On Shouting “Fire!”: Regulating Decoupled Communication in Distributed Systems[⋆]

Takahiro Murata and Naftaly H. Minsky

Dept. of Computer Science, Rutgers University
Piscataway, NJ, 08854 USA
{murata,minsky}@cs.rutgers.edu

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 essential 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 tamed 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. Along the way, we illustrate the importance of such regulation, and its effectiveness, by considering the treatment of alarms in a large hospital.

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 [5] – the most prominent are: the *publish/subscribe* (P/S, for short) [6,14], and the Linda-like *tuple-space* [4,12] 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

⋆ Work supported in part by NSF grants Nos. CCR-97-10575 and CCR-98-03698, and by the NJ Commission on Science and Technology “excellence award.”

impart, it sends 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 Sect. 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 [10,8], by providing P/S mediators with access-control capabilities. Some commercial P/S mediators, particularly those that implement

[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.

JMS [7], 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) [13], and a middleware called Moses that implements this mechanism – an overview of which is provided in Sect. 3.

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. Sect. 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. Sect. 3 is an overview of law-Governed Interaction (LGI) – the computation mechanism on which this paper is based. In Sect. 4 we show how our example alarm policy of Sect. 2 is formulated, and scalably enforced under LGI. In Sect. 5 we consider the performance of our control mechanism; and we conclude in Sect. 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 by experts, but* not *by laymen (Fig. 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.*

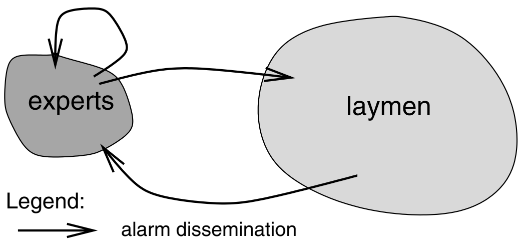

Fig. 1. Alarm dissemination between experts and laymen

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 [18].

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 an 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 (a) that maintains the status of communication between users (not necessarily involving the mediator), and (b) that, based on such status, exercises further regulation on the communication, including fulfilling the obligation to publish, or to subscribe to, a specified event-notice on behalf of certain users – 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 belong in its entirety 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 Sect. 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 [11] 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 [13,1,2].

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

[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.

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; (b) transformation of certain messages, possibly rerouting them to different destinations; and (c) causing certain messages to be emitted spontaneously, under specified circumstances, via a mechanism we call *obligations*.

A crucial feature of LGI is that its laws can be *stateful*. That is, a law $\mathcal{L}$ can be sensitive to some function of the history of the interaction among members of $\mathcal{C}_\mathcal{L}$, 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. This architectural decision is based on the observation that a centralized mechanism to enforce interaction-laws in distributed systems is inherently unscalable, as it can become a bottleneck, and a dangerous single point of failure. The replication of such an enforcement mechanism, as seen in the Tivoli system [9], would not scale either, due to the required synchronous update of $\mathcal{CS}$ at all the replicas, when dealing with stateful policies.

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 coming due of an *obligation*; and the occurrence of an *exception* in executing an operation in the ruling for another event. The agent at which a regulated event has occurred is called the *home agent* of the event. The ruling for a given regulated event is computed based on the local control state $\mathcal{CS}_x$ of the home agent 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. The operations that can be included in the ruling for a given regulated event, called *primitive operations*, are all local with respect to the home agent. They include: operations on the control-state of the home agent, such as insertion (`+t`), removal (`-t`), and replacement (`t<-s`) of terms; operations on messages, such as `forward` and `deliver`; and the imposition of an obligation on the home agent.

To summarize, an LGI law satisfies the following locality properties: (a) a law can regulate explicitly only *local events* at individual home agents; (b) the ruling for an event `e` can depend only on `e` itself, and on the *local control-state* $\mathcal{CS}_x$ of the home agent x; and (c) the ruling for an event can mandate only *local operations* to be carried out at the home agent x.

Decentralization of law-enforcement: The enforcement of a given law is carried out by a distributed set $\{\mathcal{T}_x \,|\, x \in \mathcal{C}\}$ 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

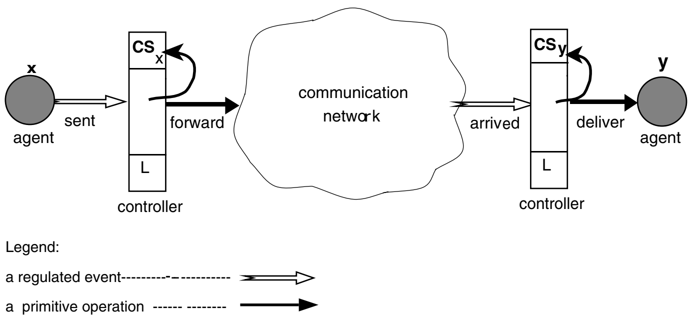

Fig. 2. enforcement of the law

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 Fig. 2).

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. These greatly simplify the structure of the controllers, making them easier to use as our *trusted computing base* (TCB).

Finally we point out that the law-enforcement mechanism ensures 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. As described in [2], 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 [15] `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 $\mathcal{C}_{\mathcal{L}}$ operate under the same law.

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[3]. 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, after locating a controller via a controller-service, it needs to supply this controller with the law $\mathcal{L}$ it wants to employ, by specifying the text of $\mathcal{L}$ or its URL. The controller then checks if

[3] A public distribution version is being finalized as of the time of writing.

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 $\mathcal{C}_{\mathcal{L}}$. All these kinds of communication between an agent and its controller, including ones mentioned below, are facilitated by a Moses API, while a graphical user interface is 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 [16].

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 $\mathcal{L}$. This can be done 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 Sect. 4, under which one may claim some of the roles, stipulated in policy AP.

4 Implementation of the Alarm Policy

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

In stipulating the actual text of the law, LGI currently supports two languages: (a) a Prolog-like language, introduced in [11], and (b) a restricted version of Java, described in [17]. The former of these languages is employed below. We envision that such stipulation is done by a group of pertinent stake-holders, with the help of computational specialists. In particular, as in law $\mathcal{AP}$, a trustworthy CA may be required, which certifies the status of various role players via issuing and revoking certificates. Note, in general, the use of certificates does not compromise the scalability of our mechanism, because: (a) a CA can be a distributed agency; and (b) the CA does not need to be on-line at all, in order to have a certificate issued by it verified. For more detail on certification, particularly on the treatment of revocation, the reader is referred to [2].

Once stipulated, law $\mathcal{L}$ should be made available to agents that may participate in community $\mathcal{C}_{\mathcal{L}}$. As explained in Sect. 3.1, 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 them. If a trustworthy HTTP server is available, one can use it to store the text, and to have it retrieved by the controller during the adoption. For convenience, 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 our presentation. In fact, our regulatory mechanism applies to a P/S mediator consisting of multiple, distributed processes, working in concert, just as well[4]. 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 Figs. 3 and 4, consists of two parts: the preamble and the rule section. The preamble gives this law its name, `ap`, and contains the following clauses: (a) the `cAuthority` clause that identifies the public key of the CA, used for the authentication of the controllers that are to mediate $\mathcal{AP}$-messages, as described in Sect. 3.1; (b) an `authority` clause that identifies `admin`, represented by its public key, as a CA for certifying various roles played in this community; and finally, (c) the `initialCS` clause defining the initial control-state of all agents in this community – it is empty in this case.

The rest of the law consists of a set of rules, most of which are followed by a comment (in italic); thus, together with our discussion, the rules should be understandable to the reader. Each rule has a *head*, to the left of symbol `:-`, and a *body*, to its right. Recall that, as explained in Sect. 3.1, the same law is interpreted individually by the controller associated with each agent in the community. A regulated event triggers, at the controller of the home agent, one rule that has a matching head at a time, if any, in the order in which the rules are written. The rule evaluation proceeds to find a rule that all the goals in its body are attained, given the control-state of this agent; in the absence of such a rule, the regulated event in question is ignored.

In addition to the standard types of Prolog goals, the body of a rule may contain two distinguished types of goals as follows: First, a *sensor-goal*, of the form `t@CS`, where `t` is any Prolog term, attempts to unify `t` with each term in the control-state of the home agent. Second, a *do-goal*, which always succeeds, has the form `do(p)`, where `p` is one of the primitive operations, mentioned in Sect. 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.) We now discuss how the rules of law $\mathcal{AP}$ implement policy AP.

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

𝒫reamble:

```
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.*

Fig. 3. Law $\mathcal{AP}$

Establishing roles: Point 1 of AP is implemented by rule $\mathcal{R}1$ as follows: The `certified` event that triggers this rule is generated when the home agent presents its controller a valid certificate, i.e., duly signed by an authority declared in an `authority` clause, in this case `admin`[5]. As seen in the head of the rule, 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 the `subject` term is used to signify the subject of the certification. `Self` is an LGI built-in variable that is bound to the identifier (id)[6] of the home agent; thus in this case the home agent must have presented a certificate whose subject is the agent itself (i.e., a self-certificate). The `attributes` term describes what is certified about the subject; given `R` in its argument `role(R)` bound to one of the three roles, (a) a P/S mediator, (b) a facility manager, or (c) an expert on K-alarms[7], it attests that the agent is allowed to assume the role in question. 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.

Regulating subscription: Rule $\mathcal{R}2$ is triggered when an agent, whose id is bound to `C`, sends its controller a subscription message of the form `subscribe(alarm( AL))`, addressed to the intended mediator, whose id is bound to `M`. `AL` in the message is bound to a list of attribute criteria of this subscription, e.g., `[kind(fire)]`. Except two special cases, explained immediately below, this message is forwarded to `M`, with no change.

Here are the special cases: First, the subscription to alarm kind `K` by a layman, i.e., a home agent without term `expert(K)` in its control-state, is (possibly) transformed; that is, `status(expert)` is placed into the attribute criteria, while all occurrences of `status(layman)`, if any, being deleted. When such a transformed subscription is forwarded to the mediator, it constrains the subscribed alarms to be issued only by experts, denying the chance of laymen subscribing to laymen's alarms. This effectively implements Point 3. Second, a layman's attempt to subscribe to alarms of kind `metaAlarm` is also denied, by requiring the home agent to be a `metaAlarm`-expert, which partially fulfills Point 5c.

When a message to subscribe to alarms arrives at the controller of the addressed mediator, the corresponding `arrived` event is generated, and 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 `Msg` is another LGI built-in variable, which carries the entire regulated message. The rule delivers the message to the mediator, after ensuring its legitimacy.

Regulating publication: 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 home agent, this

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

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

[7] Goals `P;Q` and `(P->Q;R)` should read `P` **or** `Q`, and **if** `P` **then** `Q` **else** `R`, respectively.

rule adds the following properties before forwarding it to the mediator: (1) id `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. The published alarm will be delivered to the mediator by rule $\mathcal{R}4$.

Once the mediator computes the set of clients that have a matching subscription to the given published alarm, the mediator attempts to notify them, which triggers rule $\mathcal{R}5$. Primitive operation `multicast` forwards the given message to a set of recipients, specified in its third argument. Note that $\mathcal{R}4$ and $\mathcal{R}5$, combined, implement Point 2.

When the event-notice propagated by the 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, in that case.

Frequency restriction on laymen: 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. $\mathcal{R}3$ also imposes an LGI obligation `releaseBlock`, specifying it to come due in 5 minutes, which is handled by $\mathcal{R}7$, to remove the `blocked` term above. Thus, $\mathcal{R}3$, by requiring such a `blocked` term be absent, prevents a layman from publishing the same alarm for the duration of time specified in Point 4.

Rules $\mathcal{R}8$ and $\mathcal{R}9$ allow a facility manager to block a layman from publishing altogether (the remainder of Point 4). Rule $\mathcal{R}9$ 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.

Handling of K-alarms: Rules $\mathcal{R}10$ and $\mathcal{R}11$ allow for a facility manger to assign a K-expert to the duty of a K-inspector (Point 5). In addition, rule $\mathcal{R}11$ implements Point 5a by forwarding subscription to all K-alarms on behalf of this appointed K-inspector.

Rules $\mathcal{R}12$ and $\mathcal{R}13$ allow each K-inspector to send a copy of the alarm to the facility manager, as a proof of noticing it (Point 5b). Note the `fm(FM)` term – inserted by $\mathcal{R}11$ when the appointment message arrived – carries the id of the facility manager.

Point 5c (or its remainder) is implemented as follows. Upon a K-inspector receiving a layman's alarm, rule $\mathcal{R}6$ imposes an obligation, `handlingExpire(A, med(M))` – set to come due in 10 minutes – where `A` and `M` are the representation of the alarm, and the mediator that has propagated the alarm, respectively. If this K-inspector acknowledges the alarm in time, resulting in triggering $\mathcal{R}12$ as seen above, the imposed obligation is repealed. Otherwise, on the obligation coming due, $\mathcal{R}14$ sends a `metaAlarm`, carrying the unacknowledged alarm and the id of this K-inspector in the attributes, `text` and `informer`, respectively, to the mediator for publication.

```
R8. sent(FM, blockLay(K), L) :- facilityManager@CS, do(forward).
R9. arrived(FM, blockLay(K), L)
        :- not(expert(K)@CS), do(+blocked(K)), do(deliver).
```
A facility manager can block a layman from publishing.

```
R10. sent(FM, inspectorOnDuty(kind(K),med(M)), X)
        :- facilityManager@CS, do(forward).
```
A facility manager can send message to put an inspector on duty.

```
R11. 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.

```
R12. sent(I, ack(A), FM) :- fm(FM)@CS, Obl=handlingExpire(A,med(M)),
           obligation(Obl)@CS, do(repealObligation(Obl)), do(deliver).
R13. arrived(I, ack(A), FM) :- facilityManager@CS,do(deliver).
```
An inspector can acknowledge a layman's alarm to the facility manager.

```
R14. 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.*

Fig. 4. Law $\mathcal{AP}$ (continued)

5 On the Performance of the 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 the 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 the Solaris 2.8 operating system and Java 1.3. (Note that this version still does not incorporate the improvement measures suggested previously in [13].) 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

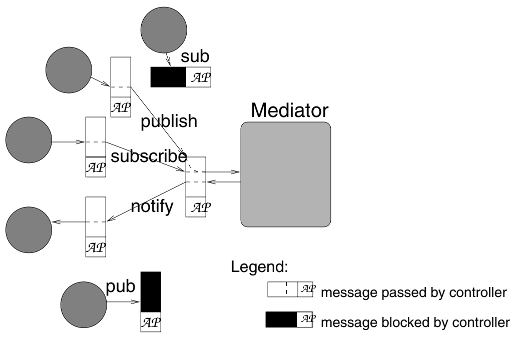

Fig. 5. The effect of the periphery processing

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 (Fig. 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 Sect. 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.

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,

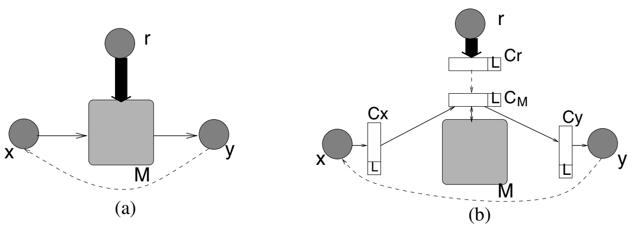

Fig. 6. Coping with an unruly informer

and (b) the proposed decentralized mechanism. The two configurations used in this experiment are depicted in Fig. 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 processing thread per user, due to rather large resource consumption, as well as difficulty in allowing concurrent operation on the subscription base.

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 Fig. 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.

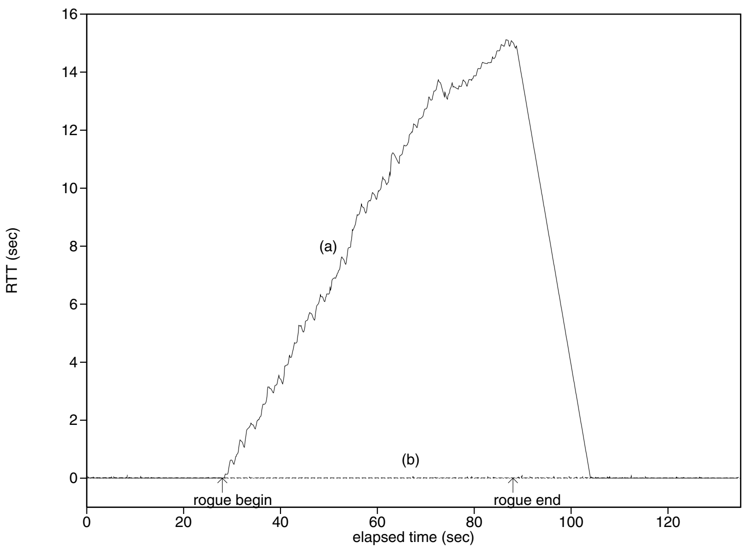

Fig. 7. Congestion caused by r

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 Sect. 5.3).

5.3 Throughput of the Mediation and End-to-End Latency

Under our mechanism, access-control related computation is carried out at the periphery, i.e., by the controllers associated with users, as depicted in Fig. 5. Since the mediator themselves have less to do per publication, their throughput increases proportionately (with respect to the real-time if the controller runs on a separate host, and to the CPU-time otherwise).

Next, 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 Fig. 8, where $T_{\alpha,\beta}$ stands for the time of communication between two processes α and β, while ϵ is used as the time for finding the matching subscriptions, that of the access-control, and that of event-evaluation by the LGI controller. 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 [13].

Based on realistic figures for the communication time, and the time to compute matching subscriptions suggested in the literature, e.g., [3], the relative

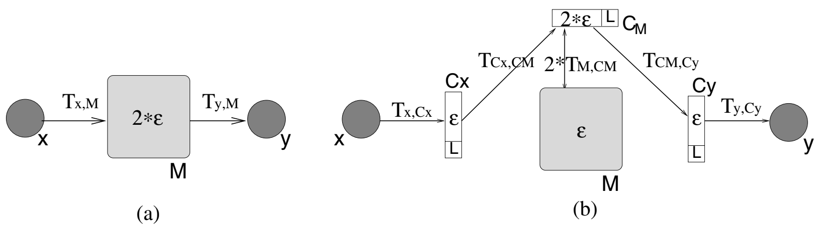

Fig. 8. Two cases for latency analysis

overhead in a WAN setting and that in a LAN setting are: a few percent and several tens of percent, respectively. The former is negligible, and the latter, we view, is acceptable, given in particular the separate controllers running, at least for the mediator, and for its users, to maintain the scalability. This estimate has also been confirmed by measurements obtained in the experiment of Sect. 5.2.

6 Conclusion

Decoupled communication, which requires no direct association between the producers of information and its consumers, is often essential 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. We have demonstrated these difficulties by taking the P/S paradigm as the representative of decoupled communication, and by using the treatment of alarms in a large hospital as an example application of the paradigm.

We have argued that appropriate regulation of decoupled communication does not lend itself to effective implementation by the P/S mediators themselves, but requires decentralized regulation defined and enforced directly over the agents attempting to communicate with each other. We have shown how such decentralized regulation can be carried out, efficiently and scalably, using the Law-Governed Interaction (LGI) mechanism.

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. 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 Intn’l Conf. on Soft. Eng. (ICSE)*, pages 443–452, May 2001.
4. N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, April 1989.
5. 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.
6. D. Garlan and D. Notkin. Formalizing dsign spaces: Implicit invocation mechanisms. In *Proc. of VDM’91: 4th Intn’l Sympo. of VDM Europe on Foraml Software Development Methods; LNCS 551*, pages 31–44, Noordwijkerhout, The Netherlands, October 1991. Springer-Verlag.
7. 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`.
8. IBM Corp. *Gryphon – The system.* website: `http://www.research.ibm.com/gryphon/Gryphon/gryphon.html`.
9. G. Karjoth. The authorization service of tivoli policy director. In *Proc. of the 17th Annual Computer Security Applications Conf. (ACSAC 2001)*, December 2001.
10. 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.
11. N.H. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, February 1991.
12. 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`).
13. 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`).
14. D.S. Rosenblum and A.L. Wolf. A design framework for internet-scale event observation and notification. In *Proc. of the Sixth European Soft. Eng. Conf.; Zurich, Switzerland; LNCS 1301*, pages 344–360. Springer-Verlag, September 1997.
15. B. Schneier. *Applied Cryptography.* John Wiley and Sons, 1996.
16. C. Serban, X. Ao, and N.H. Minsky. Establishing enterprise communities. In *Proc. of the 5th IEEE Intn’l Enterprise Distributed Object Computing Conf. (EDOC 2001), Seattle, Washington*, September 2001. (available from `http://www.cs.rutgers.edu/˜minsky/pubs.html`).
17. C. Serban and N.H. Minsky. Using java as a language for writing lgi-laws. Technical report, Rutgers University, July 2002.
18. C. Wang, A. Carzaniga, D. Evans, and A.L. Wolf. Security issues and requirements for Internet-scale publish-subscribe systems. In *Proc. of the 35th Annual Hawaii Intn’l Conf. on System Sciences*, Hawaii, January 2002.