A Novel, Privacy Preserving, Architecture for Online Social Networks

Zhe Wang[1] and Naftaly H. Minsky[1,*]

[1]Rutgers University, Department of Computer Science

Abstract

The centralized nature of conventional OSNs poses serious risks to the security and privacy of information exchanged between their members. These risks prompted several attempts to create decentralized OSNs, or DOSNs. The basic idea underlying these attempts, is that each member of a social network keeps its data under its own control, instead of surrendering it to a central host, providing access to it to other members according to its own access-control policy. Unfortunately all existing versions of DOSNs have a very serious limitation. Namely, they are unable to subject the membership of a DOSN, and the interaction between its members, to any global policy—which is essential for many social communities. Moreover, the DOSN architecture is unable to support useful capabilities such as narrowcasting and profile-based search.

This paper describes a novel architecture of decentralized OSNs—called *DOSC*, for "online social community". DOSC adopts the decentralization idea underlying DOSNs, but it is able to subject the membership of a DOSC-community, and the interaction between its members, to a wide range of policies, including privacy-preserving narrowcasting and profile-sensitive search.

Received on XXXX; accepted on XXXX; published on XXXX

Keywords: Online social networks; Decentralization; Control; Privacy; Security

doi:10.4108/XX.X.X.XX

1. Introduction

An *online social network* (OSN) can be defined broadly as a community of people that interact with each other via some electronic media. The most popular ones of these are the huge OSNs, like Facebook and Twitter. But there are many others, mostly small or mid-size, OSNs that play important social roles. These include: support groups consisting of people suffering from a certain illness, such as AIDS; students who wish to share views of their teachers; workers discussing their work condition and their managers; and physicians consulting each other about their difficult cases. Such disparate OSNs may have different purposes and different concerns, but the conventional architecture of practically all of them—the huge and the small— is *centralized*. That is, the interaction between members of an OSN is mediated via a central host—or a virtually central one, which may run on many computers, but is managed centrally.

Unfortunately, although centralization is a very convenient way for implementing OSNs, it has several well known drawbacks, which include: (a) risks to the security and privacy—to which we will refer, henceforth, simply as "security"—of information exchanged between the members of an OSN; (b) lack of scalability; and (c) the existence of a single point of failure. The last two of these drawbacks can be mitigated via very large, complex, and expensive infrastructures—like those used by Facebook and Twitter. But the main risks to the security of such OSNs is harder to mitigate because they are rooted in the centralized architectures per se, independently of the policies imposed on an OSN by its central host. These risks are due to two main factors. First, members of centralized an OSN are at the mercy of the organization that maintains it. This organization can, in particular, sell the information in its possession (legally or illegally), or even modify it. Second, the

*Corresponding author. Email: minsky@cs.rutgers.edu data maintained by the central host is vulnerable to various malicious attacks, which can be quite lucrative. Such attacks can be mounted by insiders, say the programmer that maintains the software of the OSN; and by attackers from the outside.

Such concerns about centralized OSNs prompted several attempts to create decentralized OSNs, or DOSNs, such as LotusNet [2], Safebook [11], PeerSoN [7], and others. The basic idea underlying all these attempts at decentralization, is that each member of the community in question should keep its data under its own control, instead of surrendering it to a central host, providing access to it to other members of the DOSNs *according to its own access-control policy*. This does enhance the security of the members of DOSNs, but it suffers from a very serious limitation, which does not mar centralized OSNs.

Due to decentralization, DOSNs lose the ability to subject its members to a global control of any kind. But because social communities generally need to operate subject to some chosen policies, which regulate the membership of the community and the manner in which its members interact with each other, this loss of control entails a serious reduction in trust between community members, and in its security. Now, it is true that the policies that govern purely social communities are generally informal, imprecise, implicit, and only occasionally enforced. But such policies should be tightened and enforced under an OSN, because its membership can be larger than that of a traditional social community, and there is much less familiarity and trust between its members.

The lack of control under DOSNs has another unfortunate consequence. It makes it impossible to provide important capabilities like *narrowcasting* and *profile-based search* without incurring massive loss of security (cf. Section 7.1).

The Contribution of this Paper: We will describe in this paper a novel way for decentralizing OSNs, giving rise to an architecture we call DOSC, for Decentralized Online Social Community. DOSC adopts the decentralization idea underlying DOSNs, complementing it with a powerful means for establishing a wide range of policies governing the membership of a social community, and the interactions among its disparate distributed members. This is done by governing the exchange of messages between the members of DOSC using a decentralized—as thus scalable—Middleware called LGI [17, 18]. Among other consequences of this architecture is its ability to support capabilities such as *narrowcasting* and *profile-based search* without loss of security. (We note here that we will continue using the term OSN as a general term for online social network, implying no specific architecture.)

It should be pointed out, that while DOSC should be sufficiently fast for human interaction in medium size communities—with thousands or tens of thousands members—some of its capabilities, like narrowcasting, would not scale to the size of OSNs like Facebook and Twitter. But these huge OSNs may not require decentralization, as the hundreds of millions of their members seem not to be very concerned about issues such as privacy.

The rest of this paper is organized as follows. Section 2 discusses the need for security in various kinds of OSNs. Section 3 outlines the nature of policies that OSNs may need to be governed by. Section 4 provides an overview of the middleware called Law-Governed Interaction (LGI) which serves as the foundation the DOSC architecture. Section 5 introduces a basic model of DOSC. Section 6 is a summarized description of an implemented case study that demonstrates how this abstract model can be used for a concrete application. Section 7 complements the case study by introducing more advanced capabilities available under the basic model; and then it generalize the basic model itself, by adding to it several more advanced features. Section 8 discusses the overall performance of DOSC, and describes its limits. Section 9 discusses related works, by others, and by the authors. And we conclude in Section 10.

2. On the Security Concerns of Centralized OSNs

Security is, or should be, of serious concerns to the members of many online social networks, particularly if they exchange sensitive messages containing such things as private medical and financial information. But the nature of these concerns is different in two major types of OSNs, which we call *autonomous* and *bound* OSNs. We will define both types of OSNs below, and discuss the nature of their security concerns, and the risks to their security due to centralization.

2.1. Autonomous OSNs

We define autonomous OSNs to be those that are not subject to any outside authority—except the authority of the law of the country in which the OSN operates. Of course, the members of an autonomous OSN are subject to the policy defined by it, which may vary widely. For example, the policies of an OSN designed for a support group of people suffering from AIDS is likely to be very different from the policies established by Facebook, which is also autonomous. The following is an example of one such OSN, and a discussion of its security concerns.

Consider an OSN created to enable a set of physicians to consult with each other about various medical issues they confront. We call this OSN *MC*, for “medical consultation.” The participants in *MC* may not know each other, and may practice all over the world. A member of *MC* may send a query—that describes an issue he or she confronts—to all other members; or, more likely, to a subset of its members, based on some criteria. And the receiver of such a query may answer it.

The information exchanged between the members of *MC* is clearly very sensitive, both to the physicians and to their patients, which would often be the subject to the queries made by members of *MC*. Having the process of consultation mediated by a central host, and having the information exchanged between the physicians maintained centrally by this host, can seriously compromise the security of both the physicians and their patients. The risk here is particularly serious because the host of such an OSN is likely to become a target for attacks—by hackers from the outside, as well as by insiders—since the information maintained by it can be exploited for illicit financial gains. Therefore, we have implemented *MC* in a decentralized manner [15] more than 10 years ago—before the term OSN has been introduced. The following are part of the constraints which have been imposed on this community by that implementation.

1. **Membership**: An agent x is allowed to join this community if it satisfies one of the following two conditions:

   (a) If x is one of the *founders* of this community, as certified by a specified certification authority (CA) called here $ca1$. (Note: we assume that there are at least three such founders.)

   (b) if x is a medical doctor, as certified by the CA called $ca2$, representing the medical board; and (ii) if x garners the support of at least three current members of this community.

   And, a regular member (not a founder) is *removed* from this community if three different members vote for his removal.

2. **Reputation**: Each member must maintain a *reputation value* that summarizes other members' feedback on the quality of his responses to posted queries, requiring no central reputation server. Furthermore, this reputation must be presented along with every response to a query. (Note that the reputation thus maintained by a member x cannot be manipulated by x himself.)

Note that although the above provisions where quite sophisticated, for a decentralized implementation, the paper that implemented them did not define a full fledged DOSC. In particular, that paper [15] did not support any of the advanced features discussed in Section 7, and did not fully support the basic model of DOSC introduced in Section 5.

2.2. Bound OSNs

We say that an OSN is bound if it operates in the context of some organization that has jurisdiction over it, and may own the information exchanged by the members of the OSN in question.

There is a growing realization[31] that OSNs that operate within an organization—such as manufacturing, commercial enterprises, medical centers, or even the military—can be beneficial for it. This seems to be particularly the case for OSNs that provide for micro-blogging, as is evident from the purchase of the Yammer—a prominent micro-blogging OSN that serves organizations—by Microsoft, for $1.2 Billion. We will have more to say about Yammer itself, but first we outline some of functional features one can expect from this kind of OSNs.

Consider a large and geographically distributed enterprise E that provides a centralized micro-blogging OSN for its employees. Suppose that such an OSN—which we call *WP*, for "WorkPlace"— distinguishes between groups of employees, enabling the members of each groups to communicate with each other. Such groups may be the following: (a) all the employees of E; (b) the non-managerial staff of E; (c) the managerial staff of E; and (d) members of various task forces operating in E. Note that these groups may overlap partially, as a single employee may belong to several groups. And the enterprise in question may impose some control over the membership of the various groups, and may establish some constraints regarding the communication between the members of different groups. For example, suppose that two of the task forces of enterprise E, which form groups in *WP*, consult to other companies, which may compete with each other. It is obviously paramount for these subgroups not to have access to each other's information.

There are several types of security concerns in the context of such OSNs. First, like in autonomous OSNs, individual members would be concerned about their own privacy. For example, a staff member would not want members of the management to read their complains about the workplace. Second, the information exchanged between the employees of enterprise E can carry sensitive information about the business of this enterprise. It is therefore important for the enterprise for this information not to be exposed to the outside, at least not on a large scale. Third, the enterprise is likely to be concerned about violations of its constraints on the communication between different groups of *WP*.

The Risks to Security due to Centralization: There are two types of centralization to be considered here, which we call strong and weak centralizations. Strong centralization is like the one practiced by Yammer, the Microsoft OSN mentioned above. Yammer provides services to a host of different enterprises—they claimed to serve about 200,000 different enterprises. Of course, Yammer establishes policies that provide necessary separation between the various enterprises it serves. But the information belonging to all these enterprises is maintained centrally by the Yammer system. Such centralization of commercial and industrial information of many different companies is very risky, as it is likely to attract attacks from the inside of Yammer, and from the outside—thus compromised the security of many of the clients of Yammer.

A much better approach would be to use an *intramural* Yammer-like OSN. This, weaker form of centralization, would be much safer than using Yammer. But if this system relies on a centralized database, it would still be vulnerable to breaches of security. Indeed, if all the information generated by the *WP* is available to its software, then the rogue programmers of this OSN will have a fairly free access to all of it, disregarding the required boundaries between different groups.

3. On the Nature of Policies that OSNs May Need to be Governed by

We survey here various types of *communal policies* that an OSN may need to establish. By "communal" we mean either global policy that is to govern all members of an OSN, or a policy that governs some subgroup of its members. All the policies discussed here can be easily established, by enforcement, under centralized OSNs, but none of them can be established under the DOSN architecture. All these kinds of policies can be established under our DOSC, as we will show for some of them in Section 6 and in Section 7.

Membership Control: Control over membership is crucial to many social communities whether they are autonomous or bound. The set of members may be predefined. Alternatively, and more commonly, the membership can be limited to individuals that satisfy certain predefined criteria, which can be checked by various credentials. For example, the membership of our OSN example *MC*, of medical consultation, is limited to physicians; and the membership of the workplace OSN example *WP* is limited to the employees of a given enterprise. Moreover, in addition, or instead, of such characterization of acceptable members, the OSN may condition the admission of new member on the approval of a number of existing members of the OSN.

Another important aspect of membership, is the removal of existing members. There are many possible types of procedure for doing that. As a simple example, consider an OSN that has a member that plays the role of a *manager*, which provides him/her with the power to remove any existing member *x* by simply sending a message "`leave`" to it; and this, in turn, should force *x* to leave the community in question.

Identification of Members: Practically every OSN needs to establish a coherent manner in which its members identify themselves to each other. One can distinguish between three basic mode of identification: (1) Members may be allowed to be *anonymous*. (2) Members may be allowed to operate under a pseudonym of their choice. And (3) members may be required to use they real and authenticated names. For example, under *WP*, members may be required to identify themselves via their unique names within the enterprise in question—which is to be authenticated by certificates provided to them by a certification authority (CA) of this enterprise.

Besides their name, such as above, members may identify themselves by a certain *profile*—that may contain such things as their medical specialty in the case of *MC*; or by their roles in the enterprise, in the case of *WP*. The authenticity of such a profile and the way it is being used needs to be established by the policy of the OSN in question.

Constraints on the Behavior of Members of an OSN: Sometimes one needs to impose constraints on what members can do. Such constraints may depend on the profile of individual members, and on the history of their interaction with others. We have just seen an example of such constraints: only a member that plays the role of manager can send a "`leave`" message to others. And any member that gets such a message must cease to operate within these community. As another example, in the context of *WP*, the type of messages that members are entitled to send, or the type of posts that they are entitled to make, may depend on their roles in the enterprise in question.

Global Access Control (AC) Policies: One of the intended consequences of decentralization under DOSNs is that it enables each member to apply its own AC policy to its own data—e.g., to the set of posts it produced, which are maintained in its own database. This is useful, but certainly not sufficient. Because an OSN may want to impose some global AC policies. This is particularly true for bound OSNs, such as *WP*. The posts being produced by the various members of this OSN really belong to the enterprise *E* in the context of which it operates, and which thus has an authority over their treatment. The enterprise may relegate to individual members the right to apply their own AC policies, *provided* that these policies conform to the global policy of an enterprise. For example, the global policy of the *WP* may be that a group of members assigned to deal with the business of a given client-company can communicate only with each other, as long as they belong to the same group.

And global constraints are common even in autonomous OSN. A case in points are the various friendship-related rules that govern the interaction between members of Facebook.

Cooperation Protocols: In Section 1 we pointed out that the DOSN architecture does not provide certain important capabilities such as search for members whose profile satisfies a specified condition, and narrowcasting based on the profile of members. However, as we shall show in Section 7, such capabilities can be provided even in decentralized OSN, if all its members can be trusted to cooperate in providing them. Such cooperation can be ensured by imposing a suitable cooperation protocol on all members of an OSN.

4. The (LGI) Middleware—a Partial Overview

Governance of interactions among the members of a decentralized OSN requires a suitable middleware at its foundation. We have chosen the middleware called *law governed interaction* (LGI) for this purpose. LGI is broadly related to conventional access control (AC) mechanisms such as RBAC [20] and XACML [14]. But it differs from them in several aspects that are critical for decentralizing OSNs; the most important of which are: (a) LGI is completely decentralized; and (b) it is fully stateful, which means that it is sensitive to the history of interaction; and (c) LGI control is quite scalable, even for stateful policies. But it should be pointed out that LGI is somewhat incidental to this model, as one can device other kinds of middlewares that can serve this purpose. Consequently, we will not make any systematic comparison here between LGI and other middlewares, because LGI is not the subject of this paper—it is just a tool.

We present here only a partial overview of LGI, focusing on the following key aspects of it, which are most relevant to this paper: (1) the local nature of LGI laws (LGI replaces the term "policy" with the term "law," for a reason not discussed here); and (2) the decentralized enforcement of laws. Another important aspect of LGI is discussed in Section 7. We also give, below, a simple but complete example of an LGI law and on its effect.

A more detailed presentation of this middleware, and a tutorial of it, can be found in its manual [17]—which describes the release of an experimental implementation of LGI. For additional information the reader is referred to a host of published papers, some of which will be cited in due course.

4.1. LGI Laws, and their Local Nature

Although the purpose of LGI is to govern the exchange of messages between different distributed actors, the LGI laws do not do so directly. Rather, a law governs the *interactive activities* of any actor operating under it, in particular, by imposing constraints on the messages that such an actor can send and receive.

A *law* $\mathcal{L}$ is defined over three elements—described with respect to a given actor x that operates under this law: (1) A set E of *interactive events* that may occur at any actor, including the arrival of a message at x, and the sending of a message by it. (2) The *control-state* (or, simply, state) S_x associated with x—which is distinct from the internal state of x, of which the law is oblivious. And (3) a set O of *interactive operations*—such as forwarding a message and accepting one—that can be mandated by a law, to be carried out at x upon the occurrence of interactive events at it.

Now, the role of a law is to decide what should be done in response to the occurrence of any interactive event at an actor operating under it. This decision, with respect to an actor x, is formally defined by the following mapping:

$$E \times S_x \to S_x \times (O)^*. \tag{1}$$

In other words, for any a given $(event, state)$ pair, the law mandates a new state, as well as a (possibly empty) sequence of interactive operations to be carried out at x. Note, in particular, that the ruling of the law upon the occurrence of an event depends on the state of x at that moment; and that the same law determines how the state can change. LGI laws are, therefore, *stateful*—i.e., *sensitive to the history of the interactive-events*, at a given actor x. Moreover, although this is not evident from the above abstract definition, an LGI law can be *proactive*, in that it can force some messages to emanate from an actor, under certain circumstances, even if the actor itself did not send such messages—thus these laws can ensure both *safety and liveness* properties.

Note that LGI laws are *local* in the sense that they depends only the occurrence of events at a single actor, and on the interactive state of this actor alone; and a law can effect directly only the interactive behavior of the actor operating under it. It is worth pointing out that although locality constitutes a strict constraint on the structure of LGI laws, it does not reduce their expressive power, as has been proved in [17]. In particular, despite its *structural locality*, an LGI law can have global sway over a set of actors operating under it.

Finally, note that the law is a complete function, so that any mapping of the type defined above is considered a valid law. This means that a law of this form is *inherently self consistent*—although a law can, of course, be wrong in the sense that it may not work as intended by its designer.

About Languages for Writing Laws: Formula 1 is an abstract definition of the semantics of laws. It does not, in particular, specify a language for writing laws. In fact, the current implementation of LGI supports two different *law-languages*, one based on Prolog, and the other on Java; and another simpler law-language is under development. But the choice of language has no effect on the semantics of LGI, as long as the chosen language is sufficiently powerful to specify all possible mappings defined by Formula 1.

Space limitation preclude the description of any of these languages, but to give a sense of how LGI operates, and to illustrate the use of dynamically changing state of agents for establishing coordination protocol, we introduce here a simple but potentially useful law, written in the Prolog-based law-language.

A Law of Polite Conversation—an Example: This law, which we call the *ping-pong* law, or $\mathcal{L}_{PP}$, establishes a communication protocol that may be viewed as supporting polite conversation. Specifically, the effect of this law can be described, informally, as follows: All members of the $\mathcal{L}_{PP}$-community are able to communicate with each other, via messages of the forms `ping(M)` and `pong(M)`—with arbitrary text `M`—subject to the following protocol. Once a member x from this community sends a ping message—representing such things as a request or a question—to another member y, x would not be able to send other pings to y until it gets a reply from y in the form of a pong messages. And y can send only one pong message to x for every ping it gets from it. (The sense in which such exchange may be considered polite is fairly self evident.)

The formal statement of this law—written in the Prolog-based law language of LGI, which is a kind of *event-condition-action* language—is displayed in Figure 1. The gist of this law may be clear from its text, and a complete explanation of it can be found in [17].

Preamble: law(PP,language(prolog)).

```
R1. sent(X,ping(M),Y) :- not(pingTo(Y)@CS),
        do(add(pingTo(Y)),do(forward).

R2. arrived(X,ping(M),Y)
     :- do(add(pingFrom(X))), do(deliver).

R3. sent(X,pong(M),Y) :- pingFrom(Y)@CS,
        do(remove(pingFrom(Y))),do(forward).

R4. arrived(X,pong(M),Y)
     :- do(remove(pingTo(X)@CS)), do(deliver).
```

Figure 1. The Ping-Pong Law

4.2. The Decentralized Law Enforcement, and the Concept of $\mathcal{L}$-*agent*

The local nature of laws enables their decentralized enforcement, because a law can be enforced on every actor subject to it with no knowledge of, or dependency on, the simultaneous interactive state of any other actor of the system. Such enforcement is scalable even for highly stateful policies that are sensitive to the history of interaction (cf. [18]). Here is how the enforcement of LGI works.

To communicate under a given LGI law $\mathcal{L}$, an actor x needs to engage a generic software entity called *controller*[1], which generally does not reside on the host of its patron x. The controller is built to mediate the interactive activities of any actor that engages it, under any well formed law that the actor chooses. Once such a controller is engaged by an actor x, subject to a law $\mathcal{L}$, it becomes the private mediator for the interactive activities of x, and is denoted by $T_x^{\mathcal{L}}$. The pair $\langle x, T_x^{\mathcal{L}}\rangle$ is called an $\mathcal{L}$-*agent*—or, more generally an *LGI-agent*, and sometimes simply an *agent*. And a set of interacting $\mathcal{L}$-*agent*, for a given law $\mathcal{L}$, is called an $\mathcal{L}$-*community*.

Figure 2 depict the manner in which a pair of agents, operating under possibly different laws, exchange a message. (An agent is depicted here by a dashed oval that includes an actor and its controller.) Note the *dual nature* of control exhibited here: The transfer of a message is first mediated by the sender's controller, subject to the sender's law, and then by the controller of the receiver, subject to its law. This dual control, which is a direct consequence of the local nature of LGI laws, has some important consequences which are beyond the scope of this paper.

Mutual Recognition: It should be pointed out that a pair of interacting LGI-agents can recognize each other as such, and can identify each other law by its one-way hash. This enables them to recognize when they operate under the same law, thus belonging to the same $\mathcal{L}$-community. And if they operate under different laws, they are able to get the text of each other's law.

About the Trustworthiness of Controllers: Consider a set S of agents interacting via LGI, and let T_S be the set of controllers employed by them. T_S is, essentially the *trusted computing base* (TCB) of S. There are several reasons for trusting the controllers in T_S, despite the fact that unlike most TCBs, T_S is to be distributed. Some of these reasons are, briefly, as follows.

First, T_S can be maintained by what is called a *controller service* (CoS), which is to be managed by

[1]Controllers can can actually be hosted by *controller-pools*, each of which can host a number of *private controllers*, which may operate under different laws.

some trustworthy company—which may well be the company, or the virtual organization, that uses the CoS as its TCB. Second, controllers are generic and, like language compilers, can be well tested, and thus more trustworthy than the disparate actors that use them. Third, the distributed T_S is more fault tolerant than a single, central, reference monitor, because it does not constitute a single point of failure. And, fourth, T_S is more secure than a central, reference monitor, because it does not constitute a single point of attack.

About Performance: A comprehensive study of the overhead incurred by LGI control had been published in [19]. Broadly speaking, this overhead turns out to be relatively small, often smaller than the overhead incurred by control mechanisms such as XACML—beside being scalable. Moreover, this overhead is quite negligible for communication over WAN. The average contribution to this overhead by the computation in a controller was found—in circa 2000—to be around 50 microseconds. It is considerably lower with the present hardware.

5. A Basic Model of a Decentralized Online Social Community (DOSC)

We introduce here a model of a decentralized OSN that, unlike the DOSN architecture, enables the governance of an OSN via enforced policy—which we call a *law*—that can establish its overall structure and behavior. We call this model DOSC (for *Decentralized Online Social Community*); and we refer to a specific OSN under this model as a *DOSC-community*, or simply a *community*.

The DOSC model is *generic*, and rather abstract, in the sense that it does not have any built-in communal structure. But it can support a wide range of different types of communities, whose structure and behavior is determined by the laws chosen for them. We do, however, present a concrete example of a specific communal structure. This is done in Section 6, and continues with some more advanced communal capabilities in Section 7.1.

The model of DOSC described here is basic. Some more advanced aspects of this model are introduced in Section 7.2 and Section 7.3. This section is organized as follows. Section 5.1 is a definition of this model; Section 5.2 describes the launching of a DOSC-Community; Section 5.3 discusses the manner in which such a community operates; and Section 5.4 discusses the analysis of networks in the decentralized context of DOSC.

5.1. A Definition of a DOSC-Community

A community C under the DOSC-model is defined as a 4-tuple $\langle M, \mathcal{L}, T, S\rangle$, where M is the set of members of C; $\mathcal{L}$ is the law that governs this community (an LGI-law, to be exact)—and is often denoted by $\mathcal{L}_C$; T is a set of generic LGI controllers that serve as the middleware trusted to enforce any law $\mathcal{L}$ loaded into them; and S, called the *support* of C, is a set of components that provides various services to community C, and is mostly specific to it. We now elaborate on this schematic definition of the DOSC-model by providing some details about its four elements, and about the relations between them. This overall structure of a DOSC-community is depicted schematically in Figure 3.

The Set M of Members of a community: An individual member x of a community C is a triple $\langle user, mediator, database\rangle$, where *user* is usually a human, operating via some kind of computational platform, like a smart phone; *mediator* is one of the LGI-controllers in T that mediates all interactions between x and other members of C, and which has direct access to both the *user* of x and its database—subject to law $\mathcal{L}_C$; and *database*, is an optional repository of information that is accessible directly only to the mediator of x, as well as to the user himself (but the access of the user to its own database is not subject to the law $\mathcal{L}_C$, as depicted by dashed arrows in Figure 3.

Note that the function of the database of x—if it exists—is to maintain information associated with this member, such as the set of Twitter-like micro-blogs posted by x, or its Facebook-like page—information that may be made accessible to other members, via the mediator, subject to law $\mathcal{L}_C$. All such databases, used by members of a given community, must have the same APIs, which must be consistent with the law of that community. (The tool set associated with the DOSC-model—which is not part of the definition of this model—contains a tool for constructing and deploying a database, with a default API. But the law of a specific community might require a different API; and the *support* of that community may contain the means for constructing databases with that API.)

The Law $\mathcal{L}_C$ of community C: This law endows a DOSC-community with its overall structure, in particular by controlling its membership, as well as the interactive behavior of its members. The generality of LGI laws (cf. Section 4) endows this model with great deal generality regarding the nature of the community governed by it. In particular, suitable laws can make a community behave like our *medical consultation* community *MC*, or like the *WorkPlace* community *WP*. For that matter, laws can be written to create analogs of Facebook or Twitter as well, although our decentralized model would not be able to sustain the present size of such OSNs.

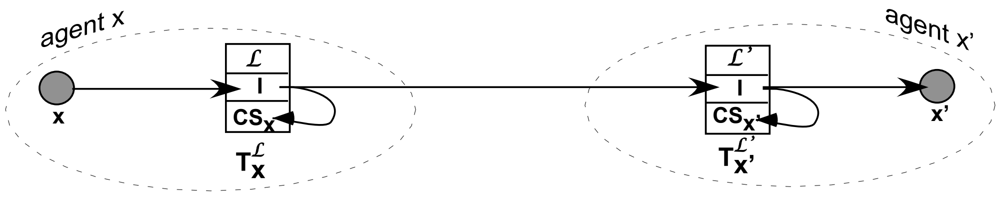

Figure 2. Interaction between a pair of LGI-agents, mediated by a pair of controllers under possibly different laws.

The set T of Controllers: T is meant to be the *trusted computing base* (TCB) of a DOSC. Every user can create its own controller, using the software provided by the released LGI middleware. But if malicious corruption of controllers by their users is of concern, then it is better for the members of a community to adopt controllers created and maintained by a trusted *controller service* (CoS), so that they can authenticate each other as bona fide LGI controllers. For such a CoS to be trusted to provide genuine controllers, this service needs to be managed by a trusted organization. In particular, in the case of bound-community, such as *WP*, the *CoS* may be maintained by the organization in the context of which the community is to operate—as in the case discussed in Section 6. For more about the security and trustworthiness of controllers see Section 4, and the manual of LGI [17].

The Support S of a Given Community C: A DOSC-community C may require various services provided by web-servers that are not themselves member of C; and, with one exception, most of them are not defined by the generic DOSC-model. Such services may be designed specifically for the community at hand, or may exist independently of it. Member of C interacts with such services subject to law $\mathcal{L}_C$, while the services themselves may or may not communicate subject to this or any other LGI-law. The only member of the support of a community that is required by this model is the *law-server LS*, which would contain law $\mathcal{L}_C$, and possibly other laws, as we will see in Section 7.2. Here are some examples of other services that may belong to S: (a) a certification authority (CA), which may be used for the authentication the various members of the community; (b) a *naming service* that provides unique names of community members; (c) an index service for searching; and (d) a reputation service that maintains the reputation of members of C.

Note that the existence of central support service would not compromise significantly the scalability of a DOSC-community, if it is used relatively rarely. And it would not compromise significantly the privacy of a DOSC-community, if it does not contain sensitive information. The law-server LS is certainly in this category, as it is used only when a new member joins the community, and it is not generally a secret. The other potential parts of the support, such as the naming service, probably belong to this category as well.

A Convention: Although the DOSC model is generic, and has nothing to say about the structure and behavior of any community operating under it, we introduce here a useful convention. It is about what is commonly called a *profile*. The profile of a member x, which we denote by p_x, is a set of attributes that can be made visible to other members, subject to the law of the community in question. Technically the profile of x is part of the *control-state* maintained by the controller. And the specific attributes that belong to the profile and the manner in which they are created and modified, depend on the law as well. We will see an example of a profile in Section 6.

5.2. The Launching of a DOSC-Community

A specific DOSC-community, C is launched by constructing its *foundation*, and then having individual members join it incrementally. The foundation of a community consists of: (1) the law $\mathcal{L}_C$ under which this community is to operate, which is to be placed in the law-server LS; (2) the controller service CoS, whose controllers would enforce this law; and (3) the support S to be used by this particular community. Each of these parts of the foundation of C can be either built specifically for it, or selected from existing such items. In particular, the controller service CoS may be designed specifically for C, but it may already exists, serving many different DOSC-communities, as well as other applications. And some, or all, parts of the *support* S of C—such as its CA—may have an independent existence, serving other applications.

Once the foundation of C is constructed, anybody can attempt to join it as a member—an attempt that may fail, as we shall see below—via the following four steps: First, the user needs to deploy its private database, if it is required by law $\mathcal{L}_C$. Second, the user needs to acquire an LGI-controller from the CoS used by C, and instruct this controller to download law $\mathcal{L}_C$ from the law-server. Third, the user must provide its newly adopted controller with a link to its private database, if

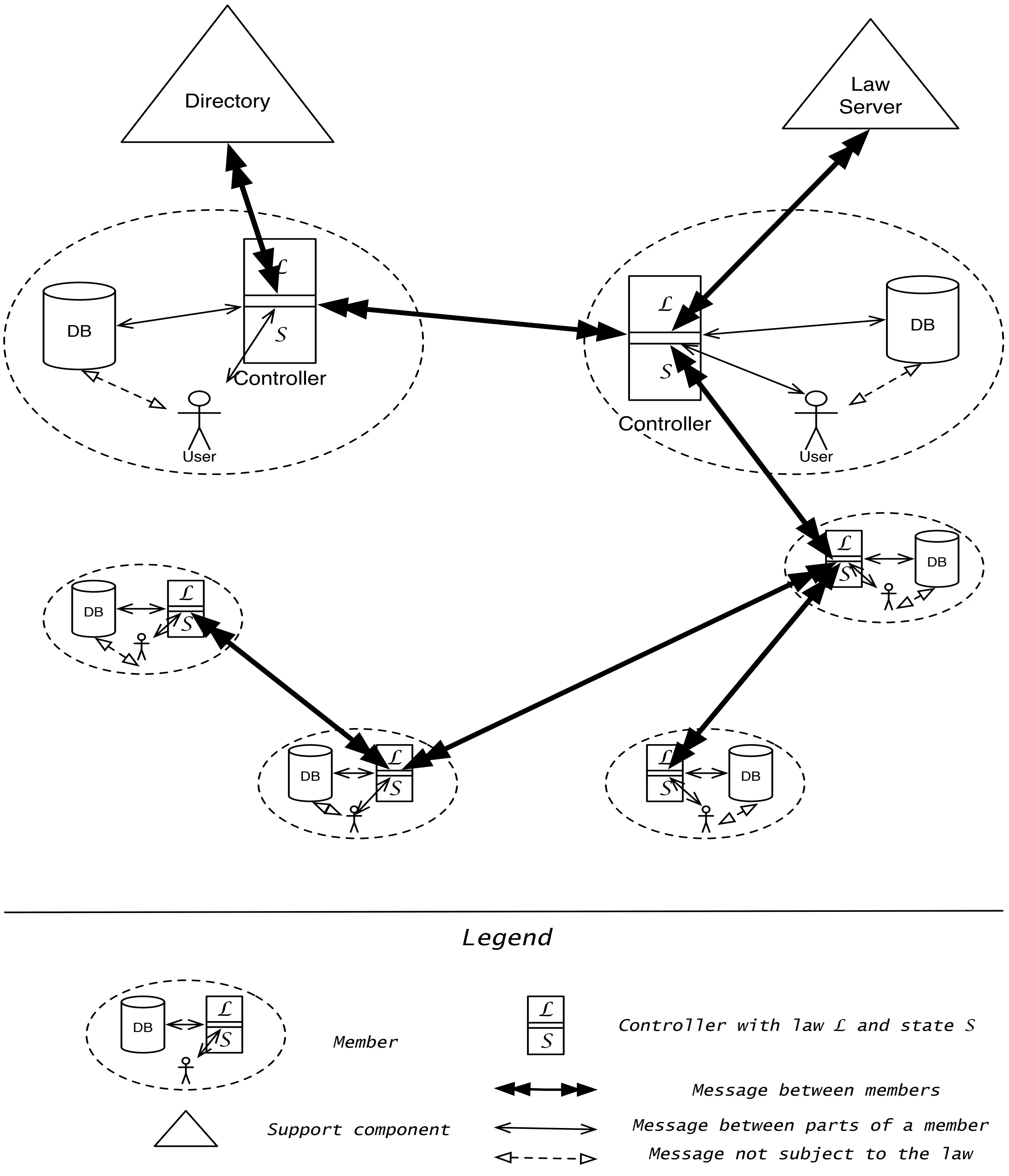
Directory
Law Server
DB
$\mathcal{L}$
S
Controller
User
Legend
Member
Controller with law $\mathcal{L}$ and state S
Message between members
Support component
Message between parts of a member
Message not subject to the law

Figure 3. The Anatomy of a DOSC Community

any. Finally, the user should *adopt* this controller as its mediator.

Note, however, that the adoption is governed by law $\mathcal{L}_C$, which may require, among other things, certain certificates to be provided by the user. If the user does not satisfy the requirements of law $\mathcal{L}_C$ then the adoption will fail. This is one way for the law to control the membership of a given community.

5.3. The Operation of a DOSC-Community

Consider a member x of a community C sending a message m to another member y. The message first arrives at the controller of x, that operates under law $\mathcal{L}_C$. These controllers would then carry out the ruling of law $\mathcal{L}_C$, which can mandate the execution of any number of the following kind of actions: (a) change its own state in some way; (b) communicate with the database of x; (c) send the message m, or some other message, to the controller of the original target of x; and (d) send some other messages to the controllers of some other members, or to some of the support components of the community. Among other things, this means that members of a community interact with each other via their controllers, and the controllers communicate with each other.

Figure 3 may help understanding the situation. This figure depicts several members, represented by ovals, each of which encloses the three components of a member. The members interact with each other via the thick arrows, involving a pair of mediators (controllers); and members interact via their own mediators with components of the support, which may not be interacting via LGI, and thus do not have their own mediators. The component parts of a member interact with each other as depicted, while the doted arrow represent the unregulated access that a user has to its own database.

It is worth pointing out here that LGI provides an important *trust modality* which is critical to this model. This trust modality is called *law-based trust*, or simply *L-trust*, and can be introduced, broadly, as follows: any pair of interacting LGI-controllers can identify, cryptographically, each other as genuine controllers, and can identify the law, under which their interlocutors operate. One consequence of this is that the law $\mathcal{L}_C$ of the given community C can be written so that members of C can interact only with other members of C. Now, *L-trust* can be defined as follows: *members of a community C can trust each other's interactive behavior to comply with their common law* $\mathcal{L}_C$.

Another important observation about the behavior of a community under this model needs to be made: the ruling of a law for a given event that occurs at a controller depends on the state of this controller, which may be different for different members. This difference can come from some certificates submitted by the user to its controller, which may authenticate the role of the user in the organization in question. And the state may change dynamically in response to some interactive activity of the community. For example, the manager of the community under our *WP* community, may be allowed by the law of *WP* community to transfer its managerial baton to some other member, which would then be able to send `leave` messages, introduced in Section 3. In other words, *the members of a community C may not be equal under its law* $\mathcal{L}_C$.

5.4. On the Analysis of Networks

In the context of OSN, a *network* means a graph generated by relationships between members, such as *following* in Twitter, and *friend* in Facebook. Such relationships can be easily represented in a DOSC by suitable attributes in the profile of members. Unfortunately, the analysis of the resulting graph is very hard in our context because it is highly distributed. However, if such a graph is not considered very sensitive—for example, if members would not mind if the information about whom they *follow* would be revealed to the public—then analysis of such a graph can be facilitated in the following way.

First, one builds into the support of the community a service called *Gr*, say. Second the law of the community should be written to ensure that *Gr* would be notified whenever the *following* relation (for example) is established or removed. So, the entire graph would be represented in the *Gr* service, and it can be analyzed fairly easily by it.

And note that the use of such a central service as part of a decentralized community does not make the community significantly less scalable. Because communication with *Gr* is done relatively rarely, and because it is essentially off line—since *Gr* itself, which receives all these messages, is not part of the community itself.

Of course, if the graph in question is too sensitive to be placed in a central service, this analysis cannot be done. But this would not be a big loss for many, if not most, DOSC communities. Because despite the apparent importance of network analysis in huge OSNs like Facebook and Twitter, it may have little or no importance for different kinds of OSNs, such as support groups. And it has only a marginal use for small or mid-size OSNs.

6. On an Implemented Case Study

This is a partial description of an implemented case study, which is meant to serve as a proof of concept of the DOSC architecture. It also serves here as a concrete example of the rather abstract model of DOSC provided above. This case study deals with the *WP* community, broadly introduces in Section 2.2; which is designed to operate in the context of a large and geographically distributed enterprise E, providing a micro-blogging for its employees. Some parts of this study are described in Section 7. *WP* has been tested with slightly more than two hundred, mostly simulated, members. We start this section with a presentation of the *support* of the *WP* community; and then, in Section 6.2, we describe various structural and behavioral aspects of it. Finally, we introduce in Section 7.1 more advanced capabilities of our *WP* community.

6.1. The Support of the *WP* Community

As mentioned in Section 5, a DOSC-community may require various services. Our *WP* community employs the following three services: a law-service (*LS*), a certification authority (CA), and a *naming service*, which we call the *secretary* of this community.

The law-service, which has been introduced in Section 5, maintains the law of this community. The CA issues digital certificates to the employees of E, certifying some of their attributes, such as their unique names within E, the role they play within this enterprise, etc. Finally, the secretary plays several roles in this community. First, it receives and maintains the certified name of every new member of *WP*, as we will see in Section 6.2. Second, it ensures that a given employee can be a member of *WP* just once. And third, the secretary serves as a *location service*, which is available for use by all members to *WP*. We do not show here the detailed implementation of such a secretary, and its use. But such details are provided by [30], where we also show how a secretary can implement pseudonymous naming structure.

6.2. On the Structural and Behavioral Nature of the *WP* Community

We start in Section 6.2 with some comments about the pseudo code we use for describing the law that governs this community. In Section 6.2, we discuss how a user becomes a member of the *WP* community and its groups, how it configures its profile, and how a member is removed. Section 6.2 discusses one of the forms of communication between members, and how it is regulated.

On the Description of the Law of the *WP* Community. Here we describe the law $\mathcal{L}_{WP}$ of the *WP* community via a pseudo-code consisting of *event-condition-action* rules. This informal code is fairly close to the the Prolog-based law-language of LGI. The *event-condition-action* rules that constitute the pseudo-code used below have the form:

UPON <e> **IF** <c> **DO** <[o]>

where e is an interactive event that occurs at one of the members of the community—or, more precisely, at the controller of this member; c is the condition of this rule, defined over the event itself, and over the state of the controller at hand; and [o], the action, is a list of one or more primitive operations. These rules are evaluated from top down, until the condition of one of them succeed—the action o of this rules is the ruling of this law.

Also, the following notations are used in this pseudo-code: (a) `sent(x, m, y)` denotes a *sent* event, namely the sending the message `m` by `x` to `y`; (b) `arrived(x, m, y)` denotes an *arrived* event, namely the arrival at agent `y` of message `m` from `x`; (c) `forward(x, m, y)` denoted the primitive operation that forwards message `m` from `x` to `y`; (d) `deliver` represents a permission for the actor to accept the message arriving at it; (e) `add(t)` operation adds term `t` to the control-state; and (f) `remove(t)` operation removes from the control-state a term that matches `t`, if any.

The law $\mathcal{L}_{WP}$—described via this pseudo code—is split into several parts, according to their functionalities. For the sake of brevity, we only discuss the laws of some of the functionalities of *WP* discussed here.

Member Profile and Membership Control. What we call a *profile* of a member is a set of attributes maintained in its control-state, that is, the state maintained by the controller of this member. All these attributes are visible to the law of this DOSC, and can be made visible to other members of the DOSC, subject to its law. We distinguish between three types of these attributes: (a) *Authenticated Attributes*; (b) *Discretionary Attributes*; and (c) *Controlled Attributes*. The *Authenticated Attributes* are those provided by the certificate used by an employee for joining this community. They include the name of the employee, its role, and the group (or groups) to which it belongs, etc. These attributes are not mutable, and can thus be easily used for indexing. But we have implemented here only name-based index, provided by the secretary. The *Discretionary Attributes* are those that a member can define and modify at will, to be visible by all members of the DOSC—they may contain such things as the interest or expertise of the member. Finally, the *Controlled Attributes* are those whose very existence, and the manner they are defined and updated is governed by the law—they include, in particular, a *follower* list and a list of last ten posts the member published.

Now, to join the community, a member needs to adopt a controller under law $\mathcal{L}_{WP}$. Rule $\mathcal{R}1$ allows a user to join the community by presenting a certificate signed by the CA employed by the enterprise in question, to authenticate its employees. Once certificate is verified by the controller, the set of attributes will be inserted into the user's profile. An example of an attribute is *role(manager)*. The certified name gets sent to the secretary of the naming service. Rule $\mathcal{R}2$ enables a member to add discretionary attributes to its profile, and Rule $\mathcal{R}3$ enables their update.

Rule $\mathcal{R}4$ enables a member to add an attribute called *filter* to its control-state. As we shall see below, the filter will provide the means for the member in question to block a specified set of members from following him.

```
R1. UPON adopted(X,cert(issuer(ca),
        subj(X),attr(A)))
        DO[ add(A), forward(X, A(name(N)),
        Secretary)]

R2. UPON sent(X,
        addProfile(Attribute(Value)),X)
        IF ¬ (Attribute in
        reservedAttributes)
        DO[add(Attribute(Value))]

R3. UPON sent(X,
        updateProfile(Attribute(Value)),X)
        IF ¬ (Attribute in
        reservedAttributes)
        DO[remove(Attribute),
        add(Attribute(Value))]

R4. UPON sent(X,
        addFilter(Attribute(Value)),X)
        DO[add(filter(Attribute(Value)))]

R5. UPON sent(X,#leave#,Y)
        IF role(manager)@p_x DO[forward]

R6. UPON arrived(X,#leave#,Y)
        DO[Quit]
```

Figure 4. Law $\mathcal{L}_{WP}$: Member's Profile and Membership Control

Finally, rules $\mathcal{R}5$ and $\mathcal{R}6$ regulate the removal of members from the community. Rule $\mathcal{R}5$ shows that only a manager can remove a member from the community, by sending the message *leave* to it—which, according to Rule $\mathcal{R}6$, would cause the removal of this member from the community. Non-managers are not allowed to to send the messages *leave*.

Communication. There are three modes of communication in this community: direct messaging, post/follow and narrowcasting. Direct messaging allows members to send messages to each other when specifying the address of the receiver as the destination. Post/follow is an analogy to Twitter's tweet/follow mechanism, where members can subscribe to another member and get notification when there is a new post. Narrowcasting is a mechanism for sending a message to a group of members whose profile satisfies a specified condition. All messages and post in this system have a *type* associated with them, which is analogous to the concept of *hashtag* in twitter. For the sake of simplicity, we discuss here only the details of post/follow, and we discuss narrowcasting in Section 7.1—we do not discuss in this paper the simplest of these modes of communication, i.e., direct messaging.

The control over communication via post/follow has two complementary parts: *global* and *local*. The global control is imposed on every member of the community, but can be sensitive to the profile of members, while the local control is discretionary to each member. We discuss both controls below, and the law that establishes them.

Global Control The global control over post/follow is imposed on both `posting` and `following`. The control over `posting` is on what types of posts members can send. For example, only managerial staff can send posts with type *management*.

The control on `following` regulates who can follow whom. Essentially, it is defined in the law by a constraint on the profiles of publisher and follower. An example of such global policies is that only the members from a same group can talk to each other. The problem is that there is no single place where these profiles can be evaluated because of the decentralization. To solve this problem, our law forces every `following` request to include the profile of the follower. And then the constraint will be evaluated at the publisher side. The example we just mentioned can be achieved by checking the profiles of the publisher and follower and rejecting the `following` request if the two members are from different groups.

Local Control If one does not want to be followed by certain members, it can block the `following` requests from them. To achieve this, a member can add a term *filter* to its control-state, which is a local constraint on the profile of the would be follower. Whenever a member f sends a `following` request to a member s, it will be forced to attach its profile p_f along with it. When the request arrives at the publisher's controller, f will not be added to s's `follower` list if its profile does not satisfy the filter.

The Law The rules of law $\mathcal{L}_{WP}$ that implements these provisions are defined in Figure 5 and described below.

According to Rule $\mathcal{R}7$, any one can send a `following` request to any member. The controller will attach its profile to the request. By Rule $\mathcal{R}8$, when the request arrives at a member, the controller checks whether the global constraint, denoted by G in the law, on the profiles of publisher and `following` requester is satisfied. Then the controller will also check whether there is a *filter* in its profile. If there is none, the controller can add the requester to the `follower` list. If there is a *filter*, the controller will examine whether the filter on the profile of the requester. If both conditions are satisfied, the controller will add the requester to the `follower` list.

```
R7. UPON sent(X,requestFollowing,Y)
        DO[forward(X,requestFollowing(p_x),
        Y)]

R8. UPON
        arrived(X,requestFollowing(p_x),Y)
        IF G(p_x, p_y) and ( filter(F)@CS
        and F(p_x) )
        DO[updateFollowerList]

R9. UPON sent(X,post(P),X)
        IF typeof(P) = ¬#management#
        or ( typeof(P) = #management# and
        role(manager)@p_x )
        DO[updateProfile(lastTenPosts(P)),
        updateDB(P),
        forward(X,P,followerList)]

R10.
    UPON arrived(X,P,Y)
        DO[deliver]
```

Figure 5. Law $\mathcal{L}_{WP}$: Communication

In Rule $\mathcal{R}9$, when a member wants to send a post to its followers, the controller will read its `follower` list and send the post to each of them. It will also update its database and an attribute called *lastTenPosts* in its profile. A management post is allowed to be sent only when the publisher has the attribute *role(manager)* in its profile. When the follower receives the post, according to the Rule $\mathcal{R}10$, controller will deliver the post to it.

7. Advanced Aspects of the Concept of DOSC

This section complements both our case study in Section 6 and the basic model of DOSC introduced in Section 5—-which were omitted from previous discussions for the sake of simplicity.

First, in Section 7.1 we show how to carry out privacy-preserving narrowcasting and profile-based search in DOSC—which have been fully implemented in our *WP* case study. These important capabilities require no more than the basic model of DOSC. Then we introduce two extensions of the basic DOSC model, whose implementation is beyond the scope of this paper. The first of these, in Section 7.2 enables the handling of very complex DOSC-communities, as well as families of such communities; and the second, in Section 7.3 enables the changing of the law of a community, while the community continues to operate.

7.1. Privacy-Preserving Narrowcasting and Profile-Based Search

By the term narrowcasting we mean here the delivery of a message to a subset of the membership of a community, consisting of the members whose profile satisfies a given condition. We define a *narrowcast* as a pair $\langle M, C\rangle$, where M is the messages to be delivered to every community member x, whose profile p_x satisfies condition C. This is obviously very useful mode of communication, particularly for fairly large OSNs, where members are generally not familiar with most of their peers, but may knows what *kind of people* they wants to communicate with. Moreover, narrowcasting can be used for profile-based search, in the following way: the message M delivered to the group of peers whose profile satisfies condition C can be a request to all of them to identify themselves via a return message.

It is easy to provide for narrowcasting in centralized OSN, because its host has direct access to all profiles of the members of a community in question. But it is problematic under decentralized OSN, whose profiles are maintained locally at each member. Yet, narrowcasting can be accomplished reasonably well even in a decentralized OSN via what is called *gossip* (or *flooding*) protocol. We will not describe here the Gossip protocol in details, except of saying that it is analogous to the real life process of gossip. But the following aspect of this protocol is important for what follows. Although the message M of a narrowcast $\langle M, C\rangle$, is to be delivered only to its intended targets—i.e., to the the community members whose profile satisfies condition C—practically all members of the community need to participate in the transfer of this message to its targets. We call these non-target participants in the gossip of a given narrowcast its *conveyors*, as their role is to help in the transmission of message M to its various targets.

The gossip protocol has two well known drawbacks, which are not very serious for its use in OSNs. First, gossip is not a very efficient way to do either narrowcasting or broadcasting. But the experience of Gnutella [21] with gossip, and our own experience with it [15] demonstrated that if the gossip protocol is carried out correctly by all its participants, its speed is sufficient for human interaction, even for fairly large communities with tens of thousands of members. Second, gossip-based narrowcasting is likely to miss a small percentage of its targets. But this is not considered a serious problem for many applications.

There are, however, two additional problems with gossip-based narrowcasting, which cannot be addressed under the DOSN architecture, but can be addressed under DOSC, because of its ability to impose constraints on the interaction between its members.

One of these problems is that the gossip protocol can be seriously undermined by the malicious or inadvertent misbehavior of even one of its participants. But as we have shown in [15] this problem can be addressed by defining a key part of this protocol via an LGI law, which is imposed on all participants in the gossip process. The second problem is that it poses a serious risk to privacy. We describe this risk below, along with a way to counter it.

The Risk to Privacy posed by gossip-based Narrowcasting, and its Mitigation: The risk to privacy in question is that once a conveyor gets a narrowcast $\langle M, C\rangle$, to convey to others, *it can read it itself*, although a narrowcast is intended only for its targets, and not for its conveyors. This is a massive privacy violation as the number of conveyors for a given narrowcast is likely to be far higher than the number of its targets; and all of them can get information not intended for them.

But this problem can be solved under DOSC, because under this architecture, it would be the *mediator* part of the conveyor that gets the narrowcast, not its user. And the mediator is a trustworthy LGI controller that operates subject to the law $\mathcal{L}$ of the community in question. So, one can prevent this privacy violation by writing this law to have a conveyor to just convey this message according to the gossip-protocol, but not to deliver it to its own user. Law $\mathcal{L}_{WP}$ of the *WP* community, introduced in Section 6, supports narrowcasting in this way, although this part of $\mathcal{L}_{WP}$ has not been discussed in that section. A simplified version of the part of $\mathcal{L}_{WP}$ that handles narrowcasting is displayed in Figure 6.

```
R11.
    UPON sent(X,narrowcast(M,C),X)
        DO[forward(X, narrowcast(M,C,p_x),
        X)]

R12.
    UPON arrived(X,narrowcast(M,C,p_x),Y)
        IF C(p_y)
        DO[deliver,
        forward(Y, narrowcast(M,C,p_x),
        followerList)]

R13.
    UPON arrived(X,narrowcast(M,C,p_x),Y)
        IF ¬C(p_y)
        DO[forward(Y, narrowcast(M,C,p_x),
        followerList)]
```

Figure 6. Law $\mathcal{L}_{WP}$: Narrowcast

Here is how narrowcasting operates under the this simplified part of $\mathcal{L}_{WP}$. First, a narrowcast $\langle M, C\rangle$, is initiated by a member x_0 sending the message `narrowcast(M,C)` to itself, and Rule $\mathcal{R}11$ attaches the profile of the initiator x_0 to this message.

Second, the arrival of this message at any member y, who may be the initiator x_0, is handled by Rule $\mathcal{R}12$ and if this rule fails, then by Rule $\mathcal{R}13$. Rule $\mathcal{R}12$, succeeds if condition $C(p_y)$ is satisfied, that is, if y is a target. The ruling of this rule will be to deliver the message to y, and then to "gossip" it to its set of followers (actually to a relatively small part of the followers, but we are simplifying here). Alternatively, Rule $\mathcal{R}13$ will be evaluated and will succeed because now the the condition $C(p_y)$ is not satisfied, and its ruling is *not to deliver this message to* y, because y is not a target, but to gossip it farther.

Note, however that this partial version of the treatment of narrowcasting is oversimplified to the point of being incorrect. It is incorrect because it does not have any way of stopping the gossip process. This is provided by the actual gossip protocol, which is part law $\mathcal{L}_{WP}$; this protocol also ensures that the narrowcasting would not overwhelm the whole community. There are two further aspects of law $\mathcal{L}_{WP}$ which are missing here. One is the imposition of a global constrain on who can target whom by narrowcasting. Second this law enables the arrival of a narrowcast at its nominal target to be blocked by the local filter defined by his target.

Finally, one has to take into account the unlikely possibility that a controller t serving as the mediator of member x has been corrupted so that would deliver to its user all narrowcasts arriving at it, whether x is their target or not. In this way the user of x would siphon in practically all narrowcast communication between members of the community, which could be a massive violation of privacy, of practically all members of the community.

This risk to privacy can be mitigated by systematically and periodically replacing the controller of every member of a community, with a fresh controller—which can be done while the community operates. (The code of the replaced controller can then be refreshed, and these controllers can be reused to serve other members. In this way, a corrupt controller would have only a relatively short span of time for getting illegal messages.

7.2. Hierarchical Organization of DOSCs

Communities under our basic DOSC model are *monolithic*, in a sense that their structure is defined by a single law. Such a law may distinguish between members based on their profile. For example, the law of the *WP* community, uses this method to make distinctions between its members based on the groups to which they belong. But it could be useful for large and heterogeneous community to provide its subgroups with a limited of autonomy—i.e., the freedom to define their all laws, and to change them at will, while conforming to the law of the community at large. Such a capability is provided by the concept of *conformance hierarchy* of LGI [4], and described broadly in Section 7.2. We then describe briefly two different, and very beneficial ways in which this concept can be applied to DOSC. Section 7.2 explains the usage of law-hierarchies for multi-group DOSCs; and Section 7.2 explains its use for forming families of DOSCs.

The Concept of Conformance Hierarchy. LGI enables the organization of a collection of laws into what is called a *conformance hierarchy*. This is a tree of laws, rooted by law called $\mathcal{L}_R$, in which every law, except of the root $\mathcal{L}_R$ itself, conforms transitively to its superior laws, in a sense to be described below. Moreover the conformance relation between laws is inherent in the hierarchy, requiring no extra validation. For a formal definition of such hierarchy of laws, and a detailed example of its use, see [4]; here we provide just an informal introduction of this concept.

The Nature of Conformance of LGI-Laws: Under access control [5, 14], the conventional view of conformance between policies is as follows: *policy P′ conforms to policy P if and only if P′ is more restrictive than P, or equal to it.* But this simple view of conformance would not do for LGI-laws, for the following reason: The ruling of an LGI-law is not confined to a decision whether to approve or reject an action by an actor; it can also require some other actions to be carried out in response to an event—such as changing the state of the acting agent in a specified manner, or changing the message being sent, and its target,in some way. And it is generally not meaningful to ask if one such action is more or less restrictive than another. So, instead of using a uniform definition of conformance, based on restrictiveness, LGI lets each law define what it means for its subordinates to conform to it. This is done, broadly, as follows.

A law that belongs to a conformance hierarchy has two parts, called the *ground* part and the *meta* part. The ground part of a law $\mathcal{L}$ imposes constraints on interactive behavior of the actors operating directly under this law—it has the structure defined by Formula 1. While the meta part of $\mathcal{L}$ circumscribes the extent to which laws subordinate to $\mathcal{L}$ are allowed to deviate from its ground and meta parts. In particular, this allows a law, anywhere in this hierarchy, to make any of its provisions *irreversible* by any of its subordinate law, by not permitting any deviation from it, by any of its subordinate laws. But it can also permit subordinates to either weaken or strengthen some of its own provisions.

One application of such conformance is setting out defaults. For example, the root law $\mathcal{L}_R$ may prohibit all interaction between actors, while enabling subordinate laws to permit such interaction, perhaps under certain conditions. Alternatively, law $\mathcal{L}_R$ may permit all interaction, while enabling subordinate laws to prohibit selected interactions.

On the Structure and Formation of a Conformance Hierarchy of Laws: A conformance hierarchy H is formed incrementally via a recursive process described informally below. First one creates the root law $\mathcal{L}_R$ of H. Second, given a law $\mathcal{L}$ already in H, one defines a law $\mathcal{L}'$, subordinate to $\mathcal{L}$, by means of a law-like text called *delta*, denoted by $\Delta(\mathcal{L},\mathcal{L}')$, which specifies the intended differences between $\mathcal{L}'$ and $\mathcal{L}$. Now, law $\mathcal{L}'$ is derived dynamically from law $\mathcal{L}$ and $\Delta(\mathcal{L},\mathcal{L}')$, essentially *by dynamic consultation*, as described informally below.

Consider the special case involving the root law $\mathcal{L}_R$, and its subordinate law $\mathcal{L}_s$ derived from $\mathcal{L}_R$ by the delta $\Delta(\mathcal{L}_R,\mathcal{L}_s)$. And consider an agent x operating under law $\mathcal{L}_s$. Now, when an event e occurs at an agent x it is first submitted to law $\mathcal{L}_R$ for evaluation. Law $\mathcal{L}_R$ may consult the delta $\Delta(\mathcal{L}_R,\mathcal{L}_s)$ of $\mathcal{L}_s$ before deciding on its ruling—although it may, instead, render its own ruling, not involving the delta. If consulted, the delta will do its own evaluation of this event, and will return its *advice* about the ruling to law $\mathcal{L}_R$. $\mathcal{L}_R$ would render its final ruling about how to respond to event e, taking the advice of the delta into account—but not necessarily accepting it, because this advice might contradict the meta part of $\mathcal{L}_R$. In this way, *the dynamically derived law $\mathcal{L}_s$ naturally conforms to its superior law $\mathcal{L}_R$, requiring no further verification.*

A notable property of the hierarchical organization of laws is that interacting agents operating under laws in a common hierarchy can identify the position of each other's laws within this hierarchy.

On the Usage of Law-Hierarchies for Multi-Group DOSCs. Consider a DOSC-community *WP′*, which is similar to community *WP* presented in Section 6, except that it is governed by a hierarchical law-ensemble H like the one depicted in Figure 7. Employees who belong to group $g1$, for example, are meant to operate subject law $\mathcal{L}_{g1}$, while people who belong to subgroup $g11$ are meant to operate under law $\mathcal{L}_{g11}$; and people who belong to the enterprise in question, but not to any of these groups are mean to operate subject to the root law $\mathcal{L}_R$. These laws may accept only the people who certify themselves as belonging to the enterprise at large, and to a specific group, if any.

Law $\mathcal{L}_R$ of H would be written to establish the various provisions that are to be shared by all groups. Such as membership control in the community at large, and the various modes of communication like direct messaging and narrowcasting. It might also regulate the inter-group communication. The laws of a particular group, can provide control over the membership in this group, as well as over its intra-group communication.

Note that the creation of a new group to *WP′* can be done very flexibly—unless this is prohibited by the root law—without changing anything else in this community. Every such addition requires the creation

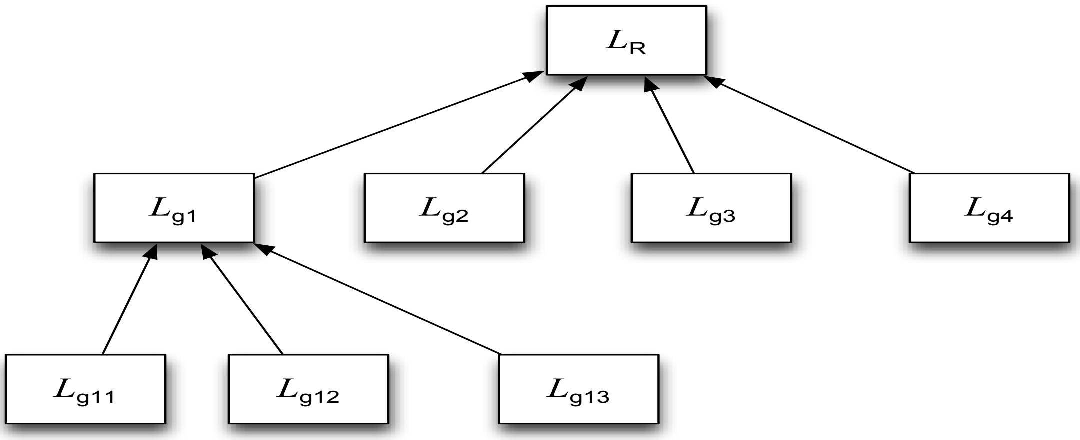

Figure 7. A Hierarchical Law-Ensemble that Governs a Complex DOSC-Community

of a new law, as subordinate of law $\mathcal{L}_R$. The changing of the law of a group is similarly flexible.

A very important consequence of such hierarchical organization of laws is that it facilitates seamless interoperability between different groups, provided that the laws of the respective groups permits exchange of messages between them on the basis that they both conform to the root law. This would be possible, however, only if such interoperability is not prohibited by the root law. For much more about this interoperability, and its advantages, the read is referred to [4].

Supporting Families of DOSC-Based Communities. Consider a family of DOSCs—such as support groups of people who suffer from various diseases—which may have nothing to do with each other but have similar structures. For example they may require similar capabilities, such as narrowcasting, and may used the same mode of addressing, such as anonymous. But despite such similarities, the individual DOSCs may want to establish their own structure. In particular, it is likely that a DOSC belonging to a given family may want to establish its own membership control.

This can be easily accomplished via a conformance hierarchy H, whose root law includes what is common to a given family of DOSCs, while each individual DOSC in this family has its own provision formulated by a law subordinate to the root of H. This arrangement can be facilitated by having the entire hierarchy H maintained by a single publicly available law-server. The structure and API of such a law-server would depend on the nature of the family of DOSC that it serves.

7.3. On the Evolution of the Law of a DOSC, while the Community Operates

Like any other piece of software, the law of a DOSC is bound to change. The problem addressed here is how to enable safe change of the law of a given community while this community continues to operate. We refer to such a change as being *in vivo* (i.e., carried out in a living organism, as it where). The importance of such evolution for a long lived system that must operate continuously is self evident.

The main difficulty with carrying out in-vivo evolution of laws stems from the decentralized nature of our enforcement mechanism. This means, in particular, that in order to update a law $\mathcal{L}$ to $\mathcal{L}'$ one needs to update the law in all controllers associated with actors operating under law $\mathcal{L}$ in a *virtually atomic* manner. By "virtually atomic" we mean, in particular, that during the distributed update process no functionally meaningful messages should be exchanged between agents operating under different versions of the law. We have solved [23] this problem for a group of agents operating under a single law. But the in vivo evolution of laws belonging to an hierarchical law ensemble is still an open problem.

8. On the Performance of DOSC, and its Drawbacks

Performance: Recall that in a paragraph called "performance" in Section 4.2 we reported that the the overhead incurred by LGI controllers is relatively small—often smaller than the overhead incurred by control mechanisms such as XACML. And given that the computation time of a controller is about 50 microseconds, depending on the complexity of the law in question, it is quite negligible for communication over WAN. Moreover, the enforcement of laws is scalable, being decentralized.

That said, certain operations such as narrowcasting and profile-based search, which require access to all members of the community, are clearly less efficient under DOSC than under the centralized OSN. And they are not scalable under DOSC. Yet for medium and small communities, the narrowcasting generally concludes virtually instantaneously, from the viewpoint of a human user. We have using DOSC-like control over narrowcasting in Gnutella [21], which had close to 20 thousands of distributed members, and the response was almost instantaneous. Of course this would not be the case with a community of the size of the membership of Facebook.

Drawbacks: We see two limitations of DOSC, the first of which can be viewed as an advantage, and the second is fairly minor. The first limitation is that unlike a centralized OSN, DOSC does not lend itself to an analysis of everything that happened in the community in the past, because the history of the communication between community members is not maintained. This is a problem for researchers who wish to study such communities, but we view it as an advantage to most community members.

The second limitation of DOSC is for communities that requires its members to maintain a database of their past contributions, as is required by our case study of the *WP* community. A centralized OSN would keep such contribution on their web site. But DOSC requires each member to have its own database maintained somewhere, and be more or less always available. But the easy and cheap availability of space on the cloud makes this limitation of little importance. Besides, some DOSCs—like the medical consultation community *MC* described in Section 2.1, which has also been implemented as a DOSC—are just conversational, and do not require any databases.

9. Related work

The concern about the security issues of centralized OSNs motivated several attempts to decentralize them, creating several versions of DOSNs. These include PeerSoN[6–9, 24], Safebook[10–12], and LotusNet [2, 3]; as well as some others [1, 13, 16, 22, 25, 26, 29]. The basic idea underlying all these projects is that each member of the social networks keeps the data under its own control, instead of surrendering it to a central host. This is a necessary measure of decentralization, but it is not sufficient. As explain in Section 1, OSNs need to be governed by policies (or, in our terminology, laws) regarding their membership and the interactions between its members. But none of the variants of DOSNs known to us provides any means for establishing such global policies. Consequently, they are unable to provide privacy-preserving narrowcasting and profile-based search—capabilities that are very important for OSNs.

Moreover, the indexes provided by many of the DOSN variants, are based on *distributed hash table* (DHT), whose components are managed by the members of the DOSNs at hand. But these members are heterogeneous, and not particularly reliable or trustworthy, and any one of them can compromise the DHT, either inadvertently or maliciously, as has been argued in [27]. In fact, a DHT can be made much more secure under DOSC, if it is managed subject to an appropriate law, like the rest of a DOSC-community. We did not do it in this paper. Instead, we implemented our name-index via a central service, which can be made much more secure than DHT under DOSNs.

9.1. Relationships of this Paper to Previous Work by the Authors

This paper is obviously related to our previous work on LGI, but LGI is used here as a tool—enough said about that. In addition, there are three papers that are related to social networks, and one of them is a direct precursor to the present work.

The first of these papers [15] implemented essentially our medical consultation *MC* example of an OSN in Section 2.1. It was done in a decentralized manner, providing secure gossip-based communication, but without the mechanism used here for ensuring that conveyors do not read all messages. The second paper [30] implemented some aspects of our WorkPlace *WP* example of an OSN in Section 2.2. Neither of these papers introduced a generic model of decentralized OSNs—not surprising, perhaps, as these papers were written before the concept of OSN became popular, or before it existed. The third paper [28] is a direct precursor of the present one, as it introduced what we call here the "basic DOSC model."

The main novelty of the present paper, with respect to the three papers mentioned above, is the advanced model of DOSC introduced in Section 7. This model features three novel and very important capabilities for social networks, which would enhance both the security and the usefulness of decentralized OSNs.

10. Conclusion

The centralized nature of conventional OSNs poses serious risks to the security and privacy of information exchanged between their members. These risks prompted several attempts to create decentralized OSNs, or DOSNs. The basic idea underlying these attempts, is that each member of a social network keeps its data under its own control, instead of surrendering it to a central host; providing access to it to other members according to its own access-control policy. Unfortunately all existing versions of DOSNs have a very serious limitation. Namely, they are unable to subject the membership of a DOSN, and the interaction between its members, to any global policy—which is essential for many social communities. Moreover, the DOSN architecture is unable to support useful capabilities such as narrowcasting and profile-based search.

We described in this paper a novel architecture of decentralized OSNs—called *DOSC*, for "online social community". DOSC adopts the decentralization idea underlying DOSNs, but it is able to subject the membership of a DOSC-community, and the interaction between its members, to a wide range of policies—including privacy-preserving narrowcasting and profile-sensitive search. Moreover, DOSC's control over the interaction between its members is scalable. Furthermore, DOSC provides flexible supports for complex, multi-group, communities; as well as to families of distinct communities.

References

[1] S. M. A. Abbas, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips. A gossip-based distributed social networking system. In *Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, WETICE '09, pages 93–98, Washington, DC, USA, 2009. IEEE Computer Society.

[2] Luca M. Aiello and Giancarlo Ruffo. LotusNet: Tunable privacy for distributed online social network services. *Computer Communications*, December 2010.

[3] Luca Maria Aiello and Giancarlo Ruffo. Secure and flexible framework for decentralized social network services. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 594–599. IEEE, 2010.

[4] Xuhui Ao and Naftaly H Minsky. Flexible regulation of distributed coalitions. In *Computer Security–ESORICS 2003*, pages 39–60. Springer, unknown, 2003.

[5] A. Belokosztolszki and K. Moody. Meta-policies for distributed role-based access control systems. In *Proc. of the IEEE 3rd International Workshop on Policies, Monterey, California*, pages 106–15, June 2002.

[6] Oleksandr Bodriagov and Sonja Buchegger. Encryption for peer-to-peer social networks. In *SocialCom/PASSAT*, pages 1302–1309. IEEE, 2011.

[7] Oleksandr Bodriagov and Sonja Buchegger. P2p social networks with broadcast encryption protected privacy. In *Privacy and Identity Management for Life*, pages 197–206. Springer, 2012.

[8] Sonja Buchegger and Anwitaman Datta. A case for p2p infrastructure for social networks - opportunities & challenges. In *Proceedings of the Sixth international conference on Wireless On-Demand Network Systems and Services*, WONS'09, pages 149–156, Piscataway, NJ, USA, 2009. IEEE Press.

[9] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. Peerson: P2p social networking: early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, SNS '09, pages 46–52, New York, NY, USA, 2009. ACM.

[10] L. A. Cutillo, R. Molva, and T. Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *Comm. Mag.*, 47(12):94–101, December 2009.

[11] Leucio Antonio Cutillo, Refik Molva, and Thorsten Strufe. Safebook: Feasibility of transitive cooperation for privacy on a decentralized social network. In *WOWMOM*, pages 1–6. IEEE, 2009.

[12] LeucioAntonio Cutillo, Refik Molva, and Thorsten Strufe. On the security and feasibility of safebook: A distributed privacy-preserving online social network. In Michele Bezzi, Penny Duquenoy, Simone Fischer-HÃijbner, Marit Hansen, and Ge Zhang, editors, *Privacy and Identity Management for Life*, volume 320 of *IFIP Advances in Information and Communication Technology*, pages 86–101. Springer Berlin Heidelberg, 2010.

[13] Anwitaman Datta, Sonja Buchegger, Le-Hung Vu, Thorsten Strufe, and Krzysztof Rzadca. Decentralized online social networks. In *Handbook of Social Network Technologies and Applications*, pages 349–378. Springer, 2010.

[14] S. Godic and T. Moses. Oasis extensible access control. markup language (xacml), version 2. Technical report, Oasis, March 2005.

[15] M. Ionescu, Naftaly H. Minsky, and T. Nguyen. Enforcement of communal policies for peer-to-peer systems. In *Proc. of the Sixth International Conference on Coordination Models and Languages, Pisa Italy*, February 2004.

[16] Andreas Loupasakis, Nikos Ntarmos, and Peter Triantafillou. exo: Decentralized autonomous scalable social networking. In *CIDR*, pages 85–95, 2011.

[17] Naftaly H. Minsky. *Law Governed Interaction (LGI): A Distributed Coordination and Control Mechanism (An Introduction, and a Reference Manual)*. Rutgers, February 2006. (available at `http://www.moses.rutgers.edu/`).

[18] Naftaly H Minsky. Decentralized governance of distributed systems via interaction control. In *Logic Programs, Norms and Action*, pages 374–400. Springer, unknown, September 2012.

[19] Naftaly 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.

[20] S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security*, 3(2):85–106, May 2000.

[21] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100. IEEE, 2001.

[22] Daniel Sandler and Dan S. Wallach. Birds of a fethr: open, decentralized micropublishing. In Rodrigo

Rodrigues and Keith W. Ross, editors, *IPTPS*, page 1. USENIX, 2009.

[23] C. Serban and Naftaly Minsky. In vivo evolution of policies that govern a distributed system. In *Proc. of the IEEE International Symposium on Policies for Distributed Systems and Networks, London*, July 2009.

[24] Rajesh Sharma and Anwitaman Datta. SuperNova: Super-peers Based Architecture for Decentralized Online Social Networks. *Computing Research Repository*, abs/1105.0, 2011.

[25] Patrick Stuedi, Iqbal Mohomed, Mahesh Balakrishnan, Zhuoqing Morley Mao, Venugopalan Ramasubramanian, Doug Terry, and Ted Wobber. Contrail: Enabling decentralized social networks on smartphones. In *Middleware*, pages 41–60, 2011.

[26] Sebastian Tramp, Philipp Frischmuth, Timofey Ermilov, Saeedeh Shekarpour, and SŽren Auer. An Architecture of a Distributed Semantic Social Network. *Semantic Web Journal*, Special Issue on The Personal and Social Semantic Web, 2012.

[27] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. A survey of dht security techniques. *ACM Comput. Surv.*, 43(2):8:1–8:49, February 2011.

[28] Zhe Wang and Naftaly Minsky. Establishing global policies over decentralized online social networks. In *Proc. of the 9th IEEE International Workshop on Trusted Collaboration*, October 2014.

[29] Tianyin Xu, Yang Chen, Jin Zhao, and Xiaoming Fu. Cuckoo: towards decentralized, socio-aware online microblogging services and data measurements. In *Proceedings of the 2nd ACM International Workshop on Hot Topics in Planet-scale Measurement*, HotPlanet '10, pages 4:1–4:6, New York, NY, USA, 2010. ACM.

[30] Wenxuan Zhang, Constantin Serban, and Naftaly H. Minsky. Establishing global properties of multi-agent systems via local laws. In Danny Weyns, editor, *Environments for Multiagent Systems III, LNAI 4389*. Springer-Verlag, 2007.

[31] Dejin Zhao and Mary Beth Rosson. How and why people twitter: the role that micro-blogging plays in informal communication at work. In *Proceedings of the ACM 2009 international conference on Supporting group work*, GROUP '09, pages 243–252, New York, NY, USA, 2009. ACM.