Establishing Global Properties of Multi-Agent Systems via Local Laws

Wenxuan Zhang, Constantin Serban and Naftaly Minsky

Department of Computer Science,
Rutgers University, Piscataway, NJ 08854, USA
`wzhang,serban,minsky@cs.rutgers.edu`

Abstract. This paper is part of a long term research program on multi-agent systems (MASs), based on the proposition that the interactions among the members of a large and heterogeneous system of autonomous agents need to be governed by a global and strictly enforced *law*; and that such laws need to be *local*, so that they can be complied with at the locus of each agent—without having any direct information of the coincidental state of other members of the MAS. Such concept of law has been realized under our LGI coordination and control mechanism. This paper shows how local laws over a MAS can be used to establish global and *aggregate* system properties in a scalable manner; where by "aggregate properties" we mean properties defined over the coincidental interactions among several, possibly many, members of a given multi-agent system.

1 Introduction

This paper is part of a long term research program on multi-agent systems (MASs), based on the proposition that the interactions among the members of a large and heterogeneous system of autonomous agents—whether software agents or people—need to be governed by a global, overarching, *law*; and on the proposition that such a law needs to be enforced, and the enforcement should be done in a decentralized manner, for scalability.

As we have shown in [1], decentralized enforcement of a law of a distributed MAS requires the law to be formulated in a *local* manner, so that it can be complied with at the locus of each agent—without having any direct information of the coincidental state of other members of the MAS. Given that the law of a MAS is to be uniformly enforced over all the members of the MAS, it obviously has global consequences, despite its local nature. This is analogous to the local (differential) laws of physics, which have global consequences such as conservation of energy.

But can a local law be used to establish *aggregate* system properties, defined over the coincidental interactions among several, possibly many, members of a given multi-agent system. This is the question addressed in this paper.

Theoretically, the answer to this question is affirmative. In fact, we have shown in [2] that the local formulation of laws does not reduce their expressive power. But this result has been derived via a construction that adds an extra agent to a given MAS, which is employed as a centralized *reference monitor* that mediates all interactions between the members of the MAS, and can thus enforce arbitrary aggregate properties. Unfortunately, while formally local, the resulting law would not be scalable, defeating the purpose of localization. This makes the theoretical proved generality of local laws less than satisfying.

This paper is a more practical study of the effective expressive power of local laws. We will demonstrate here that at least some—and we believe many, if not most—aggregate properties of a MAS can be implemented via strictly local laws, involving only a minor degree of centralization, whose effect on the scalability of law enforcement is negligible.

We couch our discussion of this issue in terms of the coordination and control mechanism for multi-agent systems called *law-governed interaction* (LGI), which has been designed according to the above stated propositions. This mechanism has been introduced in 1991 [1], implemented experimentally in 1995 [3]; and after several extensions, such as in [2], it has been released for public use in [4].

The rest of this paper is organized as follows. We start in Sect. 2 with a motivating example of a policy that involves aggregate properties. In Sect. 3 we provide an overview of the LGI mechanism. In Sect. 4 we show how the aggregate properties of our example can be established by means of an LGI law. This is followed with brief discussion of related works in Sect. 5, and with conclusion in Sect. 6.

2 An Example

Consider a large distributed enterprise E, which spans a large geographical area. Suppose that the management of this enterprise decided to provide its employees with the ability to conduct confidential and orderly discussions among themselves, free from any danger of intervention or eavesdropping by the management. For this purpose, a policy called CD (for "confidential discussion") has been defined, which is to govern all such discussion groups, to be called CD-communities. The policy CD is stated informally below:

1. *Only employees of enterprise E who do not belong to management are permitted to be members of a CD-community.*
2. *The members of a given CD-community address each other via their self-chosen aliases, and members cannot infer the $eName$ of its peers, or their LGI-addresses, from their aliases.*
3. *The alias chosen by a member of a CD-community must be unique, in the following senses: (a) no two community members can have the same alias; and (b) each employee can choose just one alias, preventing a single employee from participating under two different aliases.*

4. *Each community member should have access to the entire membership of his community (that is, to the entire list of aliases) at any given moment in time.*

Points 1 of this policy ensures that people not employed by the given enterprise, or people employed by the enterprise as managers, cannot participate in any CD-community, nor can they eavesdrop on any discussion within such a community. Note that this point involves a subtle sensitivity to the environment in which this community operates, sensitivity of the general kind advocated in [5] (we will support this particular sensitivity by means of digital certification). Point 2 ensures anonymity of the participants in any given CD-community, and thus personal confidentiality. Finally, Points 3 and 4, ensure a reasonable order in the discussion conducted by the members of a given CD-community. (All told, this is a minimalistic policy, which, as we shall see later, can be used as a basis over which more sophisticated policies can be defined).

Note that this is a global policy, with some inherently aggregate provisions, such as Point 3 of this policy that requires uniqueness of member aliases, and Point 4 which requires access of each member to the total community membership. This policy can be easily enforced via a central regulator that mediates all exchange of messages between members of an CD-community. But we will show that it can be done in a virtually decentralized, and thus scalable, manner, by specifying policy CD via a local LGI law.

Finally, we point out that that this is not just a toy example, as it deals with issues that appear frequently in multi-agent systems. Some broader perspectives over this example are discussed in Sect. 4.1.

3 An Overview of LGI

Law-Governed Interaction (LGI) is a message-exchange mechanism that allows an open and heterogeneous group of distributed *actors* to engage in a mode of interaction *governed* by an explicitly specified and strictly enforced policy, called the "law" of this group. By "actor" we mean an arbitrary process, whose structure and behavior is left unspecified. An actor engaged in an LGI-regulated interaction, under a law $\mathcal{L}$, is called an $\mathcal{L}$-*agent* (or simply an "agent," when the identity of the law does not matter); the messages exchanged under a given law $\mathcal{L}$ are called $\mathcal{L}$-messages; and the group of agents interacting via $\mathcal{L}$-messages is called an $\mathcal{L}$-community. LGI turns a set of disparate actors, which may not know or trust each other, into a *community* of agents that can rely on each other to comply with the given law $\mathcal{L}$. This is done via a distributed collection of generic components called *private controllers*, one per $\mathcal{L}$-agent, which need to be *trusted* to mediate all interactions between these agents, subject to a specified law $\mathcal{L}$ (as illustrated in Figure 1).

A prototype of LGI was released in October 2005 [4]; this section provides only a very brief overview of LGI. For more information, the reader is referred to the LGI tutorial and manual, available through the above mentioned website, and to a host of published papers.

The Concept of Law Under LGI: LGI laws are formulated in terms of three elements, called: *regulated events*, *control-state*, and *primitive operations*—which are defined in the context of each agent operating under LGI. Only an abstract description of these elements is provided here.

Regulated Events (or, simply, *events*) constitute the *domain* of LGI laws. They are the local events that may occur at an individual agent (called the *home* of the event at hand), whose disposition is governed by the law under which this agent operates. All regulated events are related to inter-agent interactions. They include *arrived* events, which represent the arrival at the home agent of a message from the outside; and *sent* events, which represent the attempt by the home agent to send a message. There are additional regulated events whose relevance to interaction is less direct. One of them is the *adopted* event, which represents the *birth* of an LGI agent—more specifically, this event represents the point in time when an actor adopts a given law $\mathcal{L}$ to operate under, thus becoming an $\mathcal{L}$-agent.

Control-State (or, simply, *state*) of a given LGI agent represents a function of the history of its interaction with other LGI agents. This function, mapping history of interaction to a state, is defined by a specific law. For example, if the number of messages already sent by a agent is somehow relevant to the law under which it operates, then this law would have to mandate maintaining this number as part of its state. That is, the semantics of the control-state is not universal, but is defined by a specific law.

Primitive Operations (or, simply, *operations*) are the actions that can be mandated by a law, to be carried out in response to the occurrence of a given regulated event. These operations can be classified into two groups. First, there are *communication-operations* that affect message exchange between the home-actor and others. These include the $forward$ operation that forwards a message to another agent, and the *deliver* operation that allows the home-actor to actually receive a message that arrived on its behalf. Second, there are the *state-operations* that affect the state of the home-agent. These, and other operations to be introduced later, are called "primitive" because they are meant to be carried out *if and only if* they are mandated by the law.

The role of a law $\mathcal{L}$ under LGI is to decide what should be done if a given event e occurs at an agent x operating under this law, when the control-state of x is s. This decision, which is called the *ruling of the law*, can be represented by the sequence of primitive operations mandated by the law, to be carried out, atomically, at x. More formally, the concept of law can be defined as follows:

Let E be the set of all possible regulated-events, let S be the set of all possible states, and let O be the set of all primitive operations, then a law $\mathcal{L}$ is a function:

$$\mathcal{L} : E \times S \rightarrow O^* \tag{1}$$

In other words, *the law maps every possible* $(event, state)$ *pair into a sequence of primitive operations, which constitute the* ruling *of the law.*

Several observation about this definition are in order: First, this definition does not specify any mechanism for enforcing LGI-laws, and does not even require enforcement. Indeed, the concept of law under LGI, like the concept of social law, is quite meaningful even if one leaves it up to individual agents to comply with it voluntarily. In our case such compliance means, in particular, that every agent subject to a law $\mathcal{L}$ carries out the ruling of this law for every regulated event that occurs in it. However, LGI does provide an enforcement mechanism for its laws.

Second, the above definition of the concept of law is *abstract*, in that it does not depend on the language used for specifying the function that constitutes a given law. This level of abstraction is useful for two reasons. First, it allows one to understand the basic properties of LGI, independently of the complexities of the language used for specifying its laws. Second, this abstraction provides LGI with a useful flexibility regarding the language actually used for specifying laws. In particular, it allows LGI to support multiple *law-languages*, while maintaining essentially the same semantics. Indeed, the current implementation of LGI supports two law-languages, based, respectively, on Prolog and on Java. In this paper, however, we will use an informal pseudo-code for describing laws, to be introduced later.

Finally, note that, as stated in the introduction, the law as defined above is *local*, so that it can be complied with at the locus of each agent—without having any direct information of the coincidental state of other members of the MAS. This is because all the elements over which the law is defined—namely, the events, the operations and the state—are all defined locally at each agent.

An Informal Language for Specifying Laws: In this paper laws will be described via a pseudo-code consisting of *event-condition-action* rules, which is similar to the formal Prolog-based law language of LGI.

The *event-condition-action* rules that constitute the pseudo-code have the form:

```
UPON e IF c DO [o],
```

where `e` is an event; `c` is an optional condition, defined over the event itself, and over the state of the home-agent; and `[o]`, the action, is a list of one or more primitive operations, which constitute the ruling of the law.

The Decentralized Law-Enforcement of LGI: The enforcement of a given law over the community is carried out by a distributed set $\{T_x | x \in C\}$ of *controllers*, one for each member of community 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 C_L, however, they all carry the *same law* $\mathcal{L}$. And each controller T_x associated with an agent x of this community carries only the *local control-state* CSx of x, while every $\mathcal{L}$-message exchanged between a pair of agents x and y passes through a pair of controllers, T_x and T_y (see Fig 1).

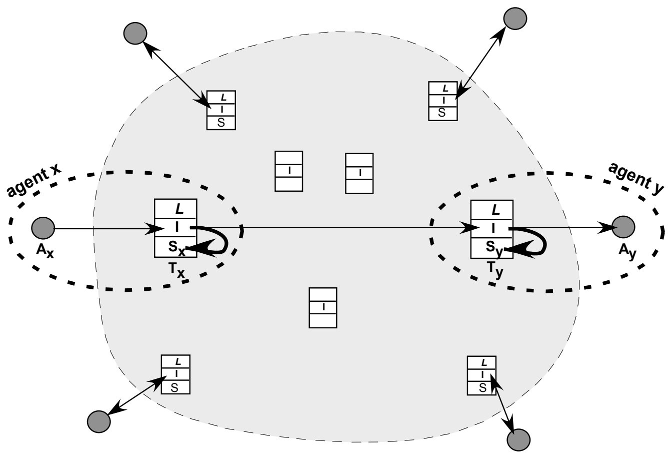

Fig. 1. Interaction via LGI: Actors are depicted by circles, interacting across the Internet (lightly shaded cloud) via their private controllers (boxes) operating under law L. Agents are depicted by dashed ovals that enclose (actor, controller) pairs. Thin arrows represent messages, and thick arrows represent modification of state.

Due to the local nature of LGI laws, each controller 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 T_x handles the events at x strictly sequentially, in the order of their occurrence, and atomically. Finally we point out that the law-enforcement mechanism ensures that a message received under law L has been sent under the same law; i.e., that it is not possible to forge $\mathcal{L}$-messages. As described in [6], this is assured by the following: (a) The exchange of $\mathcal{L}$-messages is mediated by correctly implemented controllers, certified by a CA specified by law $\mathcal{L}$; (b) these controllers are interpreting the *same law* $\mathcal{L}$, identified by a one-way hash H of law $\mathcal{L}$; and (c) $\mathcal{L}$-messages are transmitted over cryptographically secured channels between such controllers. Consequently, how each member x gets the text of law $\mathcal{L}$ is irrelevant to the assurance that all members of C_L operate under the same law.

The creation of LGI-agents, and their naming Given a controller T, an actor A may generate a new $\mathcal{L}$-agent by sending what is called an *adoption* message to T, thus adopting it for operating its controller, under a specified law $\mathcal{L}$. In response, T would create a new controller, subject to law $\mathcal{L}$, identifying it by a local name n (unique among the names given to the other private controllers already operating on T).

This new controller, and the agent it represents, are henceforth known by the name `‘‘n@dName(T)’’` where `dName(T)` is the domain-name of the controller T, such as `‘‘ramses.rutgers.edu’’`. This name—for example joe@ramses.rutgers.edu—is the *LGI address* of the newly formed agent, to be used by other agents for communicating with it.

The Hierarchical Organization of Laws LGI provides for laws to be organized into hierarchies, a facility designed for the modularization of complex laws, and to support such things as coalitions of institutions, and complex organizations such as enterprises. A hierarchy, or tree, of laws $t(\mathcal{L}_0)$, is rooted in some law $\mathcal{L}_0$. Each law in $t(\mathcal{L}_0)$ is said to be (transitively) subordinate to its parent, and (transitively) superior to its descendants. Given a pair of laws $\mathcal{N}$ and $\mathcal{M}$ in $t(\mathcal{L}_0)$, we write $\mathcal{N} \prec \mathcal{M}$ if $\mathcal{N}$ is subordinate to $\mathcal{M}$.

Semantically, the most important aspect of this hierarchy is that if $\mathcal{N} \prec \mathcal{M}$ then $\mathcal{N}$ *conforms* to $\mathcal{M}$, in the sense that law $\mathcal{N}$ satisfies all the stipulation of its superior law $\mathcal{M}$.

LGI provides a very efficient mechanism, outlined in [7], for constructing such law-trees, in a top-down manner. This is done as follows. The root $\mathcal{L}_0$ of a hierarchy is a normal LGI law, except that we create it to be *open* for refinements, by: (a) allowing it to consult a collection of rules designed to refine it—called a *delta*; and (b) taking the advice returned by this delta into account, when computing its ruling. The refinement of law $\mathcal{L}$ via the delta produces a regular LGI law $\mathcal{L}'$. $\mathcal{L}'$ could be closed to further refinements—which produces a hierarchy of depth two; or $\mathcal{L}'$ could further consult other deltas at a lower level, thus producing a cascade of refinements, and a hierarchy of arbitrary depth. In brief, 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* $\bar{\mathcal{L}}'$, a collection of rules[1].

4 A Virtually Decentralized Implementation of the *CD* Policy

We describe here a law $\mathcal{L}_{CD}$ that implements the informally stated policy CD introduced in Sect. 2. For simplicity, this law is written in our pseudo-code language. This law is also overly simplistic, in that it does not handle exceptions, which is important to do when dealing with message passing, and for which LGI provides ample tools, and it is missing certain minor details, as we point out later. However, a completed version of this law, written in our executable Java-based law-language, is published via
`http://www.moses.rutgers.edu/lcd1/Lcd.java1`.

The $\mathcal{L}_{CD}$ law is written under the assumption that the enterprise E in question employs a certification authority (CA), called eCA, which issues identity-certificates to its employees. Each such certificate is supposed to authenticate

[1] This is somewhat analogous to inheritance of classes, except of the strict constraint of conformance between a superior law and its subordinates.

an employee, identifying his official name in the enterprise, called his $eName$, which we assume to be unique; and specifying the position of this employee in the enterprise, such as whether he or she is a manager.

We start with a brief overview of the structure and behavior of the community operating under $\mathcal{L}_{CD}$. First, this community contains a distinguished agent called the *secretary*, denoted by S, which serves both as the registrar of the community, and its name-server. The secretary maintains in its control-state (CS) a set of *member profiles*, each represented by a triple $\langle N, A, L\rangle$, where N is the eName of an employee, A is the alias by which this member is to be known to others in this community, and L is the LGI-address used for communication by the underlying LGI mechanism. The aggregation of these profiles in the CS of S would allow law $\mathcal{L}_{CD}$ to ensure the uniqueness required by Point 3 of our policy.

The other members of this community communicate with the secretary S, mostly for two purposes: (a) to register with it, thus becoming an *active* member of the community; and (b) to get from S the *ID* of other community members, where the *ID* is a pair $\langle A, L\rangle$, which is the member-profile of that member, as maintained by S , without its $eName$. Each community member maintains in its control-state a set (cache) of such IDs, called the *acquaintance list* (or *aList*) of this agent. This cache maps aliases, used for explicit addressing of members under this law, into the the LGI-addresses used by the underlying LGI mechanism. We will see later how the aList is populated, but as long as one communicates with members whose alias exists in one's aList, the communication is direct, and does not involve the secretary.

We will now discuss the operations of the *CD*-community in greater details, showing how it is governed by law $\mathcal{L}_{CD}$. This discussion is organized into a sequence of short paragraphs dealing with different aspects of this community, such as: joining the community, interacting with peer agents, and leaving the community. But we start with law $\mathcal{L}_{CD}$ itself.

Law $\mathcal{L}_{CD}$ that Governs Confidential Discussion Communities: Like all LGI laws, law $\mathcal{L}_{CD}$, displayed in Figure 2, consists of two parts: the *preamble*, and the *body*. The preamble is a small set of declarative clauses, which specify such things as: the name of the law ("CD," in this case); the language in which the law is written (not specified here, but could be either Prolog or Java); one or more trusted CAs, identified by their name under this law, and by their public keys (this is done for "eCA," in this case); and some aliases, used to simplify notations[2] (in this case, the alias "secretary" for the LGI-address of the agent that serves this role).

The body of the law is its algorithmic part. In this paper, the body is described by a sequence of numbered, and informally stated, event-condition-action rules, as defined in Sect. 3. Each of these rules is followed by a comment, in italic. These rules are executed by the controller associated with each agent, whenever a regulated event occurs at this agent; these rules are executed sequentially,

[2] Note that the keyword "alias" here is not the "alias" used elsewhere in the paper.

```
Preambles:
law(CD)    The name of the law
authority(eCA, keyHash(hash−of−key−of−eCA))    The CA trusted by the enterprise
alias(secretary, secretary@rutgers.edu)        The address of the secretary

R1. UPON adopted(issuer, subject, attributes([entName:eName;position:position]))
            IF (issuer = eCA) and (position =/= manager) DO [myEName  = eName]
            ELSE self−destruct
            END IF
The home agent must be authenticated upon adoption in order to join the community.

R2. UPON sent(source, join(eName,alias), secretary)
            IF (myEName = eName) DO [forward]
The home agent applies to be an active member,
by sending a message including its eName and a proposed alias to the secretary.

R3. UPON arrived(secretary, activate(alias), dest)
             DO [deliver; active = true; myAlias = alias]
The home agent is activated to be an active member.

R4. UPON sent(source, getID(alias), secretary)
            IF (active = true) DO [forward]
The home agent looks up the ID of another agent, by its alias.

R5. UPON arrived(secretary, id(alias, LGIAddress), dest)
              IF (active = true) DO [save id(alias,LGIAddress) in aList]
The home agent gets the ID from the secretary, and saves it in the local aList.

R6. UPON sent(source, M, destAlias)
            IF (active = true)
                 destLGIAddress = getAddressFromAList(destAlias)
                 IF (destLGIAddress is not null)
                       enhancedMessage = msg(source(myAlias), dest(destAlias), M)
                       DO[forward(source, enhancedMessage, destLGIAddress)]
                 END IF
            END IF
When an actor attempts to send a message to another agent,
the controller would forward the message only if the home agent is an active member,
and it would piggyback the aliases of both the sender and the receiver on the message.

R7. UPON arrived(source, msg(source(sourceAlias),dest(destAlias),M), dest)
            IF (active = true) and (myAlias = destAlias)
                DO[deliver(sourceAlias,M,destAlias)]
                IF (the ID of the source agent is not in aList)
                     update aList
                END IF
            ELSE
                return an exception message to the source agent
            END IF
When the message arrives at the home agent, the controller would deliver the message
only if the home agent is an active member, and is the intended destination;
it would update the local aList if the ID of the source agent is not included.

R8. UPON sent(source, quit(eName), secretary)
            IF (myEName = eName) and (active = true) DO [forward; quit]
The home agent leaves the community.
```

Fig. 2. A fragment of $\mathcal{L}_{CD}$

from top to bottom. The rules of this particular law are discussed in some detail below.

Joining the** CD **community: Two steps are required for an employee e to join a CD-community, and be able to communicate with its other members. The first step would create a new $\mathcal{L}_{CD}$-agent whose control-state contains the authenticated eName of its actor, provided that this actor is a normal employee of enterprise E, and not a manager. But this new agent is *inactive*, as it cannot communicate with other members of the CD-community, except the secretary S. The second step, if successful, would activate the agent in question, by providing it with an official *alias*, and allowing it to communicate with its peers.

The first step is to select an LGI-controller, and have it adopt law $\mathcal{L}_{CD}$—thus creating a new $\mathcal{L}_{CD}$-agent, which we call here x. The first event in the life of the new agent is the *adopted* event, handled by Rule `R1` of $\mathcal{L}_{CD}$. This rule requires the adoption message sent by e to its controller to contain a certificate signed by eCA, and this certificate is required to contain two attributes: `position` and `entName`.

Now, if the value of the `position` attribute in the submitted certificate is not `manager` then the value of the `entName` attribute is assigned to the `myEName` variable of the CS of x. But if the position attribute of the certificate indicates that this employee is a manager, then Rule `R1` will cause an exception message to be sent to employee e (not shown in Figure 2), and the newly formed LGI-agent would self destruct. This is in conformance with Point 1 of the CD policy, which allows only non-management employees to participate in this community.

At this point the newly created agent x is *inactive*, in the sense that it cannot do anything but send a message `join(eName,alias)` to the secretary S—which is the second step of joining an $\mathcal{L}_{CD}$-community. The event of sending the `join` message is handled by Rule `R2` of $\mathcal{L}_{CD}$, which ensures that the first argument of this message is identical to the variable `myEName`, and is thus the authenticated $eName$ of the employee in question.

Note that the rules of law $\mathcal{L}_{CD}$ that deal with the arrival of messages (such as `join`) at S, and with the responses of S to such messages, are not shown in Fig. 2, for simplicity[3] But the effect of sending the `join` message to S is as follows: when this message arrives at S, S would check that both the `eName` and the `alias` are unique in the CD community. Note that these condition—required by Point 3 of the CD policy—are checked with respect to the set of member-profiles, of all active community members, maintained by the secretary. If this condition is satisfied, S would send the message `activate(alias)` to x.

When a message `activate(alias)`, sent by S, arrives at x, it would be handled by Rule `R3`. This rule would save the value of `alias` in a variable `myAlias` in the CS of x; and it would set a variable `active` in the CS of x to be *true*. This would make x a fully active member of the CD-community, as we shall see below.

[3] Note again that the complete law $\mathcal{L}_{CD}$ is published through `http://www.moses.rutgers.edu/lcd1/Lcd.java1`.

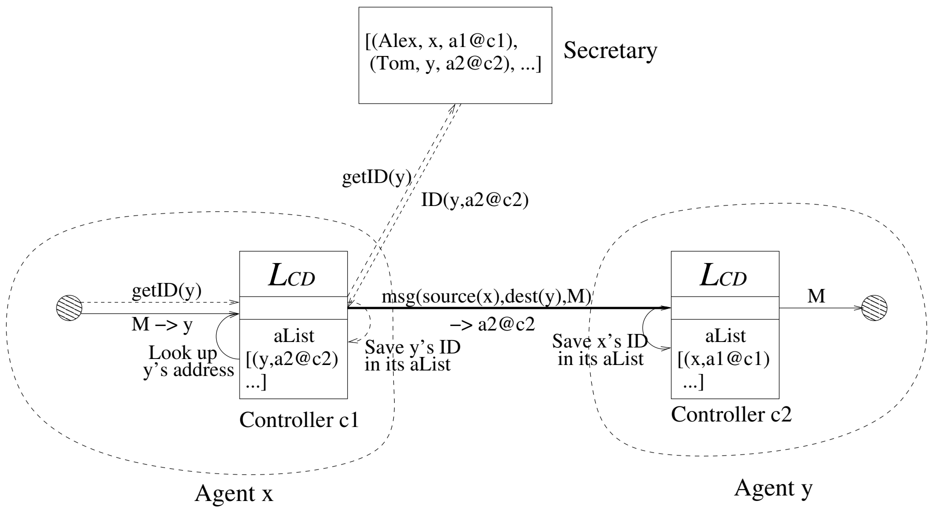

Fig. 3. Communication between two active members

Populating the Acquaintance List (aList): As has already been pointed out, for a member x to be able to send a message to another member y, it needs to have its ID $\langle alias, LGIaddress\rangle$ in its $aList$, which serves the role of an addressing cache. This cache is populated in two ways:

First, by requesting, and obtaining, an ID, or a whole set of them, from the secretary. As depicted in Fig. 3, an active member x can send S a message `getID(y)`, where `y` is an alias of the member whose ID is being requested. The sending of this message is handled by Rule `R4`, which forwards it to S, provided that x is an active member. The secretary will reply by sending x the requested ID , if any. When this reply arrives at x, it would be handled by Rule `R5`, which would store the new ID in the CS of x. In a similar fashion x can ask for the set of IDs of all current members of the community (but this capability is not shown in Figure 2).

The second way for the ID of y to be added to the $aList$ of x, is for y to send any message to x. We will see how this is done next.

Communication Between the Members of a CD-Community: Exchange of messages among members of the CD-community are regulated by Rules `R6` and `R7`. By `R6`, a message `M` sent to `destAlias` would not be forwarded to anybody if (a) if the sender is not an active member, or (b) if the sender does not have an ID $\langle destAlias, L\rangle$ in its $aList$. However, if both of these conditions are not satisfied; that is, if the sender is an active member, and if it does have the right ID, then the following message would be forwarded to the LGI-address `L` associated with the `destAlias`:

```
msg(source(myAlias),dest(destAlias),M).
```

Note that this message carries the aliases of the sender and of the target, along with the original message `M`—it is called an "enhanced message".

By Rule `R7`, when the enhanced message arrives at its target y the message `M` carried by it is delivered to its actor only if (a) y is an active agent, and if (b) y is the agent identified by the `destAlias` carried in the enhanced message. Also, if the *aList* of y does not contain the *ID* of the sender, then this *ID* will be added to it. However, if either of these conditions is not satisfied, then an appropriate exception message would be sent to the sender. We will not elaborate here on the various possible reasons for the above to conditions to fail. But one of them is that an employee may change the LGI-address from which it operates. More about this possibility below.

Migration of Agents: An important advantage of symbolic addressing via aliases, is that it abstracts out the actual IP-address from which one operates. This allows an employee to migrate, from one computer (and controller) to another, without requiring any change in how he is addressed by others. But, such migration requires the member profile maintained by the secretary to be updated. This is done as follows:

After an agent moves from one controller to another, its actor must submit its certificate again, to the new controller. If the certificate is valid, the agent would be authenticated as an enterprise employee and the controller would be able to obtain its eName. Then the agent would inform S about the address changing, by sending a message `updateAddress(myEName)`. S would be able to locate the ID of the agent by the eName and thus update its LGI address. Also, S would reply a message to the agent, in order to activate it to be an active member, and to provide the alias it registered before.

Quitting the** CD **Community: An active member may remove itself from the *CD* community at will. It does this by sending a message `quit(eName)` to S, who will identify this member by its `eName`, then remove it from the list of active members.

By Rule `R8`, when sending the message, the law ensures that (1) `eName` has been bound to the variable `myEName`, the certified eName of the home agent, and (2) the value of the variable `active` is true, which means the home agent is an active member of the *CD* community. The home agent will quit the community by executing the `quit` operation, after the message is sent.

4.1 Broader Perspectives

Law $\mathcal{L}_{CD}$ is only a special case of a class of laws that can be used in a wide range of applications. We will mention here two types of such applications, both of which employ the hierarchical organization of laws provided by LGI, and briefly discussed in Sect. 3.

Confidential Discussion Groups Operating under Different Rules of Engagement: Elaborating on the motivation given in Sect. 2 for the CD policy, suppose that different groups of employees in enterprise E would like to operate under different kinds of rules of engagement, while conforming to the CD policy mandated by the enterprise. One group may want to restrict its members to a specific department, another group may want to establish a version of the Robert's Rules of Order suitable for electronic discussion, and a third group may want to support some kind of secure voting protocol.

This can be done by changing law $\mathcal{L}_{CD}$ into an equivalent law $\mathcal{L}_{CD}$' that admits refinements. The above mentioned refinements can then be defined as subordinate laws to $\mathcal{L}_{CD}$', which would thus be guaranteed to conform to the enterprise mandated CD policy. The mechanism for creating such refinements is beyond the scope of this paper, but the interested reader can find the necessary details in [7].

Symbolic Addressing: The symbolic addressing via aliases provided by law $\mathcal{L}_{CD}$ could be useful in general, and not just in the context of an enterprise. This is because the LGI-addressing is dependent on the absolute IP-address of the the controller being used by a given actor, and is difficult to maintain invariant of the location of the actors itself, which may be mobile. One can provide for symbolic addressing, by removing from $\mathcal{L}_{CD}$' the part that requires authentication via certificate signed by a specified CA, but allowing it to be further refined, as discussed above. This would allow the creation of arbitrary laws that conform to the symbolic addressing capability of $\mathcal{L}_{CD}$'.

5 Related Work

The LGI coordination and control mechanism for multi-agent systems has been introduced in 1991 [1]. In the years following this work, several authors considered the role of laws in multi-agent systems. Some of these, like [8]and [9], view a law of a MAS as purely a specification device, without any enforcement mechanism. Others, such as [10], [11] and [12], did consider enforcement, but not in a decentralized manner. Moreover, none of these authors used local laws, which we consider essential to any coordination and control mechanism for multi-agent systems.

The literature regarding name services, which is the subject of the specific example used in this paper, is very rich. Suffice it to mention the most prominent name service, in current Internet infrastructure, the Domain Name System (DNS) [13]. But most of the standard name services, including DNS, do not provide any control over the community it serves, which is the main advantage of our approach to this issue.

6 Conclusion

The main objective of this paper has been to demonstrate that a regulatory mechanism for agent-based systems, which is based on strictly local laws, can be used to establish globally aggregate system properties with only minor effect on scalability.

In conclusion, we note that although this paper has bean couched in term of the LGI mechanism that enforces laws, its implication are not limited to LGI. Indeed, as we have pointed out, the concept of LGI law can be used for multi-agent systems, even if one leaves it up to individual agents to comply with the given law voluntarily. Since voluntary compliance also requires the law to be local, the ability of such laws to establish aggregate properties is important in this context as well.

References

1. Minsky, N.H.: The imposition of protocols over open distributed systems. IEEE Transactions on Software Engineering (February 1991)
2. Minsky, N.H., Ungureanu, V.: Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. TOSEM, ACM Transactions on Software Engineering and Methodology **9**(3) (July 2000) 273–305 (available from `http://www.cs.rutgers.edu/~minsky/pubs.html`).
3. Minsky, N.H., Leichter, J.: Law-governed Linda as a coordination model. In Ciancarini, P., Nierstrasz, O., Yonezawa, A., eds.: Object-Based Models and Languages for Concurrent Systems. Number 924 in Lecture Notes in Computer Science. Springer-Verlag (1995) 125–146
4. Serban, C., Minsky, N.H.: The lgi web-site (includes the implementation of lgi, and its manual). Technical report, Rutgers University (June 2005) (available at `http://www.moses.rutgers.edu`).
5. Weyns, D., Omicini, A., Odell, J.: Environment as a first class abstraction in multiagent systems. Journal on Autonomous Agents and Multiagent Systems **14**(1) (January 2007)
6. Minsky, N.H.: Law Governed Interaction (LGI): A Distributed Coordination and Control Mechanism (An Introduction, and a Reference Manual). (June 2005) (available at `http://www.moses.rutgers.edu/documentation/manual.pdf`).
7. Ao, X., Minsky, N.H.: Flexible regulation of distributed coalitions. In: LNCS 2808: the Proc. of the European Symposium on Research in Computer Security (ESORICS) 2003. (October 2003) (available from `http://www.cs.rutgers.edu/~minsky/pubs.html`).
8. Shoham, Y., Tennenholz, M.: On the synthesis of useful social laws for artificial agents societies. In: Proceedings of AAAI-92. (1992)
9. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: the gaia methodology. ACM Transactions on Software Engineering and Methodology (TOSEM) (July 2003)
10. Esteva, M., de la Cruz, D., Sierra, C.: Islander: an electronic institutions editor. In: Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). (2002)
11. Gatti, A., Lucena, C., Brio, J.: On fault tolerance in law-governed multi-agent systems. In: Proc. of ICSE'05, 4th Int. Workshop on Soft. Eng. for Large-Scale Multi-Agent Systems, ACM (July 2006)
12. Artikis, A., Sergot, M., Pitt, J.: Specifying norm-governed computational societies. Technical report, Department of Computing, Imperial College of Science Technology and Medicine, London (2006)

13. Mockapetris, P.: Domain names - concepts and facilities. (available at ftp://ftp.is.co.za/rfc/rfc1034.txt) (Nov. 1987)