PACE: An Architectural Style for Trust Management in Decentralized Applications

Girish Suryanarayana, Justin R. Erenkrantz, Scott A. Hendrickson, Richard N. Taylor

Institute for Software Research
University of California, Irvine
{sgirish,jerenkra,shendric,taylor}@ics.uci.edu

Abstract

Distributed applications that lack a central, trustworthy authority for control and validation are properly termed decentralized. Multiple, independent agencies, or "partners", cooperate to achieve their separate goals. Issues of trust are paramount for designers of such partners. While the research literature has produced a variety of trust technology building blocks, few have attempted to articulate how these various technologies can regularly be composed to meet trust goals. This paper presents a particular, event-based, architectural style, PACE, that shows where and how to incorporate various types of trust-related technologies within a partner, positions the technologies with respect to the rest of the application, allows variation in the underlying network model, and works in a dynamic setting. Initial experiments with variants of two sample decentralized applications developed in the PACE style reveal the virtues of dealing with all aspects of application structure and trust in a comprehensive fashion.

1. Introduction

Decentralized applications are characterized by, among other things, lack of a centralized controlling authority. "Partners" in such an architecture must coordinate by making local autonomous decisions based on potentially incomplete or inaccurate information collected from other partners. A large class of decentralized architectures are open, meaning that the set of partners belonging to the application may change over time, and may consist of both legitimate and illegitimate partners.

In an open, decentralized architecture, illegitimate partners, or peers, may publish intentionally false or misleading information while legitimate peers lack a central authority that can, on their behalf, differentiate the legitimate information from the illegitimate. Consequently, the responsibility of determining which information can be trusted falls squarely on each participating agency, making trust management an essential issue of open, decentralized architectures.

The existing literature has not directly addressed the question of how to design peers for participation in such applications. A large amount of research focused on decentralized peer-to-peer applications has centered on designing appropriate network architectures, particularly when the application involves mobile elements. Research on trust has emphasized the development and exploration of decentralized trust models and algorithms, but has not articulated clearly how they may be utilized in application design and development. Our focus is on meeting this need, describing how trust management may be incorporated in decentralized applications, with an approach grounded in event-based software architectures.

In particular, this paper presents PACE, a trust-centric architectural style that addresses the concerns of trust management in open, decentralized applications. PACE provides explicit guidance on the incorporation of trust mechanisms, while providing the freedom to experiment with different trust models and underlying network architectures.

Though a number of architectural styles exist in the architectural community [3, 22, 28], none of them address the issue of trust in decentralized environments explicitly. However, some of these styles lend themselves more naturally to the constraints imposed by trust and so can be leveraged in our approach. In particular, we believe that the event-based architectural styles which allow loosely-coupled components to asynchronously interact with each other best suit our purposes. PACE is built on such a foundation.

The rest of the paper is organized as follows. The next two sections elaborate on the concepts of decentralization and trust, while Section 4 discusses related work. Sections 5 and 6 present the guiding principles and details the PACE style, with Section 7 detailing the style's induced properties. Section 8 presents our evaluation of two decentralized applications followed by a discussion in Section 9.

2. Decentralized Architectures

A decentralized architecture is a collection of entities, called peers, that interact without the presence of a trusted central controlling authority. Each peer works towards

achieving its own individual local goals, that may or may not serve a common system goal. Furthermore, in an open, decentralized architecture there is no authority preventing the addition of peers with malicious goals. Therefore, each decentralized peer is charged with the task of determining the validity of information received from other peers. This local autonomous determination is the defining principle of open, decentralized architectures. The rest of this paper will refer to open, decentralized architectures as simply decentralized.

There are primarily two layers of abstraction in a decentralized architecture: external and internal. The external architecture facilitates the interaction between peers by describing the topological arrangement of peers and the underlying network infrastructure. On the other hand, the internal architecture is responsible for directing the behavior of a peer towards achieving its local goals. While there has been research towards resolving issues in the external architecture [6], the internal architecture has remained mostly unexplored. Therefore, we believe that the internal architecture warrants detailed investigation.

3. Threats of Decentralization

As discussed above, peers with malicious intent may impose a threat to the goals of others. Peers must take appropriate countermeasures in order to neutralize these threats. A potentially effective countermeasure is to develop trust relationships with others[1].

3.1. Trust Relationships

Many researchers have discussed the requirements for a computational model for trust. One of the underlying principles of trust which has been identified is the concept of perception[10]. As others have pointed out, computational trust models are also highly subjective[14, 18]. Therefore, if we choose to quantify trust, then we must acknowledge that any trust value will be subject to inherent internal flaws due to errors in perception, and external inconsistencies due to incorrect subjective evaluations of others.

Decentralized trust management was first coined by Blaze et.al. who defined it in terms of security policies, security credentials, and trust relationships[4]. A trust relationships between two entities was solely based upon credential verification, and was used to control access to services and resources. A more complete determination of a trust relationship also takes into account the reputation of an entity and provided services as perceived by other entities in the system. Trust management systems based on this definition are also known as reputation-based systems[24, 32].

For our purposes in a decentralized application, we embrace both definitions and will generalize trust to be a measure of the perceived confidence between two peers. More formally, we will use the definition of the trust relationship model introduced in [1]: a trust relationship is always between two entities, is non-symmetrical, and is conditionally transitive.

3.2. Threat Modeling

Instead of considering security as an abstract principle, it is important to model potential threats to that system prior to creation. Thus, we will follow the guideline for evaluating threats as presented in [26]:

1. Understand and assess the real threats to the system
2. Describe the policy required to defend threats
3. Design countermeasures to enforce policy

We now discuss some threats that we believe are introduced by decentralization. In Section 5, we will outline the principles PACE introduces that help defend against these threats. In Section 7, we will discuss the induced properties of the style that act as countermeasures in the PACE style.

3.2.1. Impersonation

Malicious peers may attempt to conceal their identities by portraying themselves as other users. This may happen in order to capitalize on the pre-existing trust relationships of the identities they are impersonating and the targets of the impersonation. Therefore, the targets of the deception need the ability to detect these incidents.

3.2.2. Fraudulent Actions

It is also possible for malicious peers to act in bad faith without actively misrepresenting themselves or their relationships with others. A user can indicate that they have a particular service available even when they knowingly do not have it. Therefore, the system should attempt to minimize the effects of bad faith.

3.2.3. Misrepresentation

Malicious users may also decide to misrepresent their trust relationships with other peers in order to confuse. This deception could either intentionally inflate or deflate the malicious user's trust relationships with other peers. Peers could publish that they do not trust an individual that they know to be trustworthy. Or, they could claim that they trust a user that they know to be dishonest. Both possibilities must be taken into consideration.

3.2.4. Collusion

A group of malicious users may also join together to actively subvert the system. This group may decide to collude in order to inflate their own trust values and deflate trust values for peers that are not in the collective. Therefore, a certain level of resistance needs to be in place to limit the effect of malicious collectives.

3.2.5. Denial of Service

In an open architecture, malicious peers may launch an

attack on individuals or groups of peers. The primary goal of these attacks is to disable the system or make it impossible for normal operation to occur. These attacks may flood peers with well-formed or ill-formed messages. In order to compensate, the system requires the ability to contain the effects of denial of service attacks.

3.2.6. Addition of Unknowns

In an open architecture, the cold start situation arises: upon initialization, a peer does not know anything about anyone else on the system. Without any trust information present, there may not be enough knowledge to form relationships until a sufficient body of experience is established. Therefore, the ability to bootstrap relationships when no prior relationships exist is essential.

3.2.7. Deciding Whom to Trust

In a large scale system, certain domain-specific behaviors may indicate the trustworthiness of a user. Trust relationships should generally improve when good behavior is perceived of a particular peer. Similarly, when dishonest behavior is perceived, trust relationships should be downgraded accordingly.

3.2.8. Out-of-Band Knowledge

Out-of-band knowledge occurs when there is data not communicated through normal channels. When trust is assigned based on visible *in-band* interactions, there may also exist important invisible interactions that have an impact on trust. For example, Alice could indicate *in person* to Bob the degree to which she trusts Carol. Bob may then want to update his system to adjust for Alice's *out-of-band* perception of Carol. Therefore, ensuring the consideration of out-of-band trust information is essential.

4. Related Work

This section gives an overview of relevant research related to our work. We first look at peer architectures. This is followed by an overview of various trust models and algorithms.

4.1. Peer Architectures

4.1.1. INTERRAP Agent Architecture

The INTERRAP agent architecture[20] defines an autonomous agent peer using a layered set of functional components and a shared hierarchical knowledge base. The main benefits of INTERRAP are the explicit modeling of local autonomous behavior, and local and cooperative plans. While these benefits certainly make the agent architecture of INTERRAP feasible for dynamic decentralized multi-agent systems, the main shortcoming is an assumption of implicit trust among agents. The architecture does not consider the effect of information sent by malicious agents which may prevent peers from achieving local goals.

4.1.2. Trust Architecture

Another internal architecture is the local trust-based admission control architecture presented in [14] that helps access control using trust-based admission control policies. This approach has two main shortcomings. The first is that the global admission control process requires a centralized application manager to coordinate the voting process for admitting a new peer. The second is that, though this framework takes an architecture-centric approach, they store only trust and interaction data persistently, and communication among the peers is not explicit.

4.2. Trust Models

The realization that trust is of immense significance in a decentralized context has motivated a lot of research across different areas to be focused on reputation and trust management models and systems [1, 12, 13, 15, 27, 32]. Below, we focus our attention on some of the interesting trust models and algorithms discussed in research literature. However, while it is certainly feasible to integrate these models into an application's internal architecture, none of these approaches have identified mechanisms to do this.

The first decentralized trust management systems like PolicyMaker[4] and Keynote[5] were simple access control systems. Their concept of trust management only involved using credential verification and secure application policies to restrict access to resources and services. These systems and other access control systems like REFEREE[7] and TrustBuilder[31] are limited in the sense that they did not enable an entity to aggregate the perception of other entities in the system in order to choose a suitable reputed service.

Reputation-based systems like XREP[8], NICE[17], and P-Grid[2] on the other hand, provide users additionally with a facility to compute the reputation of a user and resources offered by him by aggregating the perception of other users in the system. This reputation information can be effectively utilized while establishing trust relationships. Some reputation systems like Trustnet[25] and NodeRanking[23], in addition, utilize existing social relationships to determine reputations. These models map the peer-to-peer network to a social network graph and rely on various parameters like the weights on edges, the number of edges entering or leaving out of a node, etc. to determine the social reputation of a peer.

5. Foundations of PACE

Our search for a suitable architectural style began with the recognition that since peers are locally autonomous, they can choose how and when to respond to information they receive. Since synchronous external interaction cannot be expected, an asynchronous internal architecture may be

well-suited. Further, in order to better evaluate effects of different network topologies, data and trust models, the architectural style should facilitate dynamism supported through loose coupling of components[21].

Event-based architectural styles have been successful in addressing the constraints of asynchronicity, dynamism, and loose coupling. C2 is one such architectural style that naturally fits these constraints[30]. Additionally, C2 provides good tool support to facilitate rapid development. Therefore, the PACE architectural style builds upon C2. We now present an overview of the C2 style followed by a description of the guiding principles behind the PACE architectural style.

5.1. C2 Architectural Style

C2 is an asynchronous, event-based architectural style, which promotes reuse, dynamism, and flexibility through limited visibility. Components and connectors have a defined top and bottom that cause them to be arranged in layers. Components are aware of elements that reside above them but not below. Hence, they may send requests, events that travel up an architecture, with an expectation that they will be fulfilled by some set of components above. Components may also send out notifications, messages that travel down an architecture, without any expectation of whether they will be handled.

5.2. PACE Guiding Principles

Our study of decentralized trust management systems helped us identify the following set of principles that have guided the creation of the PACE architectural style.

5.2.1. Digital Identities

Without the ability to associate identity with published information, it is a challenge to develop meaningful trust relationships. The concept of identities, both physical and digital, are necessary to facilitate meaningful relationships. However, it is important to understand the limitations of digital identities in respect to physical identities.

There may not be a one-to-one mapping between digital and physical identities as one person may utilize multiple digital identities or multiple people may share the same digital identity. Additionally, anonymous users may be present who resist digital identification. Therefore, it is not always possible to tie a digital identity to one physical individual and make accurate evaluations of a person. Instead, a critical criteria of trust relationships in decentralized applications should be the actions performed by digital identities not by physical identities. *PACE, therefore, considers trust relationships only between digital identities.*

5.2.2. Separation of Internal and External Data

Reflection on our previous work in decentralized emergency response applications [29] revealed the importance of modeling external data separately from internal data. This separation helps resolve conflicts between externally reported information and internal perceptions. For example, a peer may favor information it has perceived directly and believes to be accurate over reported information. A peer may also not want to disclose sensitive data, so it must have the privilege to report information which differs from what it actually believes. *Explicit separation of internal from external data supports this, and is adopted by PACE.*

5.2.3. Explicit Trust

Without a controlling authority that governs the trust process, peers require information to make decisions whether or not to trust what they perceive. Active collaboration between peers may provide enough knowledge for peers to reach their local decisions. In order to process this trust information internally across the entire architecture, trust cannot be localized to only one component. Each component responsible for making local decisions needs the ability to take advantage of this perceived trust. If the perceived trust is not visible, then accurate assessments may not be able to made. *Therefore, PACE requires trust relationships to be visible to the components in the peer's architecture as well as published externally to other peers.*

5.2.4. Comparable Trust

Ideally, published trust values should be syntactically and semantically comparable - that is, equivalent representations in one implementation should have the same structure and meaning in another. If the same value has different meanings across implementations, then accurate comparisons across peers cannot be made. There has been no clear consensus as to which trust semantics provide the best fit for applications[1, 18], therefore it is believed that enforcing a constraint at the architectural level to use a particular trust semantic would be too imposing. *While trust values should but are not required to be semantically comparable, PACE imposes a constraint that trust values must be syntactically comparable by enforcing that they be represented numerically.*

5.2.5. Dependencies of Layers

As depicted in Figure 1, the PACE architectural style consists of the following layers of functionality: *Communication*, *Information*, *Trust*, and *Application*.

The *Communication* layer is responsible for performing data collection and transmitting data to other external peers. PACE requires that all external communication must be performed through this layer. However, since the activity of data collection does not depend upon data storage or analysis, this layer can reside at the top of a C2-based architecture. Externally received data can then be sent as notifications and data to be sent can be treated as requests.

All notifications emitted from the Communication layer and all data must be stored within the *Information* layer.

Data may be selectively queried, updated, and deleted from this layer. The next layer is the *Trust* layer which is responsible for evaluating the received messages and updating the Information layer with the results of these evaluations. Since the Trust layer depends on internal data for its evaluations, it must be below the Information layer due to the C2 visibility rules.

While the layers described above are generic and may be implemented in an application-independent fashion, the *Application* layer is domain-specific. It is responsible for controlling the local behavior of the peer and can build upon the services provided by the generic layers. Therefore, an application developer is responsible primarily for implementing the Application layer.

Thus, the arrangement of these layers is influenced both by their explicit interaction and C2's visibility rules discussed in Section 5.1. The components and connectors that comprise these layers are discussed in Section 6.

5.2.6. Implicit Trust

PACE assumes implicit trust of components that constitute the internal architecture. The only exception is the Communication layer because it is not responsible for validating the messages from other peers. So any notification sent by the Communication layer cannot be trusted. *Therefore, these notifications require an explicit trust value*.

Since the Communication layer is the only one that can have external communications and is situated at the top of the architecture, it cannot, by definition, issue requests to other layers. Rather, requests can originate only from components below the Communication layer. *Since these components are internal and thereby trusted, requests originating from them are implicitly trusted.*

Consequently, components within the architecture treat requests and notifications differently. For example, the Information layer only allows requests to query, update or delete stored information, and prevents notifications from external peers received through the Communication layer to do the same.

6. PACE Architectural Style

Building upon the foundations presented in the previous section, we now introduce the PACE architectural style with its specific topological and component constraints. Figure 2 illustrates a sample internal architecture of a peer constructed in the PACE style. The components in this diagram, however, are generic, and can be replaced by a system architect with other components that meet these same constraints.

6.1. Communication Layer

The Communication layer handles the interaction of a peer with others. This layer has three main functions:

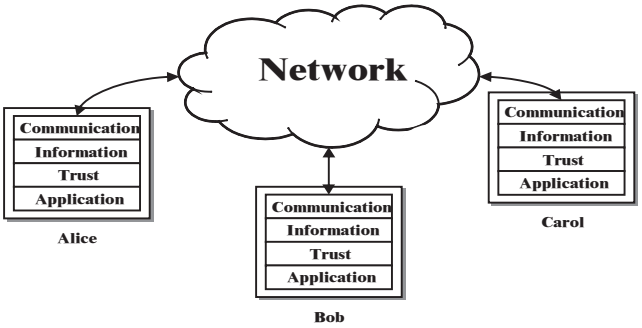

Figure 1. External Architecture

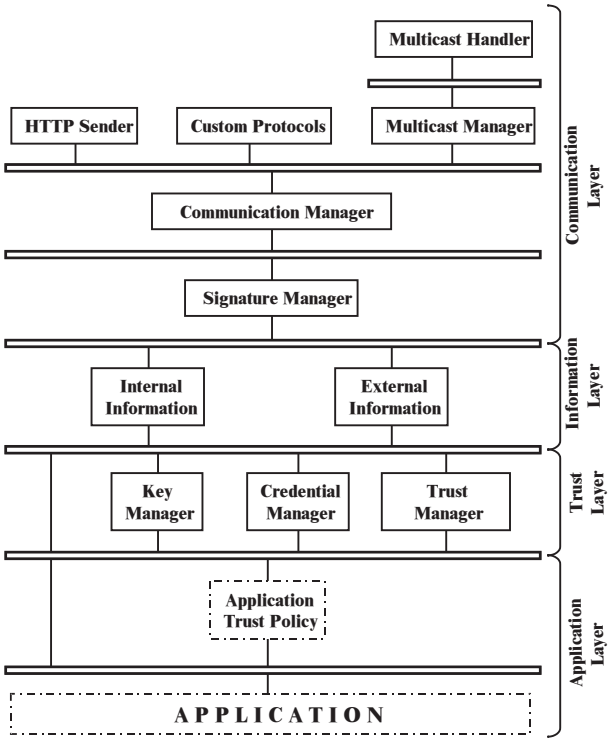

Figure 2. Internal Architecture in PACE

1. abstraction of underlying connection protocols
2. allowing multiple connection mechanisms at once
3. signing of messages and verification of identities

In order to achieve maximum flexibility, the type of the underlying protocols used are isolated to the protocol handler component. Each protocol handler is controlled by the communication manager. Underneath the communication manager is the signature manager which signs and verifies messages as they pass through the architecture.

6.1.1. Protocol Handler

In order to communicate using several network infrastructures simultaneously such as HTTP, SMTP, or FLAPPS[19], PACE supports multiple protocol handlers. These are responsible for translating internal events into

the specific format best suited for the associated external protocol and vice-versa. In order to support multiple protocol handlers, each protocol handler is configured for a specific URL scheme that it will service.

There are two categories of protocols that can be supported: stateful and stateless. Stateful protocols often require a persistent connection or are setup with a specific server or group of nodes. Therefore, a protocol handler for such a stateful protocol can only service one particular instance. However, if the underlying protocol is stateless, then one protocol handler can service all requests for its registered scheme.

6.1.2. Communication Manager

The communication manager is responsible for the dynamic creation of protocol handlers. This creation is determined by having a registry of current protocol handlers and the protocol specified in the address field of a message.

6.1.3. Signature Manager

The signature manager is responsible for signing requests and verifying notifications. Public-key infrastructure standards can be used for digital identiies[11]. As discussed in Section 5.2.1, identities in PACE can only be digital - there is no guarantee that the digital identity which signed a message always corresponds to a physical identity. Reconciling physical and digital identities will require leveraging out-of-band knowledge.

To provide explicit identification, non-repudiation and integrity checking, the Signature Manager digitally signs outbound messages with the locally configured private key. The signature manager then embeds the signatures and corresponding public key within the outbound message to allow transport over protocols which do not support digital signatures. When an event is received by others, the included public key can be used to verify the signature.

6.2. Information Layer

To conform with the separation principle in Section 5.2.2, the Information layer consists of two components: the internal information component that stores requests, and the external information component that stores notifications. Information stored in this layer can be queried and potentially modified by requests.

6.2.1. Internal Information

The data stored in the internal information component is typically intended to be persistent across instantiations of a peer. This allows a peer to accumulate a historical record of its own prior actions and beliefs. This component only allows direct modifications and queries through requests in order to prevent unintentional distribution of data to other peers. Only if a request is tagged with an address field, the message will be stored locally and forwarded to the Communication layer for transmission.

6.2.2. External Information

The external information component maintains only messages received from other peers and is not meant to be persistent. It is imperative to understand that the external data may be incomplete or include intentionally false information published by other peers. As discussed next, the Trust layer assigns trust values on this information.

6.3. Trust Layer

The Trust layer encapsulates the trust management policies of the local peer. In order to achieve this, different credential and reputation-based trust models such as those discussed in Section 4.2 can be incorporated in this layer.

6.3.1. Key Manager

The key manager component generates the unique key pairs that the signature manager uses to sign externally-bound requests. Configured public and private key pairs are stored as internal information. If the key manager does not detect a configured key pair, it generates a new set of keys. This new set is stored in the internal information component and sent to the signature manager for its use.

6.3.2. Trust Manager

The trust manager incorporates different trust models and algorithms such as those discussed in Section 4. Depending upon the type of trust management system employed, trust managers can be of two types: reputation-based, and credential and policy-based. A reputation-based trust manager uses reputation models and algorithms to assign explicit trust values to messages received from other peers. A credential and policy-based trust manager, on the other hand, acts as a trust engine that authorizes information access to a peer based on whether the peer's credentials match the policies defined by the application trust policy component.

6.3.3. Credential Manager

The credential manager component is responsible for maintaining the locally cached identity information stored in the Information layer. It may request public keys from other peers when needed and also respond to key revocation notifications.

6.4. Application Layer

The Application layer consists of components which depend upon the specific needs of the application. The PACE framework does not provide implementation of these components, and expects the application developer to select suitable implementations. It should also be noted that all outgoing communications are initiated in this layer.

6.4.1. Application Trust Policy

The application trust policy depends upon the type of

Table 1. Summary of Threats, Policies, and Key Components in PACE

Threats	Policies	Key Components	Comments
Impersonation	Signatures	Key manager, Signature manager	Without the correct private key, the signature will not validate as coming from the corresponding public key
Fraudulent Actions	Trust Values, Broadcasts	Application layer, Trust manager, Communication layer	In response, malicious users may be assigned a low trust value, which can be broadcast to others to warn
Misrepresenting Trust	Trust Values	Application layer, Trust manager, Communication layer	Users are able to consider the evaluations of others; messages may be published to warn others of malicious activity
Collusion	Signatures, Transitivity	Signature manager, Trust manager	A malicious collective can be defeated using explicit trust communication, digital signatures, and isolation of malicious peers
Denial of Service	Isolation	Communication layer	By isolating protocols to the Communication layer, malicious attacks can be blocked; firewalls can be actively controlled
Addition of Unknowns	Untrusted Events Still Seen	Signature manager	Do not assign trust values to unsigned or incorrectly signed messages; allow users to still view and respond to such events
Deciding Whom To Trust	Domain-Specific Policies	Application trust policy	Each application may have certain behavior indicative of goodness or maliciousness that can be detected by an algorithm
Out-of-Band Knowledge	Overrides	Application layer	Almost infeasible to have trust model capture all relevant inputs, therefore the user may need to adjust manually

trust management system employed and matches the type of trust manager used. In a reputation-based system, the application trust policy is responsible for assigning trust values based on domain-specific semantic meanings of messages, and supporting different dimensions of trust relationships. Dimensions could be topic-based[1], for example, Alice may trust Bob completely when it comes to buying books, but may trust Carol more when it comes to buying cars. Consequently, there can be multiple ways in which a particular trust value may be computed by the trust model, requiring the relationships between these different trust dimensions to be explicitly defined. In a credential and policy-based system, on the other hand, the application trust policy maintains domain-specific policies that are used for authorization.

6.4.2. Application

This component defines the local behavior of each peer that is specific to an application. The application component may be a sub-architecture that can take advantage of the services provided by the other layers. It can include such functionality such as providing an interface to the user and carrying out the local goals of the peer.

7. Induced PACE Benefits

In this section and as detailed in Table 1, we discuss how the guiding principles in Section 5 and the PACE architectural style as described in Section 6 together induce properties that act as countermeasures against the threats outlined in Section 3. Some of these properties are effected by the PACE architectural style and canonical implementations of its standard components; others are application specific and so need to involve the Application layer.

7.1. Impersonation

Since all external communication in the PACE architecture is constrained to the Communication layer, it offers a single point where impersonation can be detected. The deception of a malicious peer that either tries to impersonate a user without the correct private key or does not digitally sign the message, can be easily detected by verifying signatures. The signature manager, key manager, and trust manager components work together to implement signing and verification of messages.

Additionally, if a private key has been compromised, a revocation for that key can be transmitted. The credential manager can store this revocation in the Information layer, and the trust manager can then refuse to assign trust values to revoked public keys even if they have a valid signature.

7.2. Fraudulent Actions

Since PACE is designed for open, decentralized architectures, there is little that can be done to prevent the entry of malicious users. Malicious actions can be detected by the domain-specific Application layer and low trust values can be assigned to those malicious peers. Explicit warnings can then be issued about those malicious peers to others who can then consider these warnings in their local evaluations.

7.3. Misrepresenting Trust

Since PACE facilitates explicit communication of comparable trust values, a peer can incorporate trust relationships of others. For example, if Alice publishes that she distrusts Bob, then Carol can use that information to determine if she should trust Bob's published trust relationships. This can be accomplished by implementing a transitive

trust model in the trust manager which allows peers to disregard trust relationships reported by distrusted peers.

7.4. Collusion

Collusion is of a greater concern than a single peer misrepresenting trust because malicious peers are working in concert to confuse. It has been proven that explicitly signed communication between peers can overcome a malicious collective in a distributed setting[16]. Adapting and combining these results with efficient schemes to identify non-cooperative groups in a decentralized setting, such as in NICE[17], and PACE's ability to detect impersonation allows collusion to be effectively addressed.

7.5. Denial of Service

The separation of the Communication layer allows isolation and response of the effects of denial of service attacks. Incorrectly-formed messages can be disposed of by the protocol handlers. The communication manager can also compensate for well-formed message floods by introducing rate limiting or other policies designed to reduce this threat. For example, it could interact with neighboring firewalls to prevent further entrance of floods.

7.6. Addition of Unknowns

Even though a peer may not have previously interacted with another peer or a message may be known to be forged, the Application layer can still receive these events. Without enough information to make an evaluation, the message will not be assigned a trust value by the trust manager. However, the user can still make the final decision to trust the contents of the message based on out-of-band knowledge that is not captured explicitly by the system.

7.7. Deciding Whom To Trust

The application trust policy component allows for automated identification of specified patterns of application-specific behavior. The detection of good or bad behavior by this component can cause the trust level of the corresponding peer to be increased or decreased respectively along a particular trust dimension.

7.8. Out-of-Band Knowledge

While PACE confines all electronic communication to the Communication layer, it is still possible for relevant trust relationships to be conveyed in person, or through other out-of-band mechanisms. Therefore, the Application layer can issue requests that modify the trust relationship for either a specific message or peer on behalf of the user.

8. Evaluation

Using a framework that supports development in the PACE architectural style, we have successfully implemented prototypes in two problem domains: decentralized auctioning and common operational picture systems. Trial usages of these systems have demonstrated the benefits induced by PACE described in Section 7.

8.1. Modeling a Decentralized Auction in PACE

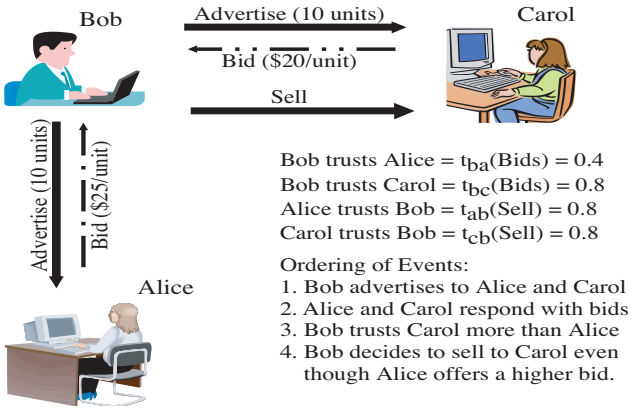

Figure 3. A Decentralized Auction

We have implemented an auctioning system in the PACE architectural style that does not have a trusted central controlling authority. Figure 3 depicts an auction where a seller advertises availability of goods for sale with buyers directly placing bids with the seller. Our system also supports reverse auctions where buyers advertise interest in specific goods and sellers submit quotes to buyers.

Each auction advertisement includes a URL that indicates where bids may be submitted. This URL may refer to the peer itself, or to a trusted third-party that will manage the auction on the seller's behalf. Communication is handled through two asymmetrical protocol handlers: advertisements are distributed through a multicast channel, while bids are placed through a point-to-point channel.

In order to uniquely identify peers, public and private keys are utilized. Internal and external data isolation is enforced by having the external information component stripping received notifications of any trust before further processing to prevent external evaluations of trust from entering the system. Lastly, all peers use the same application trust policy and trust model component to allow uniform semantic comparison of explicit trust values.

The Communication, Information, and Trust layers are implemented generically; as described in Section 5.2.5, all domain knowledge is restricted to the Application layer. Since PACE builds upon the C2 style, PACE applications can reuse technology designed for C2. In particular, we described the auctioning architecture using xADL[9] and built our PACE framework upon the c2.fw framework.

8.2. Demonstrating PACE's Induced Benefits

The auctioning system helps demonstrate that the bene-

fits induced by PACE can indeed address the threats, as discussed in Section 7. All attempts at *impersonation* are exposed since messages with false signatures are not tagged as verified by the signature manager and also result in a system warning being emitted. Peers respond to malicious peers performing *fraudulent actions* by broadcasting appropriate trust values. This allows prospective buyers and sellers to avoid those with low trust values.

Furthermore, by displaying broadcasted trust values through a graphical interface, *misrepresentation of trust* can be easily detected by identifying outliers among those values where a consensus emerged. To allow resistance against *collusion* and misrepresenting peers, a simple transitive trust model is employed, which only takes into consideration explicitly trusted peers' trust values. By isolating communication to one layer, the effects of basic *denial of service* attacks are reduced by disallowing passage of ill-formed messages from outside to other layers in the system.

When the system first initializes, auctions and bids are still reported by untrusted peers. By deferring trust computation initially, this allows the manual formation of trust relationships and *addition of unknowns*. In order to *decide whom to trust*, the application trust policy identifies two distinct trust dimensions: the buying and selling of items. This allows a peer to make trust decisions based on specific prior actions. Lastly, a user can directly modify trust values of either a particular message or a peer to enable *out-of-band knowledge*.

8.3. Common Operational Picture

In this domain, independent governmental organizations share real-time information with other independent countries. This gathered data is then depicted visually in order to allow an operator to assess the incoming data and issue commands accordingly. A screenshot of the running system is depicted in Figure 4.

We have previously created a system for this domain in the C2 architectural style, which raised two issues that we felt could be best resolved by migrating to the PACE architectural style: complexity in adding new operational entities and organizations; and insufficient ability to assess the trust of incoming data. By modeling each operational entity as a peer written in the PACE style, a clear separation of control can be identified. Peers can be added and removed from the system dynamically. The addition of explicit, comparable trust and digital signatures also allows for a more faithful representation of the system which allows further exploration of trust issues.

Additionally, we found that in such a large collection of nodes, not all layers were required on every node. Instead, if a node only transmits, but does not receive information, the full Information and Trust layers are not needed. The exception is the Key Manager component to provide digital identities on transmitted information. In practice, however, we found that most nodes indeed required all four layers as they transmitted and received information with others.

9. Discussion

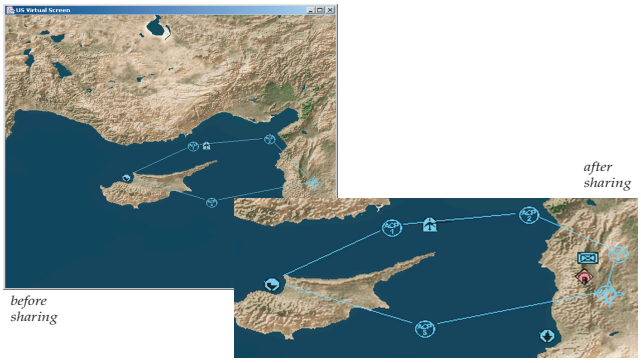

Figure 4. Common Operational Picture

It is important to understand what PACE provides as an architectural style. It does not provide a trust model, but instead facilitates the introduction of a variety of trust models to the architecture. Similarly, PACE does not prescribe a communication method, rather it allows for the introduction of multiple communication models to be shared among the same internal architectural instance.

There is an abundance of technologies that can help address these essential aspects of communication and trust management in decentralized applications. So far, these approaches have remained disjoint with no guide for composing them to construct decentralized applications. Additionally, as described in Section 5, the absence of explicit trust at all architectural levels makes it a challenge to accurately assess information in a component-based implementation.

The PACE architectural style addresses these challenges by presenting an approach for integrating communication, data and trust models independently within an internal architecture to support dynamic modification. Our evaluation has revealed that the PACE architectural style induces a number of beneficial properties that allows several threats of decentralization to be addressed. We have also implemented a decentralized auction system and common operational picture system using a generic reusable framework constrained by the PACE architectural style, which have illustrated the feasibility of the PACE architectural style as a guide to integrating trust into decentralized applications.

10. Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0205724.

11. 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, November 5-10, 2001.

[3] Batory, D. and O'Malley, S. The Design and Implementation of Hierarchical Software Systems with Reusable Components. *ACM Transactions on Software Engineering and Methodology*. 1(4), p. 355-398, October, 1992.

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

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

[6] Carzaniga, A., Rosenblum, D.S., et al. Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service. In *Proceedings of the Nineteenth ACM Symposium on Principles of Distributed Computing*. p. 219-227, ACM Press. Portland, OR, July, 2000.

[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, November, 2002.

[9] Dashofy, E.M., Hoek, A.v.d., et al. An Infrastructure for the Rapid Development of XML-based Architecture Description Languages. In *Proceedings of the 24th International Conference on Software Engineering (ICSE 2002)*. p. 266-276, ACM. Orlando, Florida, May, 2002.

[10] Deutsch, M. Cooperation and Trust: Some Theoretical Notes. In *Nebraska Symposium on Motivation*, Jones, M.R. ed. Nebraska University Press, 1962.

[11] Diffie, W. and Hellman, M.E. New Directions In Cryptography. *IEEE Transactions on Information Theory*. 22(6), p. 644-654, November, 1976.

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

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

[14] 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.

[15] Josang, A. and Ismail, R. The Beta Reputation System. In *Proceedings of the 15th Bled Electronic Commerce Conference*. Bled, Slovenia, June 17-19, 2002.

[16] Lamport, L., Shostak, R., et al. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*. 4(3), p. 382-401, July, 1982.

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

[18] Marsh, S. *Formalising Trust as a Computational Concept*. Thesis. Department of Mathematics and Computer Science, University of Stirling, 1994.

[19] Michel, B.S. and Reiher, P. Peer-to-Peer Internetworking. In *Proceedings of the OPENSIG 2001 Workshop*. Imperial College, London, 24-25 September 2001, 2001.

[20] Muller, J.P. and Pischel, M. An architecture for dynamically interacting agents. *International Journal of Intelligent and Cooperative Information Systems (IJICIS)*. 3(1), p. 25-45, 1994.

[21] Oreizy, P. and Taylor, R.N. On the Role of Software Architectures in Runtime System Reconfiguration. In *Proceedings of the Fourth International Conference on Configurable Distributed Systems*. p. 61-70, IEEE Computer Society Press. 1998.

[22] Perry, D.E. and Wolf, A.L. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*. 17(4), p. 40-52, October, 1992.

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

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

[25] Schillo, M., Funk, P., et al. Using trust for detecting deceitful agents in artificial societies. *Applied Artificial Intelligence Journal, Special Issue on Trust, Deception and Fraud in Agent Societies*. 2000.

[26] Schneier, B. *Secrets and Lies: Digital Security in a Networked World*. 432 pgs., John Wiley & Sons, Inc., 2000.

[27] Shankar, N. and Arbaugh, W. On Trust for Ubiquitous Computing. In *Proceedings of the UBICOMP2002 - Workshop on Security in Ubiquitous Computing*. Goteborg, Sweden, September 29th, 2002.

[28] Shaw, M. and Garlan, D. *Software Architecture: Perpectives on an Emerging Discipline*. 242 pgs., Prentice Hall, 1996.

[29] Suryanarayana, G. and Taylor, R.N. A Decentralized Algorithm for Coordinating Independent Peers: An Initial Examination. In *Proceedings of the Tenth International Conference on Cooperative Information Systems (CoopIS)*. p. 213-229, Irvine, California, October 30 - November 1, 2002.

[30] 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, June, 1996.

[31] 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.

[32] Zacharia, G. and Maes, P. Trust Management Through Reputation Mechanisms. *Applied Artificial Intelligence*. 14, p. 881-907, 2000.