$.
```
sent(X,sr(from(N),service(S),payment(P,E)),[Y,Ly]) :- conforms(Ly,ThisLaw),
        name(N)@CS, budget(B,E)@CS,B >= P, delegate(ThisGoal),
        if (permitS@Ruling) do(budget(B,E)<-budget(B-P,E)), do(forward).
```

Any service request must contain the name of the sender and the payment, and be authorized by the local law delta of the sender enterprise.

$\mathcal{R}10$.
```
arrived([X,Lx],sr(from(N),service(S),payment(P,E)),Y)
     :- conforms(Lx,ThisLaw),
        if (myDirector(D,E)@CS, role(server)@CS, delegate(ThisGoal),
        permitA@Ruling) then
        (budget(B,E)@CS, do(budget(B,E)<-budget(B+P,E)), do(deliver))
        else (name(M)@CS, do(forward(Self,srReject(from(M),service(S),
        payment(P,E)),[X,Lx]))).
```

Only a certified server can process the service request if it is authorized by its local enterprise law delta. Otherwise, the request will be rejected.

$\mathcal{R}11$.
```
arrived([X,Lx],srReject(from(N),service(S),payment(P,E)),Y) :-
        conforms(Lx,ThisLaw), budget(B,E)@CS,
        do(budget(B,E)<-budget(B+P,E)),
        do(deliver).
```

The arrival of the service reject message will restore the client's budget by that payment.

$\mathcal{R}12$.
```
rewrite(forward(X,M,[Y,Ly])) :- conforms(Ly,ThisLaw),
        if (M==(grantBudget(_,_)
        |giveCurrency(_,_)|sr(_,_,_)
        |srReject(_,_,_))) then replace([]).
```

Make sure that no subordinate law delta can violate the `properties` *of this law.*

Figure 9: Law $\mathcal{P}_{\mathcal{C}}$ - continuation

we have called E-currency, i.e., currency that can be used for paying servers in enterprise E. The law allows such currencies to be *moved* from one agent to another, subject to the various policies, but it provides no means for forging currencies. E-currency can be moved by any agent to another via the `giveCurrency(B,E)` message, regulated by Rules $\mathcal{R}7$ and Rule $\mathcal{R}8$, provided that such transfer is permitted by the enterprise refining delta.

It is instructive to observe how Rule $\mathcal{R}7$ checks for the permission of currency transfer by a refining delta. After the `delegate(ThisGoal)` clause (`ThisGoal` would be bound to the head of the evaluated rule, in this case, the currency `sent` event), a check is performed to see whether an operation `permitS` is in the ruling compiled thus far (`Ruling`); if `permitS` is present, then the currency transfer is performed.

Finally, we note that both the redistribution of the service currency by the enterprise's director, and the subsequent movement of it among the regular agents, are governed by the same Rule $\mathcal{R}7$ and Rule $\mathcal{R}8$.

(c) Regulation over service requests: By Rule $\mathcal{R}9$, an agent with official name `N`, certified by its enterprise director, can issue request for service `S` of enterprise `E` via message:

```
sr(from(N),service(S),payment(P,E))
```

if the payment `P` doesn't exceed its current budget and the request is authorized by its refining delta. If it is the case, the corresponding payment will be reduced from the sender's budget before the service request is forwarded.

By Rule $\mathcal{R}10$, only the certified server can receive a service request if the payment type is correct and the request is authorized by the server's local enterprise refining law delta. If this is the case, the two operations will be carried out: (1) the server's budget will increase by that payment, and (2) the request will be delivered to the server.

Otherwise, a service rejection message with the previous payment will be forwarded back to the client and the client will get the refund of that payment as in Rule $\mathcal{R}11$. Both Rule $\mathcal{R}9$ and Rule $\mathcal{R}10$ use the similar process as Rule $\mathcal{R}7$ to check the permission of the request sending and receiving from the refining delta.

(d) Limiting the power of refining deltas: Finally, $\mathcal{P}_\mathcal{C}$ imposes some limitations on the possible effects of refining deltas, to ensure such things as that currencies cannot be forged. First, as has already been pointed out, the `protected` clause in the preamble of this law does not allow certain terms to be changed by deltas.

Second, by Rule $\mathcal{R}12$, this law would reject any recommendation by delta to forward certain messages, such as `grantBudget` and `giveCurrency` messages that might cause, effectively, currency to be forged. This is done because $\mathcal{P}_\mathcal{C}$ cannot depend on refining deltas not to violate these important properties that our law is responsible for.

5.2 Establishing Local Policies of Member-Enterprises

Law $\mathcal{P}_1$ of enterprise E_1: this law is formed by refining law $\mathcal{P}_\mathcal{C}$ via delta $\overline{\mathcal{P}_1}$, which is shown in Figure 10. As specified in the preamble of this delta, the controllers interpreting this law would be required to be certified by the CA identified by the public key `publicKey3`—as well as by the CA required by the superior law $\mathcal{P}_C$. Furthermore, the preamble specifies the address of agent, which is to be used as an audit-trail server under this law.

Preamble:

law(name($\mathcal{P}_1$),ca(publicKey3)) **refines** $\mathcal{P}_\mathcal{C}$.
alias(e1Auditor,"e1Auditor@e1.com").

$\mathcal{R}$1. `sent(X,giveCurrency(B,E),[Y,Ly]) :-`
`role(directorOf(e1))@CS,` **do(permitS)**.

In this enterprise, only the enterprise director can give its own service currency to others.

$\mathcal{R}$2. `sent(X,sr(from(N),service(S),payment(P,E)),[Y,Ly]) :-`
do(permitS), `do(deliver(Self,ThisGoal,e1Auditor)).`

Authorize all the service requests. Furthermore, monitor all the service requests sent by the agents of this enterprise.

$\mathcal{R}$3. `arrived([X,Lx],sr(from(N),service(S),payment(P,E)),Y) :-`
do(permitA), `do(deliver(Self,ThisGoal,e1Auditor)).`

Authorize all the service requests. Furthermore, monitor all the service requests received by the servers of this enterprise.

Figure 10: Law delta $\overline{\mathcal{P}_1}$

By Rule $\mathcal{R}$1, law $\mathcal{P}_1$ allows only the director of enterprise E_1 to distribute currencies within the enterprise. No other agent can transfer its currencies to others.

Rules $\mathcal{R}$2 and $\mathcal{R}$3 ensure that copies of all the service requests sent by agents of E_1 and those received by the servers of E_1 will be sent to its own audit-trail server–`e1Auditor`.

Law $\mathcal{P}_2$ of enterprise E_2: this law is formed by refining law $\mathcal{P}_\mathcal{C}$ via delta $\overline{\mathcal{P}_2}$, which is shown in Figure 11. This law differs from $\mathcal{P}_1$, in the following respects: First, by Rule $\mathcal{R}$1, any agent in enterprise E_2 can transfer part of any service currency it has to others, and any such currency movement will be audited by the enterprise's auditor `e2Auditor`.

Also, by Rules $\mathcal{R}$2 and $\mathcal{R}$3, agents of E_2 are allowed to engaged in coalition activities only between 6:00PM to 8:00AM; where, by "coalition activity" we mean sending or receiving service requests.

$\mathcal{P}$reamble:

law(name($\mathcal{P}_2$),ca(publicKey4)) **refines** $\mathcal{P}_\mathcal{C}$.
alias(e2Auditor,"e2Auditor@e2.com").

$\mathcal{R}$1.
```
sent(X,giveCurrency(B,E),[Y,Ly]) :-
        do(permitS), do(deliver(Self,ThisGoal,e2Auditor)).
```

Any agent in this enterprise can give part of its service currency to others and that currency movement will be monitored by this enterprise.

$\mathcal{R}$2.
```
sent(X,sr(from(N),service(S),payment(P,E)),[Y,Ly]) :-
        clock(T)@CS, if (T > 6:00PM; T < 8:00AM) then do(permitS).
```

The agents of E_2 are allowed to send the coalition service requests only after the normal working hour,namely, from 6:00PM to 8:00AM.

$\mathcal{R}$3.
```
arrived([X,Lx],sr(from(N),service(S),payment(P,E)),Y) :-
        clock(T)@CS, if (T > 6:00PM; T < 8:00AM) then do(permitA).
```

The servers of E_2 are allowed to receive and process the coalition service requests only after the normal working hour, namely, from 6:00PM to 8:00AM.

Figure 11: Law delta $\overline{\mathcal{P}_2}$

5.3 Discussion

As demonstrated by the above case study, the LGI hierarchical model provides an effective and flexible means for regulating coalitions. The global coalition law can establish some general provisions, allowing member enterprises to extend and refine it as needed.

The hierarchy also provides for very flexible interoperation between different coalition members, running under subordinate laws. Indeed, different member enterprises can interoperate with each other, based on the fact that they all conform to a common coalition law, and so can be assured that everybody is "playing" by the rules set forth by the global coalition law. A significant aspect of this flexibility is that laws with a common superior can be modified at any point without affecting their interoperability.

6 Conclusions

This paper introduces a new, and fully implemented, regulatory mechanism for coalitions. The main contributions of this mechanism are as follows:

First, this mechanism is based on a very general view of the governance of coalitions, assuming that each coalition C is governed by a global policy P_C, and that each coalition member E_i is governed by its own policy P_i, which must conform to P_C.

Second, since it is based on LGI, our mechanism can support a wide range of highly dynamic (stateful) policies, and it enforces these policies in an efficient and decentralized manner.

Finally, our mechanism satisfies the following *principle of flexibility*:

> Each local policy P_i can be defined and changed independently of the local policies of other coalition members, and without any knowledge of them.

Such a flexibility is important for several reasons. First, it provides each enterprise with the *autonomy* to define its own policy at will, subject only to the pre-agreed coalition policy P_C. Second, the mutual independence of the local policies of member enterprises simplifies their formation and their evolution. Finally, this principle allows an individual enterprise to keep its own policy *confidential*, since policies of other enterprises do not depend on it.

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*, pages 116–127, May 2001.

[3] X. Ao, N.H. Minsky, and T.D. Nguyen. A hierarchical policy specification language, and enforcement mechanism, for governing digital enterprises. In *Proc. of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, Monterey, California*, pages 38–49, June 2002.

[4] A. Belokosztolszki and K. Moody. Meta-policies for distributed role-based access control systems. In *Proc. of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, Monterey, California*, pages 106–115, June 2002.

[5] C. Bidan and V. Issarny. Dealing with multi-policy security in large open distributed systems. In *Proceedings of 5th European Symposium on Research in Computer Security*, pages 51–66, September 1998.

[6] W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, 1981.

[7] V. Gligor, H. Khurana, R. Koleva, V. Bharadwaj, and J. Baras. On the negotiation of access control policies. In *Proc. of the Security Protocols Workshop, Cambridge, UK*, April 2001.

[8] L. Gong and X. Qian. Computational issues in secure interoperation. *IEEE Transctions on Software Engineering*, pages 43–52, January 1996.

[9] 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).

[10] H. Khurana, V. Gligor, and J. Linn. Reasoning about joint administration of access policies for coalition resources. In *Proc. of IEEE Int. Conf. On Distr. Computing (ICDCS),Vienna, Austria*, pages 429–, July 2002.

[11] P. McDaniel and A. Prakash. Methods and limitations of security policy reconciliation. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 66–80, May 2002.

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

[13] N.H. Minsky and V. Ungureanu. Unified support for heterogeneous security policies in distributed systems. In *7th USENIX Security Symposium*, January 1998. (available from `http://www.cs.rutgers.edu/~minsky/pubs.html`).

[14] 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.

[15] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Proc. of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, Monterey, California*, pages 50–59, June 2002.

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

[17] D. Shands, R. Yee, J. Jacobs, and E. Sebes. Secure virtual enclaves: Supporting coalition use of distributed application technologies. In *Proc. of Network and Distributed System Security Symposium, San Diego, California*, Feb 2000.

[18] M. Thomson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based access control for widely distrbuted resources. In *Proceedings of 8th USENIX Security Symposium*, August 1999.