On Shouting "Fire!": Regulating Decoupled Communication in Distributed Systems*

Takahiro Murata and Naftaly H. Minsky
Department of Computer Science
Rutgers University
Piscataway, NJ, 08854 USA
E-mail: {murata|minsky}@cs.rutgers.edu

December 26, 2002

Abstract

Decoupled communication, which requires no direct association between the producers of information and its consumers—as under the *publish/subscribe* (P/S) middleware —is often useful for the integration of distributed and heterogeneous applications. But the indefinite, and potentially global, reach of decoupled communication—the very reason for its power—has a dark side, which may complicate the system using it, making it less predictable, more brittle, and less safe. Just think about the effect of shouting "fire" in a packed theatre, particularly, but not only, if it is a false alarm.

It is our thesis that the inherent drawbacks of decoupled communication can be alleviated by decentralized regulation of its use. We show how such regulation can be carried out scalably by means of a distributed control mechanism called Law-Governed Interaction (LGI), and a middleware called Moses that implements this mechanism. And we illustrate the importance of such regulation, and its effectiveness, by considering the treatment of alarms in a large hospital.

Keywords

Decoupled communication, middleware, publish/subscribe, regulation, scalability, law-governed interaction.

1 Introduction

The issue addressed in this paper is that of communication in large heterogeneous distributed systems, under which, the communicators may not know the location, identity, or even the presence, of those they communicate with. Among several techniques that have been developed to support such communication—which effectively *decouples* the producers of information from its consumers, both in time and in space [6]—the most prominent are: the *publish/subscribe* (P/S, for short) [7, 15], and the Linda-like *tuple-space* [5, 13] middlewares. The publish/subscribe paradigm, which we will take here as a representative of what enables decoupled communication, provides *mediators* between the producers of information (which we call here *informers*), and its consumers (the *clients*, or *subscribers*). Under this paradigm, when an informer has some information to impart, it sends

*Work supported in part by NSF grants Nos. CCR-97-10575 and CCR-98-03698

it to the mediator[1]—an act called *publishing*. The mediator, in turn, would communicate the published information (often called an *event-notice*) to all its clients who previously expressed interest in it by *subscribing* to a certain kind of event-notices.

As an example of decoupled communication, consider a large hospital, where various kinds of emergency situation may arise, such as: fire, low level of blood supply, a blackout in some regions of the facility, etc. When such an emergency is discovered by some agent—which may be an employee, a patient, a visitor, or some system component—it needs to be communicated to the various agents that might be concerned with it, whose identity, or its very existence, is likely not to be known to the discoverer of the emergency. Such communication can be carried out effectively via a P/S service, which allows the discoverer of an emergency to raise an alarm by publishing it, and allows anybody concerned with a given type of alarms to subscribe to it.

But the indefinite, and potentially global, reach of decoupled communication—the very reason for its power—has a dark side, which may complicate the system using it, making it less predictable, more brittle, and less safe. For example, raising an alarm that warns of a low level of blood supply may disrupt various routine activities, anywhere in the hospital—and may, therefore, be very harmful, particularly if it is a false alarm. Or, just think about the effect of shouting "fire" in a packed theatre. Even a valid alarm of low blood supply may cause harm by creating unnecessary panic, if it is communicated to the wrong people, like to patients waiting for an operation, or to their relatives. Moreover, an individual agent is in no position to evaluate the effect of an alarm he (or it) is about to publish, since he does not know who, if anybody, may be listening to it; nor will he be able to rely on any feedback from his listeners. So, this mode of communication is conducted, in a sense, in the dark, making its consequences unpredictable, and potentially dangerous.

It is the thesis of this paper that these drawbacks of decoupled communication can be alleviated by a suitable regulation of its use. We will motivate this thesis by presenting, in Section 2, a policy for regulating alarms in a hospital, which specifies, in particular: (a) who is qualified, and under what condition, to raise which kind of alarms; (b) who should be able to, or ought to, subscribe to which alarm; and (c) how should certain agents respond when they receive an alarm. In general, a policy that regulates decoupled communication needs to govern a large, heterogeneous, and indefinite community of agents, which might be dispersed throughout a system. The specification, and the scalable enforcement, of such *communal* policies is the subject of this paper.

The need to regulate decoupled communication—long ignored by the research community—has been recently addressed by some researchers working on publish/subscribe middleware [11, 9], by providing P/S mediators with access-control capabilities. Some commercial P/S mediators, particularly those that implement JMS [8], also provide rudimentary access-control, based on conventional access-control lists (ACLs). However, as we will demonstrate later, policies required to regulate decoupled communication go beyond access-control, and we will contend that it is not appropriate to implement such policies in the servers of such communication mode. Therefore, we take here an alternative approach, which, in the context of the publish/subscribe paradigm, can be characterized as follows: Instead of having the P/S mediator define and enforce an access-control policy over all publications and subscriptions made through it, we have a substantially richer policy defined and enforced directly over the agents attempting to communicate via the P/S mediator. This is done by means of a distributed coordination and control mechanism called Law-Governed Interaction (LGI) [14], and a middleware called Moses that implements this mechanism—an overview of which is provided in Section 3.

[1]We assume here, for simplicity, a single mediator, but, as explained later, the technique of regulation proposed in this paper can be straightforwardly applied to multiple mediators working in concert.

We will also show that the proposed regulatory mechanism, which is entirely decentralized, is more scalable than the centralized access-control built into the P/S mediator, and that it can better protect the mediator from abuse by careless or buggy users. Finally, we will demonstrate that our regulatory mechanism is more powerful than the mediator-based alternative, in particular, in that it is able to: (a) impose obligations on the informers, or on the recipients, of certain event-notices; and (b) ensure the qualification of the mediators themselves.

The rest of this paper is organized as follows. Section 2 is a study of the nature of regulation that may be required over decoupled communication, using the treatment of alarms within a hospital as a case in point. Section 3 is an overview of law-Governed Interaction (LGI)—the computation mechanism on which this paper is based. In Section 4 we show how our example alarm policy of Section 2 is formulated, and scalably enforced under LGI. In Section 5 we consider the performance of our control mechanism; and we conclude in Section 6.

2 An Institutional Alarm Policy—a Motivating Example

We start this section with an elaboration on the alarm example introduced above, by stipulating a detailed policy for the treatment of alarms within a large hospital. We will then argue that this policy cannot be implemented effectively by P/S mediators alone, requiring a degree of control over the community of users of the P/S services.

2.1 An alarm policy for a hospital

Before we present the policy itself, some comments about our assumptions and terminology are in order. First, we will distinguish between various *kinds* of alarms, for emergencies such as: fire, low level of blood supply, etc.—assuming, for simplicity, that the sets of alarms of different kinds are disjoint. Alarms of kind K are called K-alarms. Second, we assume that for each kind K of emergency, there are some designated *experts*, called K-experts, who take the primary responsibility for recognizing the occurrence of this emergency, and for dealing with the associated alarms. Agents that are not thus designated as experts, on a given type of emergency, are called *laymen* with respect to it.

Third, we recognize certain roles that various agents may play in this context. These include: (a) P/S servers, also called *mediators*; (b) the above mentioned *experts* on various kinds of emergencies and alarms; (c) *K-inspectors*, who have special responsibility with respect to K-alarms, to be discussed below; (d) a *facility manager*, whose function is to regulate dynamically various aspects of the system, as we shall see; and (e) a *certification authority* (CA) called admin, whose certification would be required for the authentication of most of the above roles.

Fourth, we assume all alarms to have the following form:

```
alarm(kind(K), time(T), informer(I), status(S), text(X))
```

where, `K` is the kind of the alarm; `T` marks the time when this alarm was raised; `I` identifies the informer who has raised this alarm; `S` specifies the status of this informer—either `expert` or `layman`—with respect to the current alarm kind `K`; and, finally, `X` is the text describing the specific alarm. Finally, regarding subscriptions to alarms, we consider here, for simplicity, only exact matching with a specified list of attribute values; e.g., subscription to

```
alarm([kind(fire), status(expert)])
```

means to capture all fire alarms published by experts.

We now introduce a detailed example of an alarm policy for a hospital, to be called simply AP. The statement of this policy is followed by a discussion of its rationale.

1. *The status of the following roles must be certified by the CA* admin*: the facility manager, a mediator, and a K-expert.*

2. *The publishing of alarms, and subscription to them, must be done via agents duly certified as valid P/S mediators.*

3. *Alarms published by experts are receivable by everyone, while alarms published by laymen are receivable only by experts (Figure 1).*

4. *A layman can publish the same alarm only once within any 5 minute period, and the facility manager can prevent laymen from publishing alarms altogether.*

5. *The facility manager can appoint an expert on alarms of type K as a K-*inspector*. The duty of such an inspector would be to examine all K-alarms raised by laymen, and decide what to do with them; their behavior is governed by the following rules:*

   (a) *Every K-inspector is **obliged to subscribe** to all K-alarms.*

   (b) *Each K-inspector is supposed to acknowledge the receipt of every K-alarm issued by a layman, within 10 minutes after receiving this alarm, by sending the copy of this alarm to the facility manager.*

   (c) *If an inspector failed to acknowledge the receipt of an alarm in a timely manner, as specified above, an alarm of kind* `metaAlarm` ***must be published**, identifying the unacknowledged alarm, and the inspector that failed to acknowledge it. Alarms of kind* `metaAlarm` *are receivable* only *by experts.*

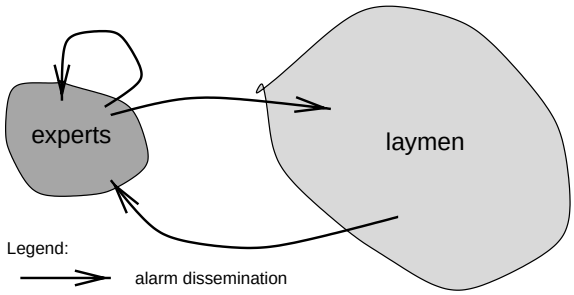

Figure 1: Alarm dissemination between experts and laymen

Rationale: Point 1 of this policy requires the holders of a role—such as a facility manager, an expert on alarms of a specific kind, and, in particular, the P/S mediator—certify themselves via the specified CA. In other words, such certification process reflects the organizational decision as to which agents should play these roles. The certification of mediators, and the requirement of Point 2 that only mediators thus certified be used for the dissemination of alarms, provide some assurance of the quality and trustworthiness of the mediators. This is, in part, a way to deal with concerns such as confidentiality of information handled by the P/S mediators, raised by [19].

Point 3 is concerned with the distinction between experts and laymen. The reason for this distinction is that a widely disseminated alarm can be as dangerous as shouting "fire!" in a crowded

theatre. Therefore, this policy limits the ability to make such alarms to those who are certified as "experts," and presumably able to recognize a true emergency condition that requires the raising of an (real) alarm. Of course, an emergency condition may be first observed by a layman, which is why laymen are allowed to raise alarms. But because laymen's alarms are not very trustworthy, they are visible only by the appropriate experts, who can examine the situation and, if necessary, issue an alarm to the entire client base.

The role of inspector is introduced, by Point 5, in an attempt to ensure that there are some responsible agents that listen to all laymen's alarms, and in a timely fashion. This assurance, which is, of course, not absolute, is achieved as follows. First, once a K-inspector is appointed by a certified facility manager, he is *obliged*, by Point 5a, to subscribe to all K-alarms, so that no such alarms would be left unseen. Second he is expected, by Point 5b, to send a copy of each such alarm to the facility manager, serving here as an auditor, as a proof that he actually noticed it. Third, if such a copy is not sent within a specified deadline—perhaps because this inspector is not attentive, or is disconnected—then, by Point 5c, an appropriate `metaAlarm` is to be raised automatically, in the hope that it will be picked up by the `metaAlarm`-inspector, or by some other expert on such alarms. We believe such monitoring of responses to certain publications is often essential for the reliability of the system composed of potentially unreliable components.

Finally, Point 4 is an attempt to protect the P/S mediator from large numbers of unnecessary alarms—which might be issued by an overzealous, or faulty, layman. This is done by: (a) limiting the frequency of publishing repeated alarms by every layman; and (b) by allowing the facility-manager to prevent certain laymen from publishing alarms altogether.

2.2 On the communal nature of policy AP

We explain here our contention that policy AP—and other such policies, by implication— is inherently communal, governing the entire community involved with alarms, and that it does not lend itself to effective implementation by P/S mediators alone. This contention has several reasons.

First, the mediators (or P/S servers) cannot ensure by themselves that `publish` and `subscribe` messages are sent only to servers duly certified as mediators, as required by Point 2. This, clearly, requires a degree of control over the operation of the agents that publish and subscribe, not allowing them to use uncertified servers.

Second, Point 4 of policy AP limits the frequency of alarms from any layman. As we already pointed out, the purpose of this provision is to reduce congestion on the mediators by protecting them from overzealous alarmists, which may be in a loop sending thousands of alarm notices. This purpose cannot be achieved if the mediator itself has to enforce policy AP, because it might still be congested, just by having to receive, and then reject, all such useless alarms.

Third, besides the client/server interactions between the users and the mediators, the regulation of policy AP ranges also over some interactions between users themselves. This is the case, in particular, for the assignment of K-inspectors for duty, which, under Point 5, is carried out by messages from facility managers to the agents in question. Such messages do not involve the mediators, which are, therefore, in no position to regulate them.

It is, of course, possible to require the assignment messages from managers be sent via the mediator, which would then be able to regulate them. But this is undesirable for two reasons: First, such message traffic might constitute a relatively large increase in the number of messages that the mediator has to process. Relatively large, because the assignment of agents to various duties is a routine matter of administration, which is likely to be much more frequent than alarms. Given such large background traffic, the mediator may not be able to react rapidly enough to emergency situation. Secondly, and more importantly, organizational activities, such as the assignment of

inspectors for duty, are orthogonal to the functionality of the P/S mediator—efficiently disseminating event-notices to matching subscribers. Attempting to couple these functionalities together would complicate the mediator, and make it less efficient, and less secure. Note further that, as seen immediately below, the scope of the regulation over such organizational activities is not just restricted to the maintenance of roles that are used to render access permissions, as is the case with traditional access-control.

Finally, requirements concerning the behavior of K-inspectors, imposed by Point 5, are *not*, by and large, suitable for implementation via the P/S mediator. First, Point 5a requires the obligation of every K-inspector to subscribe to all K-alarms be fulfilled. Second, Point 5b implies a requirement that each K-inspector be monitored for his acknowledgement of the receipt of every K-alarm issued by a layman. Finally, Point 5c requires a K-inspector's failure to fulfill such duty result in the publication of a suitable meta-alarm. In essence, these points require an enforcement mechanism that (a) maintains the status of communication between users (not necessarily involving the mediator), and (b) fulfills the obligation to publish, or to subscribe to, a specified event-notice based on such status—which are, again, largely orthogonal to the core functionality of P/S service.

In conclusion, we argue that a policy like AP, which regulates alarms within a hospital, represents one of possibly many administrative aspects of the hospital that are inextricably intertwined with its general operation at large; thus, the implementation of such a policy does not really belong to the P/S mediator—a means of transporting event-notices. Moreover, as seen above, some provisions of policy AP just do not lend themselves to effective implementation by the P/S mediators. Indeed, none of the existing P/S services can handle the entire scope of policy AP. In Section 4 we will show how policy AP can be formulated and enforced in a decentralized, communal manner, with only marginal involvement of the P/S mediator.

3 Law-Governed Interaction (LGI)—an Overview

Broadly speaking, LGI [12] 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 *interaction-law* (or simply the "law") of the group. The messages thus exchanged under a given law $\mathcal{L}$ are called $\mathcal{L}$-messages, and the group of agents interacting via $\mathcal{L}$-messages is called an $\mathcal{L}$-community $\mathcal{C}_\mathcal{L}$ (or, simply, a *community* $\mathcal{C}$) .

We refer to entities that participate in an $\mathcal{L}$-community as *agents*[2], by which we mean autonomous actors that can interact with each other, and with their environment. An agent might be an encapsulated software entity, with its own state and thread of control, or a human that interacts with the system via some interface. A community under LGI is *open* in the following sense: (a) its membership can change dynamically, and can be very large; and (b) its members can be heterogeneous. For more details about LGI than provided by this overview, the reader is referred to [14, 1, 2].

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;

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

and (c) causing certain messages to be emitted spontaneously, under specified circumstances, via a mechanism we call *obligations*.

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* ($\mathcal{CS}$) of the community. The dependency of this control-state on the history of interaction is defined by the law $\mathcal{L}$ itself.

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

On a centralized enforcement 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 parts: (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 2(a)). The structure of the control-state, and its effect on the exchange of messages between members of $\mathcal{C}_\mathcal{L}$ are both determined by law $\mathcal{L}$.

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

Moreover, when dealing with stateful policies, these drawbacks of centralization cannot be easily alleviated by replicating the reference monitor $\mathcal{T}$, as it is done in the Tivoli system [10], for example. The problem, in a nutshell, is that if there are several replicas of $\mathcal{T}$, then any change in $\mathcal{CS}$ would have to be carried out *synchronously* at all the replicas. 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 member of community $\mathcal{C}$ (see Figure 2(b)). Unlike the central reference monitor $\mathcal{T}$ above, which carries the $\mathcal{CS}$ 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; the occurrence of an *exception*; and the coming due of an *obligation*. The operations that can be included in the ruling for a given regulated event, called *primitive operations*, are all local with respect to the agent at which the event occurred (called, the "home agent"). They include: operations on the control-state of the home agent; operations on messages, such as `forward` and `deliver`; and the imposition of an obligation on the home agent.

To summarize, an LGI law must satisfy the following locality properties: (a) a law can regulate

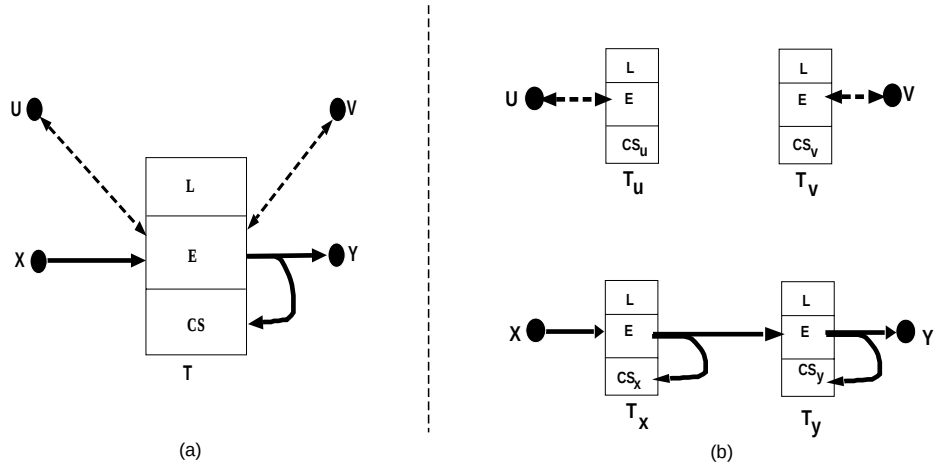

Figure 2: Law Enforcement: (a) centralized version; (b) decentralized law enforcement under LGI

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 member 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, while every $\mathcal{L}$-message exchanged between a pair of agents x and y passes through a pair of controllers, $\mathcal{T}_x$ and $\mathcal{T}_y$ (see Figure 2(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, with the locality of laws, greatly simplifies the structure of the controllers, making them easier to use as our *trusted computing base* (TCB).

3.2 The deployment of LGI

The mechanism of LGI, and particularly that of controllers as the law-enforcer, has been implemented (in Java) as a messaging middleware called Moses. Thus, all one needs for the deployment of LGI is the availability of a set of such trustworthy controllers, which run as distinct processes from each other, and from any clients, and a way for a prospective client to locate a running controller. For this purpose, we have also implemented a *controller-service* to maintain a set of controllers, as part of Moses.

For an agent x to engage in LGI communication under a law $\mathcal{L}$, after locating a controller via a controller-service, it needs to supply this controller with the law $\mathcal{L}$ it wants to employ, via specifying the text of $\mathcal{L}$ or its URL. The controller then checks if law $\mathcal{L}$ is well-formed, and if so, it starts to serve for this client. Only through this hand-shake between a controller and an agent—a procedure

called *adoption* of law $\mathcal{L}$—the agent can start to participate in $\mathcal{L}$-community $\mathcal{C}_{\mathcal{L}}$, becoming able to interact with other members. All these kinds of communication between an agent and its controller, including ones mentioned below, are facilitated by a Moses API. A graphical user interface is also provided for human users.

Note that it is quite possible for a single agent, x, to adopt the same law $\mathcal{L}$ more than once, whether connecting to a single controller or multiple controllers. In such a case, however, each adoption results in a distinct membership of x in $\mathcal{C}_{\mathcal{L}}$, and x participates in this community, representing in effect multiple members. That is, each such membership of x is associated with its own control-state, with respect to which, the locality of the law is strictly preserved. In an application where tighter membership control is necessary, one can choose to deploy a secretary of the community, as explained in [17].

Once x has adopted law $\mathcal{L}$, it may need to distinguish itself as playing a certain role, etc., 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 in [1]. A simple illustration of such certification is provided by our example law $\mathcal{AP}$ in Section 4, under which one may claim the role of a facility manager, for example.

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 flexibilities, which can be used to minimize the overhead of LGI under various conditions, as shown in [14].

3.3 The basis for trust between members of a community

Note that we do not propose to coerce any agent to exchange $\mathcal{L}$-messages under any given law $\mathcal{L}$. The role of enforcement here is merely to ensure that *any exchange of $\mathcal{L}$-messages, once undertaken, conforms to law $\mathcal{L}$*. In particular, our enforcement mechanism needs to ensure that a message received under law $\mathcal{L}$ has been sent under the same law; i.e., that it is not possible to forge $\mathcal{L}$-messages. For this, one needs the following assurances: (a) that the exchange of $\mathcal{L}$-messages is mediated by correctly implemented controllers; (b) that these controllers are interpreting the *same law* $\mathcal{L}$; and (c) that $\mathcal{L}$-messages are securely transmitted over the network.

Broadly speaking, these assurances are provided as follows. Controllers used for mediating the exchange of $\mathcal{L}$-messages authenticate themselves to each other via certificates signed by a CA specified by law $\mathcal{L}$, as we will see in Section 4. (Note that different laws may, thus, require different certification levels for the controllers used for its enforcement.) This authority is an important element of the trust between agents that exchange such messages. Messages sent across the network must be digitally signed by the sending controller, and the signature must be verified by the receiving controller. To ensure that a message forwarded by a controller $\mathcal{T}_x$ under law $\mathcal{L}$ would be handled by another controller $\mathcal{T}_y$ operating under the *same* law, $\mathcal{T}_x$ appends a one-way hash [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. (Thus, how members x and y get the text of law $\mathcal{L}$ is actually irrelevant to the assurance that both operate under the same law, although, for convenience, we have provided an HTTP-based law-server that provides agents with a library of laws.)

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 communicate with others that operate only under this law. For instance, given law $\mathcal{AP}$ introduced in Section 4, if the community of users of the P/S services are committed to publish and subscribe via $\mathcal{AP}$-messages, then the mediator will have to operate under law $\mathcal{AP}$ as well, if it is

to be able to serve them—and will thus be required to authenticate itself as prescribed by this law. Conversely, once the mediator commits itself to using $\mathcal{AP}$-messages (which is expected to be part of what qualifies a certified mediator, in implementing Points 1 and 2 of policy AP in Section 2), its users would be effectively compelled to do their publishing and subscription via $\mathcal{AP}$-messages as well. This is probably the best one can do in the distributed context, where it is impossible to ensure that all relevant messages are mediated by a reference monitor, or by any set of such monitors.

4 Implementation of the Alarm Policy

We now demonstrate our mechanism to regulate decoupled communication by introducing law $\mathcal{AP}$, which is the implementation, under LGI, of policy AP discussed in Section 2. We start with some general remarks on the deployment of a law.

To begin with, of course, the actual text of the law has to be stipulated. LGI currently supports two languages for writing laws: (a) a Prolog-like language, introduced in [12], and (b) a restricted version of Java, described in [18]. The former of these languages is employed below. We envision that such stipulation is done by a group of pertinent stake-holders, including, in our example, the administrators of this hospital, and the representatives of other users, with the help of computational specialists. In particular, this is the time when the communal decision is made as to which (kinds of) agents (can) play which roles in the interaction, including the choice of trustworthy CA. As a consequence, one needs to ensure that necessary certificates, in our example, for the mediator, the facility manager, and for K-experts of various K, are issued by the chosen CA and possessed by the intended agents, in order for them to claim the role. Note that the community under the current law may reuse the previously issued certificates, if they fit the current purposes.

Once law $\mathcal{L}$ is stipulated, it should be made available to agents that may participate in community $\mathcal{C}_{\mathcal{L}}$. As explained in Section 3.3, the trustworthiness of communal interaction is immune to how the actual text of $\mathcal{L}$ is distributed; e.g., one may send it in an e-mail to his peers, to be used for its adoption by each of them. If a trustworthy HTTP server is available, one can use it to store the text, to have it retrieved by the controller during the adoption of $\mathcal{L}$. As mentioned before, we also provide an HTTP-based law-server as part of the Moses middleware.

So far, we have assumed that the P/S mediator to be deployed is implemented as a single process. However, this is only for the sake of simplicity of presentation. In fact, our regulatory mechanism applies to a P/S mediator consisting of multiple, distributed processes, working in concert, just as well.[3] Particularly, no change in law $\mathcal{AP}$ below would be required to regulate the use of such a P/S mediator of a distributed architecture. This is because our regulation is applied *only* to the communication between the mediator and its users, not between the distributed mediator processes that make up the entire P/S service. Note that, in such a case, however, each mediator process has to adopt law $\mathcal{AP}$, and to present a certificate, attesting its mediator role, as a single-process mediator would, which will be explained shortly.

4.1 Law $\mathcal{AP}$ to regulate alarms

Law $\mathcal{AP}$, displayed in Figures 3 and 4, consists of two parts: the preamble and the rule section. The preamble gives this law its name, `ap`. It also contains the following clauses: First there is the `cAuthority` clause that identifies the public key of the CA to be used for the authentication of

[3]Note that our argument in Section 2.2 for communal regulation, as opposed to mediator-based regulation, also applies regardless of the mediator architecture.

the controllers that are to mediate $\mathcal{AP}$-messages, as described in Section 3.3. Second, there is an `authority` clause that identifies admin, represented by its public key, as a CA for certifying various roles played in this community. Finally, the `initialCS` clause defines the initial control-state of all agents in this community—it is empty in this case.

$\mathcal{P}$reamble:

```
law(ap).
cAuthority(publicKeyOfCAuth).
authority(admin, publicKeyOfAdmin).
initialCS([]).
```

$\mathcal{R}1$.
```
certified([issuer(admin),subject(Self),attributes([role(R)])])
     :- (R=mediator; R=facilityManager; R=expert(K)), do(+R).
```

Given an appropriate certificate from admin*, a term is inserted to represent the corresponding role.*

$\mathcal{R}2$.
```
sent(C, subscribe(alarm(AL)), M) :- member(kind(K),AL),
         (expert(K)@CS -> AL1=AL
         ; delete(AL,status(layman),AL2), append(AL2,[status(expert)],AL1)),
         (K=metaAlarm -> expert(K)@CS ; true),
         do(forward(C,subscribe(alarm(AL1)),M)).
```

Regulating subscription to alarms: laymen can subscribe to alarms published by experts, and only experts can subscribe to `metaAlarm`*.*

$\mathcal{R}3$.
```
sent(I, publish(alarm(kind(K),text(X))), M) :- clock(T)@CS,
         (expert(K)@CS -> S=expert
         ; S=layman, not(blocked(K)@CS), not(blocked(kind(K),text(X))@CS)),
         A=alarm(kind(K),time(T),informer(I),status(S),text(X)),
         do(forward(I, publish(A), M)),
         (S=layman -> do(+blocked(kind(K),text(X))),
              do(imposeObligation(releaseBlock(kind(K),text(X)),5,min)) ; true).
```

Regulating publishing of alarms.

$\mathcal{R}4$.
```
arrived(C, Msg, M)
     :- (Msg=subscribe(A); Msg=publish(A)), mediator@CS, do(deliver).
```

A mediator can receive subscriptions and publications.

$\mathcal{R}5$.
```
sent(M, notify(A), Cs) :- mediator@CS, do(multicast(M, Msg, Cs)).
```

A mediator can notify clients.

$\mathcal{R}6$.
```
arrived(M, notify(A), U)
     :- A=alarm(kind(K),time(T),informer(I),status(S),text(X)),
        (S=layman -> expert(K)@CS; true),
        (inspector(K)@CS, S=layman ->
         imposeObligation(handlingExpire(A,med(M)),10,min) ; true), do(deliver).
```

An alarm propagated by a mediator is delivered, but a layman does not get an alarm published by another layman.

$\mathcal{R}7$.
```
obligationDue(releaseBlock(kind(K),text(X))) :- do(-blocked(kind(K),text(X))).
```

The `blocked` *term to control the frequency of publishing is removed when the specified amount of time passes.*

Figure 3: Law $\mathcal{AP}$

The rest of the law consists of a set of rules, in a Prolog-like syntax. Most rules are followed by a comment (in italic), which, together with our discussion, should be understandable even for a reader not well versed in the LGI language for writing laws. Each rule has a *head*, to the left of symbol `:-`, and a *body*, to its right. Recall that the same law is interpreted individually by the controller associated to each agent in the community. A regulated event occurring at this home agent, as explained in Section 3.1, triggers a rule that has a matching head, if any (the matching is done in the order in which the rules are written). The triggered rule proceeds to check if all the goals in its body are attained, given the control-state of this agent.

In addition to the standard types of Prolog goals, the body of a rule may contain two distinguished types of goals. 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*, which always succeeds, has the form `do(p)`, where `p` is one of the primitive operations, mentioned in Section 3.1. It appends the term, `p`, to the ruling of the law. Thus, successful evaluation of a rule body with do-goals leads to a non-empty ruling, and the execution of the primitive operations therein. In what follows, we may speak of this effect as if the said rule itself were to execute the pertinent operations. (By default, an empty ruling implies that the event in question has no consequences—such an event is effectively ignored.)

Rule $\mathcal{R}1$ permits an agent to undertake some designated role by presenting an appropriate certificate issued by `admin`. As stipulated in Point 1 of policy AP, the roles regulated here are: (a) a P/S mediator; (b) a facility manager; and (c) an expert on K-alarms. The `certified` event that triggers this rule is generated when the controller is presented with a valid certificate, i.e., duly signed by an authority declared in an `authority` clause, in this case `admin`.[4] The `certified` event has as its argument the following representation of the submitted certificate:

```
[issuer(admin), subject(Self), attributes(role(R))].
```

Term `issuer(admin)` tells about the issuer of the certificate, while `subject(Self)` is used to signify the subject of the certification. `Self` is an LGI built-in variable that is bound to the identifier[5] of the home agent, which means that this `certified` event is generated when the home agent presents a self-certificate to its controller. Term `attributes` above describes what is certified about the subject, and in this case asserts that the agent is qualified for the role bound to `R` (which should be one of the three mentioned above). Thus, the rule inserts the binding of `R` into the control state, treated as the token of the certification in the rest of the law.

Rules $\mathcal{R}2$ through $\mathcal{R}6$ are central to the regulation over the interaction between P/S mediators and other members of the community. Rule $\mathcal{R}2$ is triggered when an agent, whose identifier is bound to `C`, sends its controller a subscription message, addressed to the intended mediator, whose identifier is bound to `M`. This rule allows any agent to subscribe to alarms of some kind with optional properties. However, the subscription of a layman is examined, and possibly transformed in the ruling, to implement Point 3; i.e., a layman may not receive an alarm published by another layman. This is done as follows: The law accepts a message of the form `subscribe(alarm(AL))` where `AL` is a list of attribute criteria of subscription, e.g., `[kind(fire)]`. The rule checks if the agent is an expert on alarms of kind `K`, specified in the message, by testing the presence of term `expert(K)` in the control state.[6] If so, the subscription is forwarded to the mediator with no change. Otherwise, the agent is a layman, and `status(expert)` is placed into the attribute criteria, while

[4] If the certificate is found invalid, an `exception` event is generated, which is ignored under this law, for the sake of simplicity.

[5] An agent identifier is of the form: `local-name@domain-name` [1].

[6] Syntax `(P->Q;R)` in the laws should read **if** `P` **then** `Q` **else** `R`.

all occurrences of `status(layman)`, if any, being deleted. When this transformed subscription is forwarded to the mediator, it effectively constrains the subscribed alarms to be issued only by experts. $\mathcal{R}2$ also denies a layman's attempt to subscribe to alarms of kind `metaAlarm`, by requiring the home agent to be a `metaAlarm`-expert, which partially fulfills Point 5c.

Rule $\mathcal{R}3$ regulates the publishing of an alarm. Given kind `K` of the alarm and its description `X` in the message sent by the agent, this rule adds the following properties before forwarding it to the mediator: (1) identifier `I` of the informer; (2) status `S`, either `expert` or `layman` depending on the informer being a K-expert; and (3) the current time. This ensures the authenticity of the alarm in general, and the informer's status in particular, on which the enforcement of Points 3 and 5b directly relies.

When a message to subscribe to alarms, or to publish an alarm, arrives at the controller of the addressed mediator, the corresponding arrived event is generated, which is handled by rule $\mathcal{R}4$. (Hereafter, we may simply say such a message is, upon its arrival at the agent, handled by the relevant rule.) Note in this rule (and in the next), another LGI built-in variable `Msg` is used that carries the entire regulated message. After ensuring the legitimacy of the mediator, the rule delivers the message to it. In parallel, rule $\mathcal{R}5$ permits a certified mediator to notify clients that (supposedly) have a matching subscription to a given published alarm. Note `multicast` is another primitive operation; it forwards the given message to a set of recipients, specified in its third argument. Combined, $\mathcal{R}4$ and $\mathcal{R}5$ implement Point 2.

Once an event-notice propagated by a mediator arrives at each selected subscriber, it is handled by rule $\mathcal{R}6$. This rule ensures that (even under a "faulty" mediator) a layman does not get a K-alarm published by another layman (Point 3), by requiring the recipient to be a K-expert, if it was published by a layman.

The frequency restriction on laymen's publishing, as stipulated in Point 4, is implemented as follows: A term, `blocked(kind(K),text(X))`, is inserted into the layman's control state by rule $\mathcal{R}3$ when it publishes a corresponding K-alarm. Rule $\mathcal{R}3$ also imposes an LGI obligation `releaseBlock`, specifying it to come due in 5 minutes, which is handled by rule $\mathcal{R}7$, to remove the `blocked` term above. Thus, rule $\mathcal{R}3$, by requiring such a `blocked` term be absent, prevents a layman from publishing the same alarm for the specified duration of time.

Rule $\mathcal{R}8$ allows for a facility manger to send a message to assign an inspector of alarms of some kind on duty (Point 5). The arrival of such a message triggers rule $\mathcal{R}9$, which checks if the recipient is a K-expert, before inserting term `inspector(K)`. In addition, rule $\mathcal{R}9$ implements Point 5a by forwarding subscription to all K-alarms on behalf of this appointed K-inspector.

Rules $\mathcal{R}10$ and $\mathcal{R}11$ implement the remainder of Point 4, which allows a facility manager to block a layman from publishing altogether. Rule $\mathcal{R}11$ ensures that it is not an expert that is blocked, and inserts a term, `blocked(K)`, whose absence is required by rule $\mathcal{R}3$ for a layman's publication to be forwarded to the mediator.

As stipulated in Point 5b, each K-inspector should promptly examine every alarm issued by a layman. Rule $\mathcal{R}12$ allows such an inspector to send a copy of the alarm to the facility manager, as a proof of noticing it. This rule ensures that the correct identifier of the facility manager is used in the message, by matching it to the one kept in the `fm(FM)` term, which was inserted by rule $\mathcal{R}9$ when the appointment message arrived. Upon the arrival of such an acknowledgement, rule $\mathcal{R}13$ delivers the message to the facility manager.

Finally, (the remainder of) Point 5c, and what is inferred from Point 5b as well, is implemented as follows. Upon a K-inspector receiving a layman's alarm, rule $\mathcal{R}6$ imposes an obligation called `handlingExpire(A,med(M))` where `A` is the representation of the entire alarm, and `M` is the mediator that has propagated the alarm to the inspector. This obligation is set to come due in 10 minutes. If this K-inspector acknowledges the alarm in time, specifically by sending its copy to the

$\mathcal{R}8$. `sent(FM, inspectorOnDuty(kind(K),med(M)), X) :- facilityManager@CS, do(forward).`

A facility manager can send message to put an inspector on duty.

$\mathcal{R}9$.
```
arrived(FM, inspectorOnDuty(kind(K),med(M)), X)
      :- expert(K)@CS, do(+inspector(K)), do(+fm(FM)),
         do(forward(X, subscribe(alarm([kind(K)])), M)).
```

A K-expert can be appointed as a K-inspector, who is obliged to subscribe to all K-alarms.

$\mathcal{R}10$. `sent(FM, blockLay(K), L) :- facilityManager@CS, do(forward).`

A facility manager can block a layman from publishing.

$\mathcal{R}11$. `arrived(FM, blockLay(K), L) :- not(expert(K)@CS), do(+blocked(K)), do(deliver).`

A `block` *message causes the corresponding term to be inserted.*

$\mathcal{R}12$.
```
sent(I, ack(A), FM) :- fm(FM)@CS, Obl=handlingExpire(A,med(M)),
         obligation(Obl)@CS, do(repealObligation(Obl)), do(deliver).
```

An inspector can acknowledge a layman's alarm to the facility manager.

$\mathcal{R}13$. `arrived(I, ack(A), FM) :- facilityManager@CS,do(deliver).`

$\mathcal{R}14$.
```
obligationDue(handlingExpire(A,med(M))) :- clock(T)@CS,
         do(forward(Self, publish(alarm(kind(metaAlarm),time(T),informer(Self),
              status(layman),text(unack(A)))), M)).
```

If an alarm received from a laymen is not acknowledged by the inspector, a `metaAlarm` *is published.*

Figure 4: Law $\mathcal{AP}$ (continued)

facility manager, handled by $\mathcal{R}12$ as seen above, the obligation that has been imposed is repealed. Otherwise, when the obligation comes due, by rule $\mathcal{R}14$, the corresponding `metaAlarm` is sent to the mediator for publication, carrying the unacknowledged alarm and the identifier of this K-inspector in two attributes, `text` and `informer`, respectively.

5 On the Performance of Proposed Mechanism

Concentrating only on the access-control aspect of the proposed mechanism, we compare its performance to that of what can be implemented by the P/S mediators themselves. Our measurement of the most recent implementation of LGI controller shows that each regulated event is processed in about 0.6 *ms*, on a Sun Fire 280R, UltraSPARC-III server (900 MHz), under Solaris 2.8 operating system and Java 1.3. (Note that this version still does not incorporate the improvement measures suggested previously in [14].) The general picture that emerges below is as follows: Our mechanism tends to decrease the load on the mediator (in some cases dramatically), reducing the probability of congestion, and to increase the mediator's throughput, but it involves modest increase in latency, when the mediator is not congested.

Note that in this section the analyses and the experiment are based on a single-process mediator. Under a multiple-process, distributed mediator, assuming that the number of clients per mediator node and the frequency of access requests are the same as those of the single-process mediator, the general picture mentioned above should remain qualitatively the same. One notable difference would be that, as the cost of the event-notice routing, which our mechanism would not affect, becomes the dominating factor of the latency, the overhead on the latency caused by our mechanism

should decrease accordingly.

5.1 Load on the mediator

Our proposed mechanism tends to reduce the load on the mediator, that is, the number of messages received by it, for two reasons.

First, certain messages would be blocked at the periphery by the controllers associated with users (Figure 5). For example, due to rule $\mathcal{R}3$, implementing policy point 4, if a layman tries to publish the same alarm too frequently, it will be blocked. This is particularly effective in blocking buggy or careless users from bombarding the mediator with a large number of messages, as demonstrated in Section 5.2. On the other hand, a mediator-based implementation is vulnerable to such unruly users, who can cause the denial of service to the entire community.

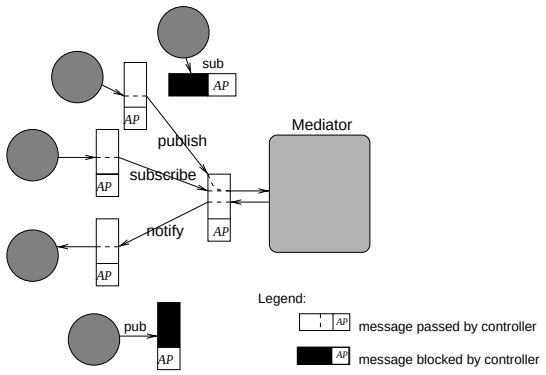

Figure 5: The effect of the periphery processing

The second factor that can affect the load on the mediator has to do with the maintenance of the status of users, which is relevant to the policy at hand. For example, our AP policy is sensitive to the appointment of an agent as an inspector. Under the proposed communal mechanism, such appointment is carried out by exchanging messages between the users themselves, not involving the mediator. On the other hand, in the mediator-based approach, any change of status relevant to the policy at hand needs to be communicated to the mediator, and thus increases the load on it. This increase might be relatively significant because it is caused by routine interaction between users, which might be much more frequent than alarms that reflect exceptional circumstances. Note that even if certificates are used to establish access rights "off-line," the mediator itself, rather than the periphery, still has to grant (or deny) each access, and it must be made aware of the loss of such rights (e.g., due to the revocation of certificates).

5.2 Congestion caused by unruly informers

We have conducted an experiment that measures the effect of an unruly informer on the performance of P/S services, under (a) the mediator-based access-control, and (b) the proposed decentralized mechanism. The two configurations used in this experiment are depicted in Figure 6. They both operate under a policy that limits the frequency of publication allowed to any one agent to ten per second. (Of course, this frequency is set well below the threshold to cause congestion to the particular mediator we used.)

In both configurations, essentially the same mediator M is used. In either configuration, one thread of M each handles the in-coming requests from another process, and deposits them in a single queue of M. The processing of user requests, including sending each event-notice to the

target subscriber(s), is implemented as another thread (henceforth called the processing thread) that serves one request in the above queue at a time. In configuration (b), the above policy is implemented in LGI similarly to the frequency control on laymen in law $\mathcal{AP}$, while in configuration (a), M's processing thread does some bookkeeping for the time of the most recent publication of each user, based on which each publication request is accepted or rejected. We decided on this implementation for (a), based on the assumption about a realistic access-control module that: (1) it would have to operate on the content of each message (not just on the frequency of all requests alone); and (2) it would not be implemented to have a single thread per user, due to rather large resource consumption.

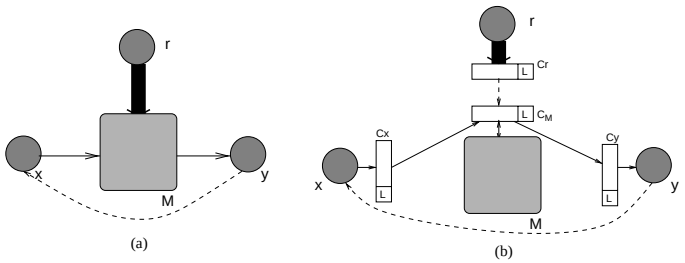

Figure 6: Coping with an unruly informer

Also, in both configurations, informer x publishes event-notices to M—slowly enough not to cause any congestion—which are conveyed to subscriber y. When y receives each notification, it sends an acknowledgement (directly) to x. Some time after these agents start the communication, $28\,s$ to be exact, a "roguish" agent r begins to publish as fast as it can (in effect, about 1500 publications per second). This publishing lasts $60\,s$. As an indicator for the latency between x and y, we measure the round-trip time (RTT) of each publication that starts when x publishes it, and ends when x receives y's acknowledgement.

Shown in Figure 7 is the result of this experiment. In the run for case (a), before r begins its publishing, the RTT remains stable at a few milliseconds. For the duration of r's publishing, the RTT increases linearly, as expected, up to about $15\,s$, by its end, nearly all of which is spent for the publication to wait in M's queue. After r's publishing ends, it takes quite some time, more than $15\,s$, for the RTT to return to the normal, while M processes all publications accumulated in its queue.

In contrast, the graph for case (b) is fairly flat throughout, because r's publications are mostly captured by the controller that handles r, and does not transmit more than ten publications per second. Although in the graph it is indistinguishable, the RTT is slightly higher than the normal range of case (a), because of the larger latency under LGI, particularly within a LAN, where this experiment has been conducted (see Section 5.4).

5.3 Throughput of the mediation

As mentioned above, the LGI controller processes each regulated event in well below a millisecond. In the literature [3, 4], it is suggested that, given thousands of subscriptions, the mediator would perform its matching algorithm in a few milliseconds on a comparable platform. Thus, if the controller runs on a separate host from the mediator's, it would not affect the throughput, i.e., the number of publications mediated per second. Moreover, under our mechanism, access-control related computation is carried out at the periphery, i.e., by the controllers associated with users, as depicted in Figure 5. Since the mediator themselves have less to do per publication, their

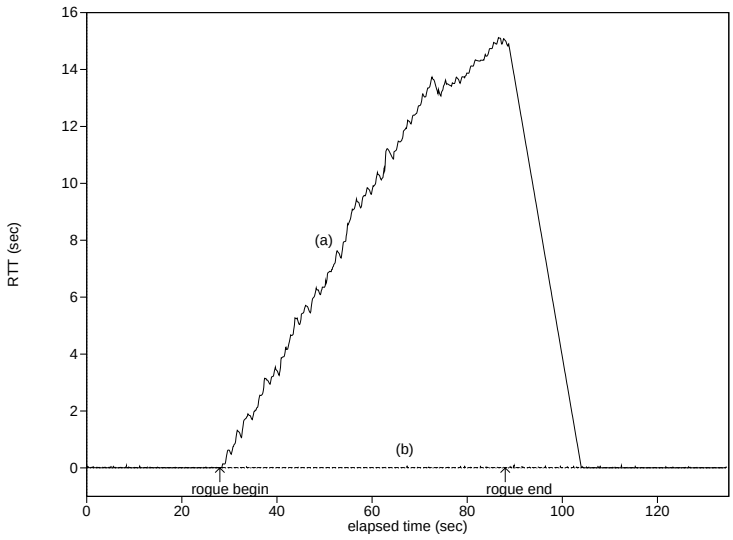

Figure 7: Congestion caused by r

throughput increases proportionately (with respect to the real-time if the controller runs on a separate host, and to the CPU-time otherwise).

5.4 End-to-end latency

We compare the latency in propagating an authorized notice from informer x to subscriber y through mediator M, *under no mediator congestion*, in two cases: (a) the mediator providing access-control, and (b) each user and the mediator regulated by LGI. These two cases are depicted in Figure 8, where $T_{\alpha,\beta}$ stands for the time of communication between two processes α and β. (ϵ will be explained shortly.) Letting T_a and T_b be the latency in cases (a) and (b), respectively, we consider the *relative overhead* $(T_b - T_a)/T_a$, by following the general discussion in [14].

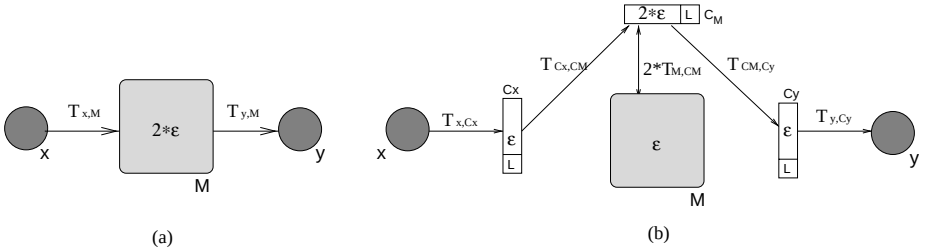

Figure 8: Two cases for latency analysis

First, we consider a setting where both pairs (x, M) and (M, y) are connected via WAN, while each controller is located in the same LAN as its client (though in a distinct host). Under this setting:

$$T_a = 2 \cdot T_{WAN} + 2 \cdot \epsilon$$

where T_{WAN} is the TCP/IP communication time between two computational entities that reside in different LANs; we use ϵ as the time for computing the matching subscriptions, as well as that of the access-control (depicted inside the rectangle for M in Figure 8). Also:

$$T_b = 5 \cdot \epsilon + 4 \cdot T_{LAN} + 2 \cdot T_{WAN}$$

where T_{LAN} is the TCP/IP communication time between two entities that reside in the same LAN; we use the same ϵ as above for the time of event-evaluation by the LGI controller, as well. We assume T_{WAN}, T_{LAN}, and ϵ to be $100\,ms$, $1\,ms$, and $1\,ms$, respectively. This estimation of ϵ is made—given the above mentioned cost of event-evaluation by the LGI controller, and that of the matching indicated in the literature—under the assumption that the cost of the access-control would be roughly equivalent to that of LGI event-evaluation. The corresponding relative overhead is nearly equal to $(3\epsilon + 4T_{LAN})/2T_{WAN}$ since $T_{WAN} \gg \epsilon$, which is about 0.03. This shows that, over the WAN, the latency increase due to our mechanism is negligible.

Second, we consider a setting where x, y, and M are all located within the same LAN, and, in case (b), each of them is associated with a controller in the same host. T_{WAN} and T_{LAN} in the above equations are replaced with T_{LAN} and T_{pipe}, respectively, where T_{pipe} is the communication time via pipe, for two entities that reside on the same machine. We assume T_{pipe} to be $0.1\,ms$. The corresponding relative overhead is $(3\epsilon + 4T_{pipe})/2(T_{LAN} + \epsilon)$, about 0.8, which we view as acceptable. This estimate has also been confirmed by measurements obtained in the experiment of Section 5.2.

6 Conclusion

Decoupled communication, which requires no direct association between the producers of information and its consumers—as practiced under the publish/subscribe (P/S) middleware, and under the tuple-space middleware—has emerged as an often useful element in the architecture of distributed and heterogeneous applications. But we maintain that this type of communication has a dark side: its indefinite, and potentially global, reach—the very reason for its power—may complicate the system using it, making it less predictable, more brittle, and less safe.

It is our thesis, however, that these drawbacks of decoupled communication can be alleviated effectively, efficiently, and scalably, by decentralized regulation of its use. Taking the P/S paradigm as the representative of what enables decoupled communication, we argued why such regulation does not lend itself to effective implementation by the P/S mediators. And we showed how it can be carried out by means of a distributed control mechanism called Law-Governed Interaction (LGI) [14], along with the Moses middleware that implements it. We also demonstrated the importance of such regulation, and its efficacy, by means of a case study of the treatment of alarms, disseminated via a P/S middleware, in a large hospital.

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. (available from `http://www.cs.rutgers.edu/~minsky/pubs.html`).

[2] X. Ao, N. Minsky, and V. Ungureanu. Formal treatment of certificate revocation under communal access control. In *Proc. of the 2001 IEEE Symposium on Security and Privacy, May 2001, Oakland California*, May 2001. (available from `http://www.cs.rutgers.edu/~minsky/pubs.html`).

[3] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, and D.C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proc. of The 19th International Conference on Distributed Computing Systems (ICDCS)*, pages 262–272, May 1999.

[4] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscirbe systems using binary decision diagrams. In *Proc. of The 23rd International Conference on Software Engineering (ICSE)*, pages 443–452, May 2001.

[5] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, April 1989.

[6] P.Th. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. Technical report, EPFL, January 2001. DSC ID: 2000104.

[7] D. Garlan and D. Notkin. Formalizing dsign spaces: Implicit invocation mechanisms. In *Proc. of VDM'91: 4th International Symposium of VDM Europe on Foraml Software Development Methods; LNCS 551*, pages 31–44, Noordwijkerhout, The Netherlands, October 1991. Springer-Verlag.

[8] M. Hapner et al. *Java Message Service*. Sun Microsystems Inc., August 2001. Version 1.0.2b, website: `http://java.sun.com/products/jms/docs.html`.

[9] IBM Corp. *Gryphon – The system*. website: `http://www.research.ibm.com/gryphon/Gryphon/gryphon.html`.

[10] 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).

[11] Z. Miklós. Towards an access control mechanism for wide-are publish/subscribe systems. In *Proc. of Intn'l Workshop on Distributed Event-Based Systems (DEBS'02)*, July 2002.

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

[13] N.H. Minsky, Y.M. Minsky, and V. Ungureanu. Safe tuplespace-based coordination in multiagent systems. *Journal of Applied Artificial Intelligence (AAI)*, 15(1):11–33, January 2001. (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. (available from `http://www.cs.rutgers.edu/~minsky/pubs.html`).

[15] D.S. Rosenblum and A.L. Wolf. A design framework for internet-scale event observation and notification. In *Proc. of the Sixth European Software Engineering Conference; Zurich, Switzerland; LNCS 1301*, pages 344–360. Springer-Verlag, September 1997.

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

[17] C. Serban, X. Ao, and N.H. Minsky. Establishing enterprise communities. In *Proc. of the 5th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2001), Seattle, Washington*, September 2001. (available from `http://www.cs.rutgers.edu/~minsky/pubs.html`).

[18] C. Serban and N.H. Minsky. Using java as a language for writing lgi-laws. Technical report, Rutgers University, July 2002.

[19] Chenxi Wang, Antonio Carzaniga, David Evans, and Alexander L. Wolf. Security issues and requirements for Internet-scale publish-subscribe systems. In *Proceedings of the Thirty-Fifth Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii, January 2002.