Architectural Support for Trust Models in Decentralized Applications

Girish Suryanarayana, Mamadou H. Diallo, Justin R. Erenkrantz, Richard N. Taylor
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455
+1(949) 824-6429
{sgirish,mdiallo,jerenkra,taylor}@ics.uci.edu

ABSTRACT

Decentralized applications are composed of distributed entities that directly interact with each other and make local autonomous decisions in the absence of a centralized coordinating authority. Such decentralized applications, where entities can join and leave the system at any time, are particularly susceptible to the attacks of malicious entities. Each entity therefore requires protective measures to safeguard itself against these entities. Trust management solutions serve to provide effective protective measures against such malicious attacks. Trust relationships help an entity model and evaluate its confidence in other entities towards securing itself. Trust management is, thus, both an essential and intrinsic ingredient of decentralized applications. However, research in trust management has not focused on how trust models can be composed into a decentralized architecture. The PACE architectural style, described previously [21], provides structured and detailed guidance on the assimilation of trust models into a decentralized entity's architecture. In this paper, we describe our experiments with incorporating four different reputation-based trust models into a decentralized application using the PACE architectural style. Our observations lead us to conclude that PACE not only provides an effective and easy way to integrate trust management into decentralized applications, but also facilitates reuse while supporting different types of trust models. Additionally, PACE serves as a suitable platform to aid the evaluation and comparison of trust models in a fixed setting towards providing a way to choose an appropriate model for the setting.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures.

General Terms

Design, Security.

Keywords

Decentralization, Peer-to-Peer (P2P), Trust Management, Reputation, Software Architecture, Architectural Style.

1. INTRODUCTION

Unlike a centralized system which relies upon a central authority to coordinate the activities of each entity in the system, in a decentralized system, each entity is completely autonomous and responsible for its own individual behavior. In such systems, each entity, also called a peer, depends upon its interaction with other peers to make adequate decisions towards the realization of its individual goals. Furthermore, in an open decentralized system, peers, including those with malicious intentions, can enter or leave at any time, thus exposing the system to potential attacks.

As an example, consider an open decentralized peer-to-peer (P2P) file-sharing application, such as Gnutella [15], that allows peers to directly share files with each other. A Gnutella peer locates other peers 'nearby' and directly queries them for content it needs. These peers may in turn forward these queries to other 'nearby' peers. Results are then returned to the original peer who can download the file directly from a peer that has the requested content. In the ideal case, all peers are reliable and serve only good content. However, in the real world, attacks on file-sharing applications by malicious peers are common and well-known. These peers may offer files that are corrupted, or even worse, encapsulate viruses or trojans within them. Downloading and opening these files pose a significant risk to peers in the system. A study in January 2004 [27] reported that 45% of 4,778 executable files downloaded through the Kazaa [16] file-sharing application contained malicious code like viruses and Trojan horses. This finding clearly serves to emphasize the threats inherent to file-sharing applications and to decentralized systems in general.

In the absence of a centralized entity that may, for example, upon receiving information about a malicious peer, stop indexing the files shared by the malicious peer, in a decentralized system, each peer must itself adopt suitable measures towards safeguarding against such attacks. Trust management solutions serve as potential candidates for such measures. In particular, a trust management system helps peers establish trust relationships with other peers in the system, and these relationships help a peer determine the trustworthiness of other peers in the system. Trust management has thus become a necessary aspect of decentralized systems.

Decentralized trust management has therefore received much attention in recent years. Several trust and reputation models with varying objectives have been developed by researchers. However, very little attention has been devoted to the exploration of how these models can be composed into a decentralized application. The PACE architectural style [21] addresses this particular need.

PACE is an event-based architectural style and has been designed with trust management in mind. It provides explicit guidance and support towards the incorporation of different trust models into a decentralized application. PACE also addresses how peers can be architecturally designed for participation in decentralized applications.

This paper describes our efforts towards the evaluation of PACE. Our goals were to assess the effectiveness of the guidelines suggested by PACE for the design of decentralized trust management solutions, and to examine the degree to which PACE supports different kinds of trust and reputation models. Towards these goals, we have incorporated four different reputation-based trust models into a decentralized Crisis Response And Situation Handling system (CRASH) built in the PACE style. The CRASH system simulates a variety of governmental and non-governmental organizations cooperating to respond to emerging crisis situations. Our experiments with these four trust models using the PACE style have revealed that PACE not only supports the integration of different trust models but also promotes easy and effective integration of the trust models, and significant reuse of components. Additionally, these experiments helped underline the use of PACE as a suitable platform to evaluate and compare the behavior of different trust models under a fixed setting.

The rest of the paper is structured as follows. Section 2 discusses relevant related work and Section 3 summarizes the PACE architectural style. Section 5 describes our approach and in particular discusses threats due to decentralization, the CRASH system that was used in our evaluation, the four candidate trust models, and how these models were designed and integrated in the CRASH architecture. Section 5 discusses our observations and evaluation. The paper ends by drawing conclusions in Section 7.

2. RELATED WORK

A considerable number of trust models and algorithms have been developed to tackle the problem of decentralized trust management [11, 22]. Access-based systems such as Policymaker [3] and Keynote [4] use credentials and application-specific policies to regulate access to resources and services. These systems evaluate the credentials of resource requestors against application-defined policies in order to determine whether access to the resources can be permitted [7, 26].

Trust management systems such as the Distributed Trust Model and REGRET [1, 20] are based on reputations. Reputation-based systems enable a peer to evaluate the trustworthiness of another peer based on relevant reputation information obtained from other peers in the system [19]. These systems use different kinds of mechanisms to acquire reputation information and determine trust. For example, in XRep [8], a peer has to query for recommendations from other peers in order to determine whether a peer can be trusted. On the other hand, in NICE [17], each requesting peer needs to present cookies received from other peers in order to prove its own trustworthiness to a peer. Other trust models, such as NodeRanking [18] and Regret [20] in addition to the opinion of peers rely upon existing social relationships among peers to determine the reputation of peers. Research in trust management also includes work on algorithms for computing trust and reputation such as EigenTrust [14].

Existing trust architectures are either inapplicable to decentralized systems or do not serve to provide comprehensive and complete guidance towards decentralized trust management. Sultan [12] and XenoTrust [9] are trust management frameworks that rely upon a centralized server to aggregate trust information obtained from entities in the system. Similarly, the trust framework described by Gray et. al [13] provides a local algorithm for computing trust, but relies on a central authority. Hence these systems are unfit for decentralized applications. DPS [10] is an architectural style for specifically building access controlled software systems but does not consider decentralized trust and reputation.

Cahill et. al [5] identify trust components for the SECURE trust management framework and the control flow between the components. However, the trust calculator component used in their framework is application-specific and may not support the assimilation of third party trust models into the framework. hTrust [6] provides a simple trust management architecture specifically for mobile ad hoc systems and incorporates essential trust, application, and communication layers. NICE [17] provides a platform for implementing cooperative distributed applications over the Internet and facilitates the implementation of different trust algorithms based on reputation. However, hTrust and NICE do not provide a set of complete and effective guidelines that specify the necessary constraints on the structure of these components and their interactions. They thus fail to provide comprehensive architectural guidance for composing trust management.

3. PACE ARCHITECTURAL STYLE

PACE stands for the Practical Architectural approach for Composing Egocentric trust. The PACE architectural style addresses the concerns of trust management in open decentralized applications. PACE is designed to facilitate the incorporation of trust models into the architecture of a decentralized peer. PACE provides comprehensive guidance about the components that should be included inside a peer, their arrangement inside the peer, and their interactions.

PACE is an extension of C2 [23], an event-based architectural style. A C2 architecture is composed of components and connectors that are organized into layers. Components in a layer are only aware of components in the layers above and have no knowledge about components below. Components communicate with each other using two types of asynchronous event-based messages - requests and notifications. Requests travel up the architecture while notifications move down the architecture.

PACE imposes a set of additional constraints on the structure and behavior of components within the peer to address trust management. First, it identifies peers in the system through their digital identities, allowing for the possibility that a single user may pose as multiple peers by using multiple electronic identities. Trust information about each digital identity is separately determined and maintained irrespective of the physical identities it represents. Second, PACE makes a specific distinction between the internal beliefs of a peer and the beliefs communicated externally to it by other peers in the system. Such a distinction is important since the information received from other peers may possibly be faulty or incomplete. PACE, therefore, explicitly divides data storage between internal and external information repositories. Third, in order to facilitate the comparison of trust values, PACE requires that trust values be represented numerically, without imposing any constraint on their semantics. Fourth, PACE requires that trust between peers be not localized to a single component but be perceptible to all the components across the architecture. This is so that each component responsible for making local decisions can take advantage of this perceived trust. Trust relationships should be discernible to the components in the peer's architecture as well as published externally to other peers.

PACE divides the components of a peer's architecture into four layers: *Communication*, *Information*, *Trust*, and *Application*. Figure 1 depicts these layers and their associated components. The *Communication* layer is responsible for handling the communication between peers in the system. It is composed of three components: the Protocol Handler, Communication Manager, and Signature Manager. The Protocol Handler enables multiple network communications which are responsible for translating internal events into the format understood by the associated external protocol and vice-versa. The Communication Manager is responsible for the dynamic creation of protocol handlers while the Signature Manager is responsible for signing requests and verifying notifications. The Signature Manager digitally signs outgoing messages with the peer's locally configured private key, and embeds the signatures and corresponding public key within the message to enable transport over protocols that do not support digital signatures. At the receiving end, this included public key can be used by the Signature Manager to authenticate the message.

The *Information* layer stores data in the system. To separate the internal beliefs of a peer from those received from other peers, the *Information* layer is composed of two components. The Internal Information component stores request messages that originate from internal components while the External Information stores notification messages received from external peers. Data in the Internal Information is persistent in order to allow the peer to maintain a record of its prior actions and beliefs. The Internal Information component allows modification and queries for data only through requests to protect against unintentional distribution of data to other peers. It is also responsible for forwarding requests for transmission to the *Communication* layer.

The *Trust* layer incorporates components that enable trust management. This layer is composed of the Key Manager, the Credential Manager, and the Trust Manager. The Key Manager is responsible for generating public-private key pairs for message authentication and storing them in the Internal Information component. The Credential Manager manages the credentials of other peers that are stored in the Internal Information. It is also responsible for requesting needed public keys and revoking them upon receiving suitable notifications. The Trust Manager incorporates different trust and reputation models like those discussed in Section 2. It is responsible for computing and assigning trust values to messages received from other peers.

The *Application* layer encapsulates application-specific components. It includes the Application Trust Rules and the Application components. The Application Trust Rules encapsulates the rules that are needed to assign trust values based on application-specific semantic meanings of messages, and supports different dimensions

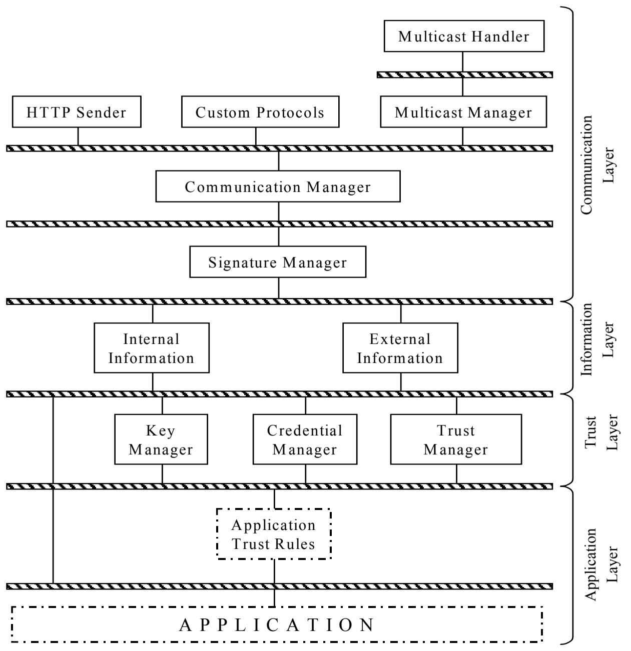


Figure 1. Peer Architecture in PACE

of trust relationships. The Application component represents local behavior of a peer and may include user interfaces. While components in the other layers can be reused across different applications, components in the *Application* layer are application dependent and hence not reusable across application domains. The application developer is expected to implement components for this layer depending upon the needs of the application.

PACE is only concerned with trust relationships between peers and not between the components comprising the internal architecture. Also as discussed earlier, PACE distinguishes between the internal beliefs of a peer and externally reported information. In particular, PACE does not trust data received externally from other peers and only trusts data originating internally. This distinction can be conveniently modeled through requests and notifications in C2. Thus, messages from external peers sent by the Communication layer to the other layers below in the form of notifications are not trusted implicitly while messages originating from components internal to the architecture in the form of requests are implicitly trusted.

It should be stressed that PACE does not require that all peers interacting in a decentralized application be built in the PACE style. Peers composed differently could still interact with those built using PACE. Similarly, PACE does not constrain the internal architecture of a peer beyond the PACE-specific parts. For example, components belonging to the *Application* layer could be built in other architectural styles or could be procured off-the-shelf.

4. APPROACH

In order to evaluate the applicability and the ease of assimilation of the PACE style to different trust models, we first selected four trust models, built implementations for them, and then used PACE to incorporate these implementations within a prototype Crisis Response And Situation Handling system (CRASH). We also subjected these four prototypes of the CRASH system to threat scenarios to observe whether integration with PACE adversely affected the capabilities of the trust models. In this section, we first briefly discuss these threats of decentralization, followed by a summary of the CRASH system, the design descriptions of the four CRASH prototypes, and a discussion of the threat scenarios.

4.1 Threats of Decentralization

Our previous work [21] outlined several threats of decentralization. In this domain, we limit our scope to the following subset of these threats. *Impersonation* refers to the threat caused by a peer that portrays itself as another in order to misuse the target peer's reputation or privileges. Another threat is due to *fraudulent actions* that are deliberately performed by a deceitful peer in order to harm or dupe other peers. For example, a peer may falsely claim to provide a particular service and charge other peers for that service. *Misrepresentation* happens when peers provide misleading information about their trust relationships with other peers. When a group of peers collude together in order to bring down the system, this threat is referred to as *Collusion*. The *Addition of Unknowns* threat refers to the problem of cold start. When a new peer joins the system, it is difficult to determine the trust relationship between the new peer and the existing peers.

4.2 CRASH System

Crisis management is an inherently decentralized application with a large number of benevolent participants, a potentially high number of malicious participants (such as lingering criminals or interfering spectators), and demand for fast information exchange. In serious crises, multiple agencies with multiple command centers are involved, and a multitude of teams individually tackle different parts of the crisis [24]. In such situations, information is scarce, unreliable, and individual teams operate largely independently while still needing to coordinate with each other. Moreover, even when a central plan is in place, it is possible that crisis responders may interact over unreliable ad hoc networks and perceived information may turn out to be incorrect. Thus, it is imperative that suitable countermeasures be in place to guard against malicious attacks that hamper communication or facilitate the dissemination of misleading and inaccurate information.

There are two categories of entities that must be considered in this particular domain: those that must make decisions based upon the available information and those that merely report information. The former category is the principal focus of our Crisis Response and Situation Handling system (CRASH). CRASH models a collection of governmental and non-governmental organizations cooperating in response to emerging situations in order to make decisions. For our evaluation, we modeled the following decision-making organizations: Police Department (PD), Fire Department (FD), Search and Rescue (S&R), Red Cross, St. Elsewhere Hospital (Hospital), a Charitable Organization (CO), and the Department of Public Works (DPW). The general public, as a non-decision maker, provides and relays information to CRASH entities. When information originates from the public (which is an instance of an unknown external event source), this information should be initially be treated with circumspect. Consequently, trust management plays a critical part in helping these decentralized entities make appropriate and timely decisions. For our evaluation purposes, we introduce the Media as an organization that not only provides information but also consists of disgruntled members who can secretly influence the decision-making ability of other organizations even during crises.

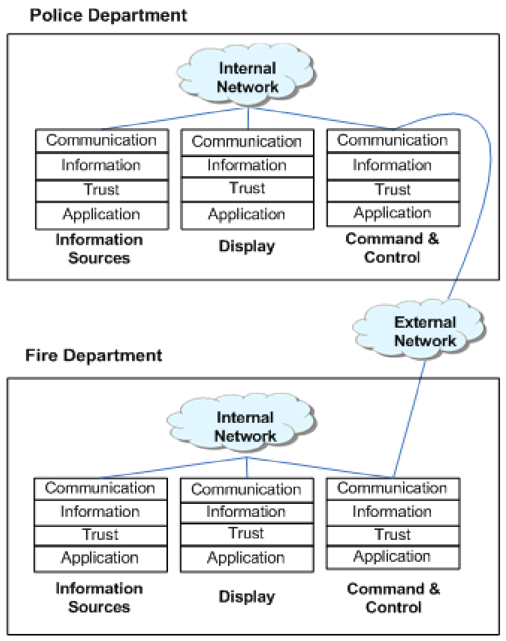


Figure 2. CRASH with Police and Fire Departments

Each CRASH entity or peer can be divided into three sub-system classes: Display, Information Gathering Sources, and Command and Control (C&C). The Display sub-system is responsible for visualizing the information currently known to the organization - such as deployment of resources and other vital information. Information Gathering Source sub-systems provide feedback and information to the entity's C&C sub-system - such as relaying reports from the public. These sub-systems are connected to the entity's C&C through internal ad hoc networks. Additionally, each entity's C&C center is also connected to the C&C center of other organizations - perhaps, again, through ad hoc networks (see Figure 2). An entity's C&C center is then responsible for aggregating data received from its information sources as well as information from other organizations. Ultimately, the C&C system is responsible for making decisions on behalf of the entity and conveying information and instructions to its affiliated resources.

4.3 Candidate Trust Models

In this section, we describe the four trust models that were used in our evaluation of the PACE architectural style. All the four models are based on the concept of determining trustworthiness of a peer based on its past reputation, and thus in essence are reputation-based. In the Distributed Trust model, the experience of other peers is used to determine trust. The NICE trust model uses cookies to store trust information and aims at protecting the system from groups of malicious peers by establishing robust cooperative groups. The REGRET model utilizes existing social relationships in the determination of trust. The complaint-based model is different from the other models in the sense that it uses only negative reputation or complaints to determine trust.

4.3.1 Distributed Trust Model

In the Distributed Trust model [1], a peer relies on two types of trust data for determining trust - self experience and recommendations provided by other peers in the system. Two different types of trust relationships are distinguished. When one peer trusts another, it constitutes a *direct* trust relationship. But if a peer trusts another peer to give recommendations about a third peer's trustworthiness, then there is a *recommender* trust relationship between the two. Corresponding to the two types of trust relationships, two types of data structures are maintained by peers - one for direct trust experiences and another for recommender trust experiences. Discrete integral trust values are used to represent the trustworthiness of peers with -1 representing distrust, 0 representing lack of knowledge, 1-3 representing increasing trust and 4 representing complete trust.

Three types of messages are exchanged between peers in this trust model: *Request for Recommendation*, *Recommendation*, and *Refresh* messages. When a peer needs a service offered by another peer for the first time (no prior transactions between the two), the peer sends out a *Request for Recommendation* message to the peers it trusts as recommenders. These recommender peers can respond by sending *Recommendations* if they know the target peer, else they forward the request to other peers whom they trust as recommenders. The model does not specify clearly how long this process of forwarding requests for recommendations continues and when it stops. For our purpose, we assumed that this process continues until all known recommender peers are queried. Since the opinion of peers may change over time, recommendations are valid only for a limited time. When recommendations expire or the trust values associated with them change, they are updated using *Refresh* messages. *Refresh* messages are also used to revoke *Recommendations* by sending *Refresh* messages with trust values 0.

4.3.2 NICE Model

The main goal of the NICE trust model [17] is to identify good peers in the system and establish solid cooperation with them in order to guard against malicious peers. At the end of an interaction, each interacting peer creates a cookie with feedback about the other peer and signs it. These signed cookies are exchanged by the interacting peers. A cookie can either be positive if the transaction is successful or negative otherwise. In other trust models it is the responsibility of the provider peer to ensure that the requestor can be trusted. However, in NICE, when a peer wants to request a certain data or service from a provider peer that it has interacted with before, it presents the provider with a cookie signed by the provider itself. The provider peer verifies it own signature in the cookie and accepts the cookie as evidence of the requestor peer's trustworthiness.

If, on the other hand, the requesting peer does not have a direct cookie, it searches for a path of trust between itself and the provider peer, and presents this path to the provider. Figure 3 shows a directed graph with trust paths between peers A and B, which can be presented by B to A in support of B's trustworthiness.

While positive cookies are exchanged by interacting peers, negative cookies are retained by the peer that creates it. This is in order to ensure that negative cookies are untampered and available to other peers in the system who can search for negative cookies

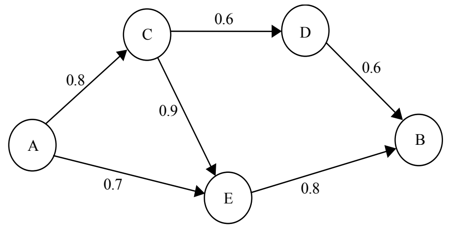

$$\begin{tikzcd}
 & C \arrow[r, "0.6"] \arrow[dr, "0.9"] & D \arrow[dr, "0.6"] & \\
A \arrow[ur, "0.8"] \arrow[rr, "0.7"'] & & E \arrow[r, "0.8"'] & B
\end{tikzcd}$$

Figure 3. Directed Graph in NICE

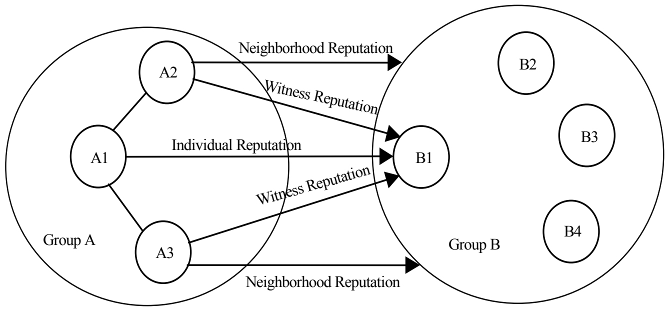


Figure 4. Individual and Social Reputation in REGRET

about a peer before interacting with it. To prevent against attacks perpetrated by colluding peers, peers create robust cooperative groups with other good peers. For this, every peer maintains a preference list of good peers based on their past interaction history.

4.3.3 REGRET Model

The REGRET model [20] assumes that peers are organized into groups and utilizes group relationships to determine trust. The evaluation of reputations is based on impressions which represent the results of interactions between peers. In REGRET there are three dimensions of reputation: *individual*, *social*, and *ontological*.

The *individual* dimension concerns the direct interaction between two peers. The social dimension takes into account group relationships. Consider two groups of peers, Group A and Group B, as illustrated in Figure 4. Suppose peer A1 wants to determine the reputation of peer B1. The *individual* dimension refers to the information A1 has about B1 as a result of their past interactions. The *social* dimension includes both how much A1 trusts all the peers in group B, and how much all the peers in group A trust B1 (*witness* reputation) and how much all the peers in group A trust all the other peers in group B (*neighborhood* reputation). REGRET, however, assumes that there exists complete trust between peers in the same group.

Reputation also depends significantly upon the context. Reputation in a particular context may also imply a certain reputation in other related contexts. Each reputation value may thus represent reputation values for other related contexts. The *ontological* dimension defines these various types of reputation and how they can be combined together to create new types of reputation.

4.3.4 Complaint-based Model

A complaint-based reputation model relies on negative feedback or "complaints" to convey reputation information. In such a model, peers do not store information about successful interactions or trustworthy peers, but rather record their negative experience in the form of complaints against interacting peers. These complaints are also forwarded to another peer or peers in the system in order to disseminate information about the malicious peer. When a peer wants to evaluate the trustworthiness of a target peer, it first searches its own history to locate any previous complaints registered by itself. It can also query other peers for other existing complaints about the target peer. Complaints received from other peers are included in the determination of the target peer's trustworthiness. This kind of complaint-based scheme has been adopted by trust management systems such as the one based on the P-Grid data structure [2].

4.4 CRASH Architectures

Our goal was to develop four different prototypes of the CRASH application, each incorporating one of the trust models. Thus four CRASH architectures corresponding to the four trust models were designed. For each trust model, the CRASH architecture was modified in accordance with the trust model's specifications. Since all four implementations used the same CRASH application, the differences between the four architectures primarily concerned the Trust Manager and the trust-specific components of the *Application* layer. The designs did not affect the *Communication* and *Information* layers since PACE allows us to reuse existing components of these layers independently of the trust model.

For each CRASH architectures, the required components in the different layers were first identified. Following this, the interactions between the components were determined in order to identify the relevant request and notification messages for the architecture. This included modeling of trust-based messages exchanged between peers so that components could appropriately react to them. The design of each trust model is discussed below. Figures 5 and 6 illustrate two CRASH architectures with the Distributed Trust and the REGRET model respectively. Non-shaded components indicate PACE components being reused. Shaded components represent new components being added and components that are cross patterned represent modified components.

4.4.1 Distributed Trust Model Design

In the Distributed Trust model, *Recommendation* and *Refresh* messages contain important recommender trust experiences that needs to be persistently available to the peer. Therefore they are stored in the Internal Information along with the direct trust experiences of the peer. The Trust Manager component encapsulates the algorithm that computes the trust value. It also filters away messages that are marked by the Signature Manager as unsigned or unverified. Two new components, Recommendation Manager and Trust Evaluator are added to the *Application* layer (see Figure 5). The Recommendation Manager dispatches messages to other peers and displays received messages. It also stores, updates, and deletes recommendations from the Internal Information. The Trust Evaluator facilitates the computation of trust by gathering necessary data required to compute trust, sending it the Trust Manager, and displaying computed trust values returned by the Trust Manager to the user. The user can then choose to store these values through the Recommendation Manager.

4.4.2 NICE Model Design

In NICE, a peer needs to store positive cookies about itself signed by other peers, and negative cookies it signed about malicious peers. Both positive and negative cookies need to be persistently

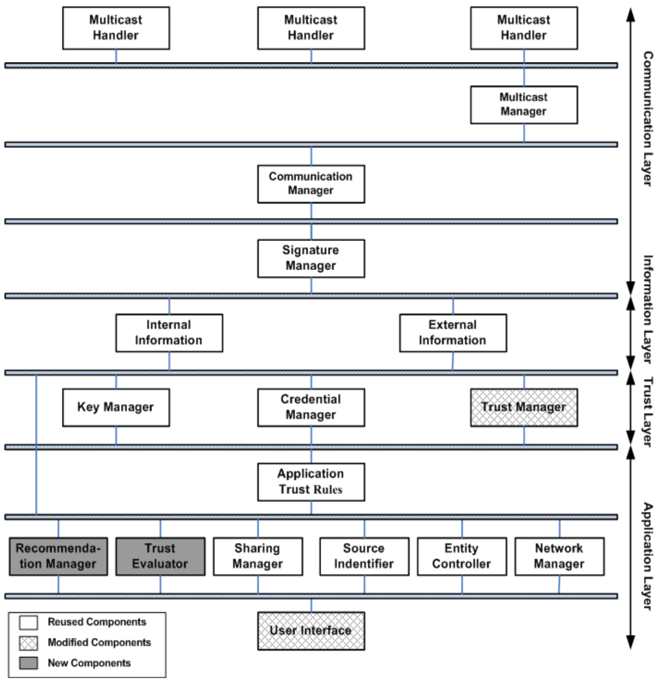


Figure 5. CRASH Architecture with Distributed Trust model

available and so are stored in the Internal Information. The trust algorithm responsible for the determination of trust based on the available cookies is encapsulated in the Trust Manager component. To provide for the functionality of the trust model, two new components are added to the *Application* layer: Cookies Manager and Trust Evaluator. The Cookies Manager sends signed positive cookies to the interacting peer upon interaction, and displays received cookies and paths to a target peer to the user. It is responsible for querying for cookies that will enable it to discover a path of trust to the target peer, responding to such cookie requests, and forwarding cookie requests to peers along the path of trust. It is also responsible for searching and locating negative cookies about a peer. It also stores, updates, and deletes cookies from the Internal Information. The Trust Evaluator facilitates the determination of trust by gathering relevant cookies (and paths of trust) from the Internal Information, sending them to the Trust Manager, and displaying values returned by the Trust Manager to the user.

4.4.3 REGRET Model Design

REGRET uses impressions to represent reputation information. Impressions are the evaluations of the transactions and are used to determine trust. Since impressions need to be persistently available to the peer, they are stored in the Internal Information. The Trust Manager encapsulates the trust computation algorithm that helps determine the trustworthiness of peers based on available impressions. Two new components, Impressions Manager and Trust Evaluator, are added to the *Application* layer (see Figure 6). The Impressions Manager is responsible for storing, updating, and deleting impressions from the Internal Information. Since trust information is stored in the form of impressions in the REGRET model, the Trust Evaluator sends out queries for impressions, responds to queries about impressions, and displays received impressions to the user. It is also responsible for collecting information pertaining to the individual and social dimensions from the

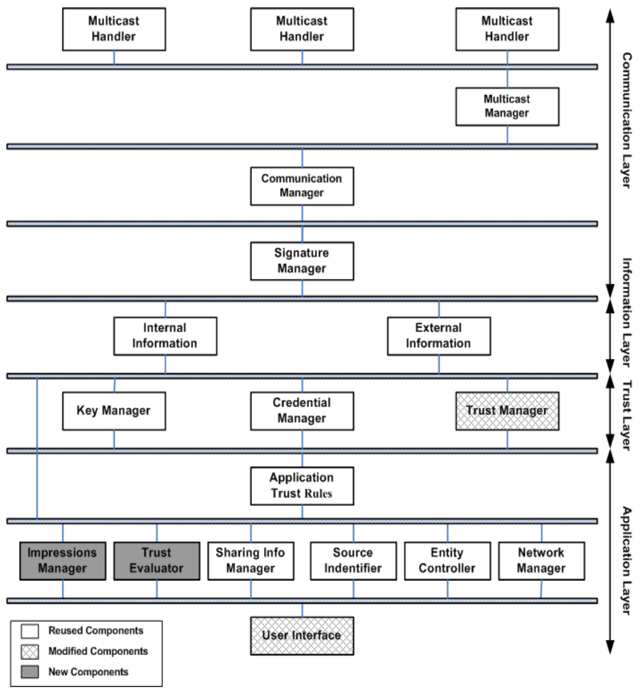


Figure 6. CRASH Architecture with REGRET model

Internal Information, requesting the computation of trust from the Trust Manager, and displaying trust values received from the Trust Manager to the user. The ontological dimension is not addressed in this architecture; however, we envision that it can be encapsulated within the Application Trust Rules component.

4.4.4 Complaint-based Model Design

In the complaint-based trust model, reputation information exchanged between peers take the form of complaints. Complaints received by a peer are stored in the Internal Information in order to make it persistently available. Similar to the designs of other trust models, the Trust Manager incorporates the algorithm that computes trust values.Two new components, Complaints Manager and Trust Evaluator, are added to the *Application* layer. The Complaints Manager creates and files complaints with other peers, queries and responds to queries for complaints, and displays complaints received filed by other peers. It also stores and manages complaints in the Internal Information. The Trust Evaluator is responsible for collecting relevant complaints, sending them to the Trust Manager for use in computing trust, and displaying values returned by the Trust Manager to the user.

5. EVALUATION

The four CRASH prototypes with their associated trust models were evaluated in the context of several decentralization threat scenarios. In this section, we discuss these threat scenarios and our observations of each trust model based on their response to the threat scenarios.

5.1 Demonstration Threat Scenarios

The four CRASH prototypes with corresponding trust models were implemented according to the design described in Section 4.4. Each implementation instance was executed and analyzed independently of others. Each instance was subjected to demonstration scenarios corresponding to the threats of decentralization summarized in Section 4.1. These scenarios are similar to the attack scenario mentioned in [25]. The objective behind these threat scenarios was two-fold. The first was to help examine whether the capabilities of the trust models were adversely affected upon integration within the PACE architectural style. The second was to enable the evaluation and comparison of these trust models in the face of decentralization threats.

A total of 100 threat scenarios were designed with an average of about 5 scenarios per threat per model. Each of these scenarios was then executed on the corresponding CRASH prototypes, and the effects observed and compared with expected results. Since each trust model has varying capabilities and behaves differently, threat scenarios for each model though similar in objective were not identical. Table 1 describes one sample scenario each for *impersonation*, *fraudulent actions, misrepresentation* and *collusion* for the Distributed Trust model. For each scenario, the table lists the corresponding model-based results that were expected and the actual observed results when the CRASH prototype of that model was subjected to that scenario. Organizations mentioned in the table such as the Fire Department and the Police Department actually represent the Command and Control sub-systems belonging to those organizations.

It should be emphasized that the scenarios in Table 1 are only a subset and do not constitute the entire set of scenarios that were actually executed on the Distributed Trust Model CRASH prototype. Similar threat scenarios were also identified for the other three trust models and executed against their CRASH prototypes. However, a detailed discussion of all scenarios and the corresponding trust model behavior in those scenarios cannot be included here due to space constraints. A summary of the observed results from our experiments is presented in the following section.

5.2 Observations

In this section, we briefly discuss some of the distinguishing characteristics of each trust model that were observed when the threat scenarios were executed on the CRASH prototypes. We believe our observations provide some interesting insights about the way these trust models react against the threats of decentralization. We also point out the pros and cons of each model over other models.

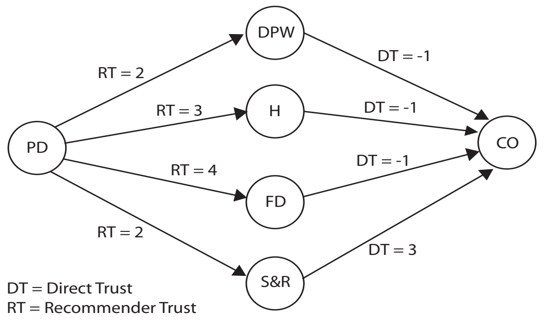
5.2.1 Distributed Trust Model

This is a simplistic trust model that uses recommendations to incorporate the opinion of others. This model primarily focuses on determining trust values for an unknown peer using direct and recommendation trust values. However, it fails to describe how trust relationships can be actually used to determine the relative trustworthiness of received application-based information in order to arrive at a decision. Thus, unless an assumption is made about how these values can be combined (for example, compute an average), the use of this model as is in an actual application is uncertain. Also, since a peer does not query others about a recommender if it already has a previous recommender trust value for the recommender, the peer becomes susceptible to attacks by a trusted recommender.

Another shortcoming of this model is its use of a discrete value system for expressing trust. However, actual trust values computed are typically continuous and it is not clear what the value repre-

Table 1. Selected CRASH Threat Scenarios for the Distributed Trust Model

Threats	Scenario	Result
Organizations and Acronyms: **PD**: Police Department, **FD**: Fire Department, **H**: St. Elsewhere Hospital, **RC**: Red Cross, **CO**: Charitable Organization, **DPW**: Department of Public Works, **S&R**: Search & Rescue, **M**: Media (with a disgruntled employee)		
***Model-specified results*:** Theoretically expected results based upon the Distributed Trust Model ***Observed results*:** Actual behavior when CRASH prototype with the Distributed Trust Model is executed		
Impersonation	1. FD receives a message about a fire from someone claiming to be from the PD.	***Model-specified***: FD detects message is not authentic and blocks it. ***Observed***: FD's signature manager cannot authenticate message against PD's public key; Message is tagged instead of blocked since PACE allows untrusted events to be seen.
Fraudulent Actions	1. PD has investigations into the activities of M. 2. PD decreases trust in M to -1. 3. M reports false information about a fire to FD. 4. FD queries other agencies about M. 5. FD trusts PD to value 3. 6. PD responds truthfully to queries.	***Model-specified***: FD's query reaches PD, who responds with a negative recommendation for M. FD has considerable *recommender* trust (3 on a scale of 4) in PD. So, it should believe the PD and not trust M. ***Observed***: Based on PD's response and FD's *recommender* trust for PD, FD computes the trust value for M and finds it to be -0.75. FD decides not to believe the information provided by M.
Misrepresentation	1. RC has only interacted with H. 2. H has a good impression about the RC. 3. However, M has influence over the staff at H. 4. FD queries other agencies about RC. 5. H sends a value of -1 for RC. 6. FD trusts H to value 3.	***Model-specified***: In this model, FD does not query other peers about H as a recommender, but relies solely on H's *direct* trust. So, it wrongly mistrusts the Red Cross. ***Observed***: As expected, FD incorrectly decides not to believe in RC.
Collusion	1. CO offers assistance to PD. 2. PD queries other agencies about CO. 3. M does not want CO to play any role. 4. M has influence over DPW, H, and FD. 5. PD trusts DPW to value 2, H to value 3, FD to value 4, and S&R to value 2.	***Model-specified***: In response to PD's query only S&R responds positively (3) whereas DPW, H, and FD each respond negatively (-1) about the CO. PD ends up mistrusting CO. ***Observed***: PD uses *direct* trust responses and its *recommender* trust values for the DPW, H, FD, and S&R to compute a value of -0.1875 for the CO (see Figure 7). PD incorrectly decides to mistrust CO.



PD's Trust in CO = [(2/4) * (-1) + (3/4) * (-1) + (4/4) * (-1) + (2/4) * (3)] / 4
= **- 0.1875**

Figure 7. Graph showing Trust Relationships in DTM

sents if it lies between two discrete values. Thus the actual significance of a particular value is lost. Further, the asymmetrical nature of the discrete values chosen has a significant impact in determining trust. Distrust is represented by -1 and total trust by 4. If two peers that are trusted completely (value 4) as recommenders were to give conflicting reports about another peer (one reports distrust and the other reports complete trust), the computed trust value would be 1.5 and wrongly signify that the concerned peer can be trusted in the range of minimal to average trust. However, this model has some useful benefits too. It includes the concept of "context" while determining trust. Context refers to the topic in which information is being provided. This is important because the same peer can be trusted to different extents depending upon the context. It uses message authentication to prevent impersonation. It also provides an explicit message to revoke recommendations. This helps actively propagate information about a malicious peer to other peers in the system who may be unaware of the malicious peer. Such a mechanism is lacking in the other models.

5.2.2 NICE

NICE is different from the other three models in that it puts the onus of proving a peer's trustworthiness on that peer itself. This model is a better fit for applications that allow access to resources or information to only trusted peers, than for applications where a requestor peer must decide the trustworthiness of the provider peer before using its resource or information such as CRASH.

Like REGRET, NICE also has a concept of group, but it is rather simplistic. Peers maintain a list of "friend" peers who are considered completely trustworthy and whose opinion is relied upon most while making a decision. This list that is slowly built over time is an effective mechanism to guard against collusion because even if the majority of the peers are to subscribe to a single opinion, it is the opinion of the friend peers that matters the most. Similarly, misrepresentation attacks can be nullified by relying on the opinion of friend peers. Our observations have revealed that NICE addresses misrepresentation and collusion threats better than the other three models. However, these measures would fail if the majority of friend peers were to collude against the peer. NICE also allows peers to search for negative cookies for other peers.

This helps a peer to become aware of fraudulent and malicious peers. One shortcoming of the NICE trust model is that the context of the provided information does not affect the extent to which a peer is trusted. In other words, if a peer is trusted, all information reported by the peer is trusted to the same extent.

5.2.3 REGRET
The main shortcoming of the REGRET model is that peers can easily become targets of impersonation attacks. This is because unlike the above two models, it does not have any mechanism to uniquely identify peers or to authenticate messages. However, integration with PACE helps address this shortcoming because PACE explicitly provides digital signature-based authentication in the Communication layer.

The unique feature in REGRET is the use of groups and group relationships that provides more trust-related data to determine trustworthiness of peers. This can prove to be very useful in making critical trust-related decisions. For example, consider the following scenario: the Police Department (PD) trusts the Fire Department (FD) and the Media equally. PD requests certain information and FD and the Media give conflicting responses. PD thus cannot determine whom to believe. If however, FD were to belong to a group called Disaster Management Group and the Media were to belong to a group called News Group, and PD trusts the Disaster Management Group more than the News group, then it becomes easy for PD to choose to believe FD over the Media.

However, in REGRET, a peer implicitly trusts all the peers belonging to its own group and in a scenario where peers within the same group lie to each other, the model will fail to make correct trust-related decisions. One significant advantage that REGRET enjoys over the three models is in the way it handles trust contexts. REGRET provides an ontological structure to express the different contexts of trust.

5.2.4 Complaint-based
This model relies only on negative reputations in the form of complaints that are created when an interaction is unsuccessful. Successful interactions are not expressed or stored in the form of positive reputations. Hence, essentially this model cannot help you decide whom to trust more when there are no complaints available about a certain peer. Similarly, this model also fails when a malicious peer creates a false complaint about a good peer. Since peers search only for complaints and no record of "goodness" is stored, the good peer will be mistaken to be a bad peer based on the false complaints found. Another shortcoming of this model compared to other models is that it does not provide any mechanism to protect against impersonation. However, like REGRET, integration of this model with PACE helps address this shortcoming because PACE provides digital signature-based authentication in the Communication layer. Further, unlike the REGRET and Distributed Trust model, this model does not include the concept of context while determining trust.

The above observations of the behavior of trust models in response to threat scenarios can be used to identify some essential characteristics of a suitable trust model for the CRASH application. The most important requirement for an appropriate trust model for CRASH is to take specific measures to counter impersonation. Using unique digital identities and key-based message authentication such as in the Distributed Trust model and NICE are excellent measures to protect against impersonation attacks. A trust model suitable for CRASH can also use the concept of a friend list from NICE. This would mean agencies would rely to a greater extent upon the information reported by other trusted agencies than that reported by the public. This can be extended in a REGRET-like manner to form explicit groups with distinct trust relationships between them. In a crisis situation, it is very important to ensure that wrong information or rumors are not being circulated. For this, the mechanism of actively informing other crisis response peers of malicious peers can be borrowed from the Distributed Trust model.

6. DISCUSSION
Our evaluation of PACE had three principal objectives. The first was to determine the ease with which different trust models could be assimilated in PACE. The second was to assess whether upon integration with PACE, the capabilities of the trust models were preserved and not affected adversely. The third objective was to examine and compare these trust models using PACE as an evaluation platform. Below we describe the observations from our experiments in the context of these three objectives.

Towards our first objective, we successfully integrated the four selected trust models into the four CRASH prototypes using the guidelines provided by the PACE architectural style. We observed that since PACE is layered and enforces visibility constraints upon layers, it is easy to reuse PACE components across different instances. In particular, for the CRASH application, the *Communication*, the *Information* and most parts of the *Trust* layer can be easily reused across prototypes. The only differences in the prototype architectures involve the Trust Manager component and components belonging to the *Application* layer. In other words, an application developer could plug in different trust models into the PACE framework with relatively little effort and could still be confident that the resulting architecture would be able to effectively realize the desired trust management solution. Thus PACE not only supports different trust models but also enables their easy assimilation into a decentralized application. Additionally, we believe that the reusability of the PACE components could be further demonstrated by applying it to an altogether different domain. Specifically, we believe that except for the *Application* layer, components belonging to the other layers could be reused across application domains.

Towards our other objectives, each CRASH prototype was subjected to threat scenarios to see how the application and the trust model reacted to the threats. We found that the observed behavior of the four prototypes in the face of threats matched the results we had anticipated based on our study of the models. This led us to conclude that the capabilities of the trust models are not debilitated by their integration with PACE, and that the trust mechanisms function fully as originally specified. In fact, PACE aids the trust models further by enhancing some of their weaknesses. In particular, PACE provides message authentication, regardless of whether the trust model being used provides it or not, by using digital identities and signatures, and including a signature manager in the communication layer. This is especially useful when working with the Complaint-based and REGRET trust models because they lack authentication mechanisms that help detect impersonation attacks.

Results observed from executing threat scenarios on the CRASH prototypes also helped us draw conclusions regarding the behavior of the associated trust models. Additionally, since PACE and the CRASH application can serve as a common evaluation platform for trust models, these results helped us compare the behavior of the trust models in response to different kinds of threat situations. We believe this kind of a comparison is especially useful since it provides a way to determine and choose a suitable trust model for a given application setting.

Our successful integration of the trust models using the PACE style and the subsequent determination that the capabilities of the models remain unaffected upon integration demonstrate the effectiveness of the guidelines set by PACE towards achieving trust management in decentralized applications. It also revealed how PACE is trust model-independent, can support different trust models, as well as facilitate the evaluation and comparison of trust models towards the selection of a suitable model.

7. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant Numbers 0205724, 0438996, and 0524033.

8. REFERENCES

[1] Abdul-Rahman, A. and Hailes, S. A Distributed Trust Model. In *Proceedings of the New Security Paradigms Workshop.* Langdale, Cumbria UK, 1997.

[2] Aberer, K. and Despotovic, Z. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the Conference on Information and Knowledge Management.* Atlanta, Georgia, Nov 5-10, 2001.

[3] Blaze, M., Feigenbaum, J., et al. Decentralized Trust Management. In *Proceedings of the IEEE Symposium on Security and Privacy.* p. 164-173, May, 1996.

[4] Blaze, M., Feigenbaum, J., et al. *RFC 2704 - The KeyNote trust-management system version 2.* <http://www.faqs.org/rfcs/rfc2704.html>, 1999.

[5] Cahill, V., Gray, E., et al. Using Trust for Secure Collaboration in Uncertain Environments. *IEEE Pervasive Computing Mobile and Ubiquitous Computing.* 2(3), p. 52-61, Aug, 2003.

[6] Capra, L. Engineering Human Trust in Mobile System Collaborations. In *Proceedings of the 12th International Symposium on the Foundations of Software Engineering (SIGSOFT 2004/FSE-12).* Newport Beach, California, USA, Nov, 2004.

[7] Chu, Y., Feigenbaum, J., et al. REFEREE: Trust management for web applications. *World Wide Web Journal.* p. 127-139, 1997.

[8] Damiani, E., di Vimercati, S.D.C., et al. A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security.* Washington DC, Nov, 2002.

[9] Dragovic, B., Kotsovinos, E., et al. XenoTrust: Event-based distributed trust management. In *Proceedings of the 2nd International Workshop on Trust and Privacy in Digital Business.* Prague, Czech Republic, Sep, 2003.

[10] Fenkam, P., Gall, H., et al. An Architectural Style for Development of Secure Software. *Lecture Notes in Computer Science.* 2437, p. 180-198, Jan, 2002.

[11] Grandison, T. and Sloman, M. A Survey Of Trust in Internet Applications. *IEEE Communications Surveys.* 3(4), Dec, 2000.

[12] Grandison, T. and Sloman, M. Trust Management Tools for Internet Applications. In *Proceedings of the 1st International Conference on Trust Management.* Crete, Greece, May, 2003.

[13] Gray, E., O'Connell, P., et al. *Towards a Framework for Assessing Trust-Based Admission Control in Collaborative Ad Hoc Applications.* Distributed Systems Group, Department of Computer Science, Trinity College, Report TCD-CS-2002-66, 2002.

[14] Kamvar, S., Schlosser, M., et al. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the WWW.* Budapest, Hungary, May 20-24, 2003.

[15] Kan, G. Gnutella. In *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, Oram, A. ed. p. 94-122, O'Reilly, 2001.

[16] Kazaa. *kazaa.com*. <http://www.kazaa.com>.

[17] Lee, S., Sherwood, R., et al. Cooperative peer groups in NICE. In *Proceedings of the IEEE Infocom.* San Francisco, USA, Apr 1-3, 2003.

[18] Pujol, J., Sanguesa, R., et al. Extracting reputation in multi agent systems by means of social network topology. In *Proceedings of the 1st Joint Conference on Autonomous Agents and Multi-Agent Systems.* Bologna, Italy, Jul 15-19, 2002.

[19] Resnick, P., Zeckhauser, R., et al. Reputation Systems. *Communications of the ACM.* 43(12), p. 45-48, December, 2000.

[20] Sabater, J. and Sierra, C. REGRET: A Reputation Model for Gregarious Societies. In *Proceedings of the 4th Workshop on Deception, Fraud and Trust in Agent Societies.* Montreal, Canada, 2001.

[21] Suryanarayana, G., Erenkrantz, J.R., et al. PACE: An Architectural Style for Trust Management in Decentralized Applications. In *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture.* p. 221-230, Oslo, Norway, Jun, 2004.

[22] Suryanarayana, G. and Taylor, R.N. *A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications.* UCI Institute for Software Research, Technical Report UCI-ISR-04-6, Jul, 2004.

[23] Taylor, R.N., Medvidovic, N., et al. A Component- and Message-Based Architectural Style for GUI Software. *IEEE Transactions on Software Engineering.* 22(6), p. 390-406, Jun, 1996.

[24] Tierney, K. Research Overview: Emergency Response. In *Proceedings of the The NEHRP Conference and Workshop on Research on the Northridge, California Earthquake of January 17, 1994.* Richmond, California, 1998.

[25] Vigna, G. and Kemmerer, R.A. NetSTAT: A Network-based Intrusion Detection Approach. In *Proceedings of the 14th Annual Computer Security Application Conference.* Scottsdale, AZ, Dec, 1998.

[26] Yu, T., Winslett, M., et al. Interoperable strategies in automated trust negotiation. In *Proceedings of the 8th ACM Conference on Computer and Communications Security.* Philadelphia, USA, Nov 5-8, 2001.

[27] Zetter, K. Kazaa delivers more than tunes. Jan 9, 2004. <http://www.wired.com/news/business/0,1367,61852,00.html>.