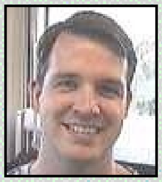
PACE: An Architectural Style for Trust Management in Decentralized Applications

Girish Suryanarayana
Univ. of California, Irvine
sgirish@ics.uci.edu


Scott A. Hendrickson
Univ. of California, Irvine
shendric@ics.uci.edu

Justin Erenkrantz
Univ. of California, Irvine
jerenkra@ics.uci.edu

Richard N. Taylor
Univ. of California, Irvine
taylor@ics.uci.edu

October 2003

ISR Technical Report # UCI-ISR-03-9

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

ISR Technical Report # UCI-ISR-03-9

September 2003

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 a sample decentralized application developed in the PACE style reveal the virtues of dealing with all aspects of application structure and trust in a comprehensive fashion.

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

ISR Technical Report # UCI-ISR-03-9, September 2003

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 a sample decentralized application 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 appications. 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 [4, 12, 23, 25, 26], 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. Section 5 presents the the PACE architectural style. Section 6 gives an overview of the various PACE components. Section 7 presents our initial evaluation of a prototype decentralized application that we built in the PACE style. Finally, we conclude with a discussion of the style in Section 8.

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 [5], 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[9]. 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.

For our purposes in a decentralized application, we will consider trust as 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

Before a system is created, it is important to model potential threats to that system. Thus, we will follow the guideline for evaluating threats as presented in [24]:

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 constraints PACE introduces that should help defend against these threats. In Section 7, we will evaluate the specific countermeasures PACE utilizes.

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

Initially, in an open architecture, when the system is first created, the cold start situation arises: a particular peer does not know anything about any other user on the system. This can occur when the system is first initialized, or when newcomers join the system after it has been established.

Without any information relating to trust, peers may not have enough knowledge to form solid trust evaluations of others. Any trust metric based on prior knowledge will be hampered until a sufficient body of prior knowledge is established. Therefore, peers need the ability to bootstrap when they have no pre-existing trust relationships.

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 relevant data that is not communicated through normal channels. For example, Alice could indicate *in person* to Bob the degree in which she trusts Carol. Bob may want to update his system to adjust for Alice's perception of Carol. While trust values are assigned by an algorithm based on visible interactions, there may exist important invisible interactions that have an impact on trust that might not be taken into consideration. Therefore, the system needs to facilitate users being able to reconcile new information gathered from other sources.

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[21] 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 and Algorithms

The realization that trust is of immense significance in a decentralized context has motivated a lot of research to be focused on reputation and trust management models and systems [7, 11, 20, 29, 30]. 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 the internal architecture, none of these approaches have identified mechanisms to do this.

4.2.1. Efficient Data Models

One approach has been to invent efficient trust data storage and mining techniques. For example, the P-Grid approach[2, 3] stores data in an innovative decentralized fashion and uses randomized algorithms that enable constructing the access structure, updating data, and performing search. This is done to address scalability issues that arise in large agent populations when trust information is stored on every peer.

4.2.2. Trust Models

Another set of approaches involve discovering new ways of modeling and analyzing trust data. One of the earliest models used a "recommendation" protocol to achieve trust[1]. Another computes trust locally using the P-Grid storage structure described above[3]. Poblano [6] is another decentralized trust model implemented over JXTA that allows reputation-guided searching.

4.2.3. Trust Algorithms

Recent efforts have also concentrated on developing algorithms to compute trust values. For example, [15] claims that beta probability density functions can be used to combine feedback and derive reputation ratings in a decentralized application. [16] describes a distributed EigenTrust algorithm that computes global trust values for each peer based on the peer's history of uploads.

5. PACE Architectural Style

We now introduce the PACE architectural style. Since peers are locally autonomous, they can choose how and when to respond to information they receive. Due to the fact that 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[22].

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[28]. 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 the introduction of 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 Architectural Style

We now introduce the PACE architectural style and its constraints upon the architecture that are geared towards addressing the threats discussed in Section 3.

5.2.1. Identities

In a decentralized system, it is often necessary to identify which physical entity published some information. Without the ability to associate identity with information, it is a challenge to develop a meaningful relationship between entities. Therefore, a critical constraint is having enough knowledge to make judgements of trust. This can be facilitated by the creation of digital identities.

It is possible that there is not a one-to-one mapping between digital and physical identities. Multiple digital personas under the control of one physical identity are often common when each digital persona represents a role. Each digital identity may be used to perform possibly distinct tasks by the same person.

Similarly, multiple physical identities could share one digital identity. This can occur when there is a group account shared by several people. There is also a special case where one digital identity is shared: anonymous users. These users do not identify themselves, so it is not possible to know who they really are. Therefore, it is not always possible to tie a digital identity back to one real individual and make accurate evaluations of their behavior.

It becomes clear that an identity does not necessarily represent any particular physical entity, but that it represents the actions performed by that entity in a particular system. Therefore, a critical criteria of developing trust relationships should be the actions performed by digital identities.

5.2.2. Explicit Trust

Without a controlling authority that governs the trust process, peers require information to make decisions whether or not to trust. This information can be about a particular identity, or about information produced by an identity. Active collaboration between peers may provide enough knowledge for peers to reach their local decisions. Consequently, peers should publish their perceived trust explicitly in order to facilitate active collaboration.

However, within an internal architecture, trust cannot just be localized to one component. Each component responsible for making local decisions needs the ability to take advantage of the perceived trust of the information. If perceived trust is not visible, then those components may not be able to make accurate assessments. Therefore, the trust relationships need to be visible to the components in the peer's architecture as well as published externally to other peers.

5.2.3. Comparable Trust

Ideally, published trust values should be syntactically and semantically comparable - that is, equivalent representations in one implementation 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. A lot of discussion has centered on the best semantic representation for trust[1, 18].

There has been no clear consensus as to which trust semantics provide the best fit for applications, 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 not required to be semantically comparable, a constraint can be imposed that trust values must be syntactically comparable by enforcing that they must be represented numerically.

5.2.4. Separation of Internal and External Data

Reflection on our previous work in decentralized emergency response applications [27] revealed the importance of modeling external data separately from internal data. This separation is necessary to help resolve conflicts between externally reported information and internal perceptions. For example, a peer may choose not to trust reported information in favor of information it has perceived and believes to be accurate.

5.2.5. Dependencies of Layers

We have identified three generic functional areas that are essential for managing decentralized applications: data collection, storage, and analysis. We believe that in order to

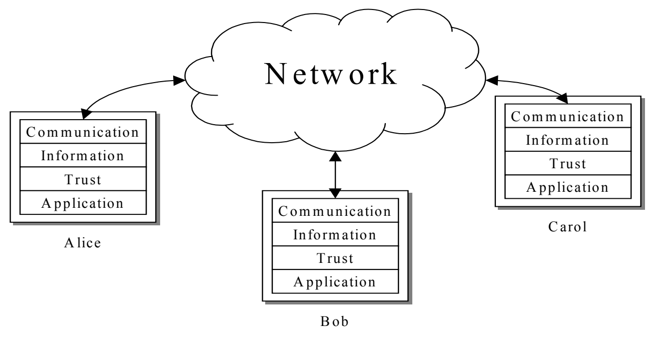


Figure 1. External Architecture

allow evaluation of the specific implementations of these functional groups, they should be isolated at the architectural level. As depicted in Figure 1, the PACE architectural style consists of the following layers of functionality:

1. Communication
2. Information
3. Trust
4. Application

The Communication layer is responsible for performing data collection and transmitting data to other external peers. The PACE architectural style requires that all external communication must be performed through this layer. However, the activity of data collection does not have any dependencies upon data storage or analysis. Therefore, this layer can reside at the top of a C2-based architecture. Externally received data can be sent as notifications, and information to be sent can be processed as requests. These notifications are not implicitly trusted.

All notifications emitted from the Communication layer and all data within the system should 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 manager depends on internal data for its evaluations, it must be below the Information layer.

While the layers described above are generic and may be implemented in an application-independent fashion, the Application layer is domain-specific. This layer is primarily 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 architecture visibility rules discussed in Section 5.1. Each of the components that comprise these layers is discussed further in Section 6.

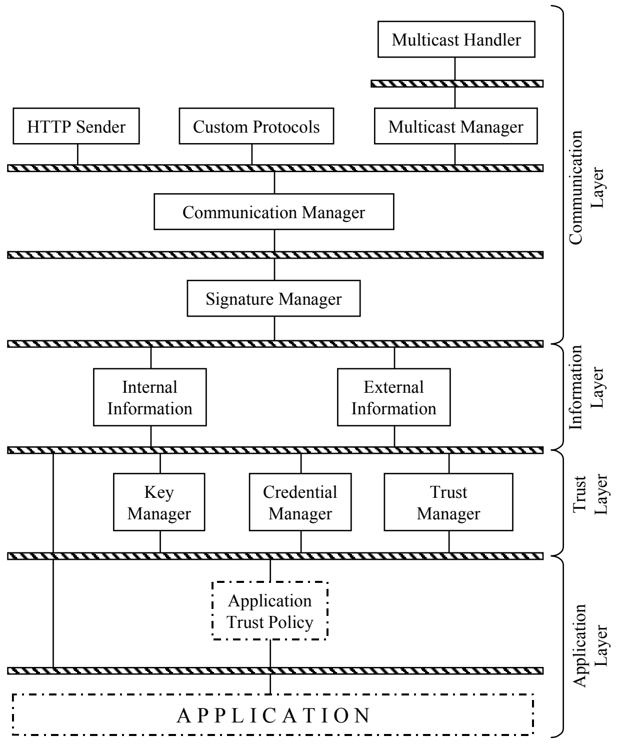


Figure 2. Internal Architecture in PACE

5.2.6. Implicit Trust

PACE assumes implicit trust of components constituting the internal architecture with the exception of the Communication layer. This is due to the fact that the Communication layer is not responsible for validating the messages from other peers. So, any notification sent by the Communication layer can not 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 can not 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 should be implicitly trusted.

Consequently, components within the architecture may treat requests and notifications differently. For example, the Information layer only allows requests to query, update, or delete stored information.

6. PACE Framework

The internal architecture of a peer that can be built using the PACE framework is illustrated in Figure 2. This framework provides the generic functionality of the upper layers which the Application layer may utilize. The system architect may have to select the components that are best suited for the particular application. This section discusses below the constraints on individual components in the PACE framework.

6.1. Communication Layer

The Communication layer handles the interaction of a peer with other peers. This layer has four main functions:

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

In order to achieve maximum flexibility, the nature 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 performs the signing and verification of messages as they pass through the architecture.

6.1.1. Protocol Handler

An architecture may have multiple protocol handlers, such as HTTP, SMTP, or another protocol, which are responsible for transferring data between peers using protocol-specific messages. A protocol handler may also interface with a particular network infrastructure such as FLAPPS[19] and Tarzan[13]. Each handler is responsible for translating internal events into the specific format best suited for the protocol it handles and vice-versa.

There are two categories of protocols that can be supported with PACE: stateful and stateless. Stateful protocols often require a persistent connection, or setup with a specific server or group of nodes. Therefore, a single protocol handler for a stateful protocol can only service a single instance of the protocol. Consequently, multiple protocol handlers for a protocol can be present - each instance will be associated with one instance of the protocol. If a protocol is stateless, then one protocol handler can be reused to service all requests for this protocol.

In order to support dynamically created protocol handlers, PACE relies upon URLs to determine which handler should service the message. Therefore, stateless protocol handlers can service all requests for its associated scheme. However, stateful protocol handlers can only deal with its specific instance of the 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. This component may implement relevant public-key infrastructure standards[10]. Since the PACE architectural style requires that identity should be explicit, the signature manager adds the configured public key of the local peer in the outbound messages. To provide non-repudiation and integrity checking, it also digitally signs this message with the public key. When an event is received on another peer, this public key can be used to verify the signature.

Since the architecture supports multiple protocols, having a signature manager allows for the embedding of signatures within protocols that do not explicitly support digital signatures. Also, for added security, the signature manager may also facilitate the encryption of outbound requests and decryption of incoming notifications.

6.2. Information Layer

As discussed in Section 5, the PACE architectural style creates a distinction between internal information generated within a peer and external information collected from other peers. The PACE framework, therefore, consists of two components: the internal information component that stores requests, and the external information component that stores notifications. Additionally, 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. Certain data may be intended for distribution 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

In contrast with the internal information component, the external information component contains only messages received from other peers and is not meant to be persistent. It is imperative to understand that the external information may be incomplete or include intentionally false information published by another peer. As discussed next, the Trust layer is responsible for assigning trust values on this information.

6.3. Trust Layer

As mentioned before, information received externally can be distinguished as either information or information about another peer. The Trust layer is responsible for implementing the trust management policies of the local peer.

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

Threats	Policies	Key Components	Comments
Impersonation	Signatures	Key manager, Signature manager	Without correct private key, the signature will not validate as coming from the 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 with explicit trust communication and digitally signed messages
Denial of Service	Isolation	Communication layer	By isolating protocols to the Communication layer, malicious attacks can be blocked at this layer
Addition of Unknowns	Untrusted Events Still Seen	Signature manager	Unsigned or incorrectly signed messages are still passed through, but no trust values are assigned
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
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

6.3.1. Key Manager

The key manager component is responsible for generating the unique key pairs that the signature manager will use to sign externally-bound requests. Configured public and private key pairs are stored in the internal information component. If the key manager does not detect a configured public and private key pair, it can generate 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. 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.

6.3.3. Trust Manager

The trust manager assigns explicit trust values to messages received from other peers. This component can leverage different trust models and algorithms such as those discussed in Section 4. Different algorithms may utilize various sources of information, such as pre-existing trust relationships, as necessary to assign a trust value to a peer or message.

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.

6.4.1. Application Trust Policy

There can be several dimensions of trust relationships that can be meaningful in an application. One could be topic-based[1] where, for example, Alice may trust Bob completely when it comes to buying books, but may trust Carol more when it comes to buying cars. Due to these dimensions, there can be multiple ways in which a particular trust value may be computed by the trust model. This requires the relationships between these different trust dimensions to be explicitly defined. This is supported in PACE by the application trust policy component. Additionally, this component may assign trust values based on domain-specific semantic meanings of messages. The use of this component is discussed further in Section 7.1.

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. Initial Evaluation

This section evaluates how the architectural constraints within PACE, introduced in Section 5, can address the threats discussed in Section 3. Additionally, we will describe a problem domain, decentralized auctioning, that we successfully implemented in the PACE architectural style

7.1. Threat Countermeasures

An overview of the threats, associated policies, and components responsible for the countermeasures are summarized in Table 1 and also discussed below.

7.1.1. Impersonation

Since all external communication in the PACE architecture is constrained to the Communication layer, there is a single point where impersonation can be detected. If a malicious peer tries to impersonate a user without the correct private key available or does not digitally sign the message, this deception can be easily identified by the fact that the signature does not verify or exist. 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, while the trust manager can then refuse to assign trust values to revoked public keys even if they have a valid signature.

7.1.2. Fraudulent Actions

Since PACE is designed for open, decentralized architectures, there is little that can be done to prevent the entry of malicious users. However, a peer may issue warnings about malicious actions to others through explicit trust communications. Trust managers of peers that receive these messages can consider these warnings in their local evaluations.

7.1.3. Misrepresenting Trust

Additionally, since PACE facilitates explicit communication of comparable trust values between peers, it is possible for a peer to incorporate trust relationships of other peers. 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.

7.1.4. Collusion

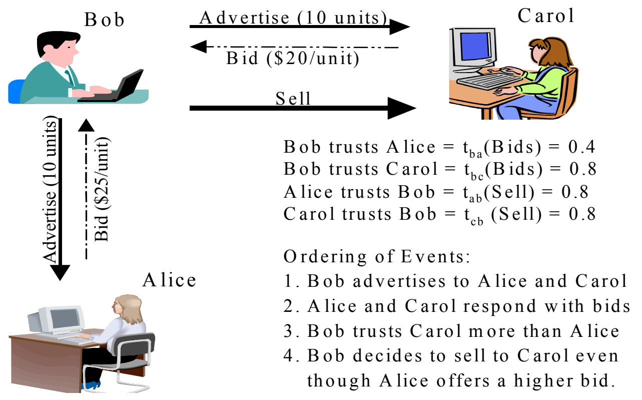
Even if a single peer can be detected that misrepresents its relationships with others, collusion is a greater concern because more malicious peers are involved. However, combined with the ability to detect impersonation in PACE, it has been proven that explicit communication between peers can overcome a malicious collective[17].

7.1.5. Denial of Service

The separation of the Communication layer allows isolation 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 the threat of these attacks. These actions will reduce the impact of the attack on the rest of the architecture.

7.1.6. Addition of Unknowns

Even though a local 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.



Bob trusts Alice = t_{ba}(Bids) = 0.4
Bob trusts Carol = t_{bc}(Bids) = 0.8
Alice trusts Bob = t_{ab}(Sell) = 0.8
Carol trusts Bob = t_{cb} (Sell) = 0.8

Ordering of Events:
1. Bob advertises to Alice and Carol
2. Alice and Carol respond with bids
3. Bob trusts Carol more than Alice
4. Bob decides to sell to Carol even though Alice offers a higher bid.

Figure 3. A Decentralized Auction

7.1.7. Deciding Whom To Trust

In order to automate the process of classifying patterns of interaction, PACE introduces an application trust policy component that allows for automated reinforcements of specified patterns of behavior. The detection of good behavior by this component can cause the trust relationship stored in the Information layer to be increased, while specifically bad behavior could cause a decrease of the associated trust relationship.

7.1.8. 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.

7.2. A Decentralized Auction

We have implemented an auctioning system in the PACE architectural style. In our auctioning model, there is no trusted central controlling authority that controls all the auctions. Instead, either each peer may be responsible for administering every auction it initiates or delegate this responsibility to another trusted peer.

Figure 3 depicts a traditional auctions where a seller advertises the availability of some goods for sale and buyers can then place bids for the items. Our example also supports reverse auctions where a buyer advertises an interest in some quantity of goods and sellers can then place quotes to the buyer to fulfill the requested demand.

7.2.1. Modeling Auctioning in PACE

We now discuss how the auction application was modeled in the PACE architectural style. Since completed auctions in our domain result in two-party transactions, the peer that initially advertises the auction can directly control the bidding process or delegate it to a third party. Each advertisement specifies a URL that indicates where bids may be placed. This URL may refer to the peer itself, or to a trusted third-party that will manage the auction.

It is important to note that the communication mechanisms used during the auctioning may be asymmetrical. Advertisements may be distributed through a broadcast channel, while bids may be placed through a private channel. Both these channels exist as protocol handlers in PACE.

For this application, we use public and private keys to uniquely identify peers in the system. Due to the randomness involved in creating key pairs, each peer is allowed to create their own key pair independently with the assumption that these keys do not conflict with other peers.

Each peer can also assign explicit trust values to a peer and a specific message. Each peer currently uses the same trust model to allow semantic comparison of trust values across peers. Peers may also inform other peers about their current trust relationships through messages. Trust values are also presented to users to allow for manual evaluation of the trust in the messages.

The internal and external separation of the data is enforced by the external information component which strips any part of an externally received notification which indicates its trust. This is to prevent malicious peers from trying to trick the peer into believing their evaluations of trust.

The auction application utilizes a framework that provides an implementation of the Communication, Information, and Trust layers of PACE. This framework is generic and does not contain any domain-specific constraints. Instead, as described in Section 5, we restrict the domain-specific functionality to the Application layer.

7.2.2. Implementation Details

Since the PACE architectural style builds upon the C2 style, applications written in PACE can reuse technology designed for C2. In particular, we described the auction architecture using xADL[8] and used the c2.fw framework to implement the auction in Java.

In order to manage public and private key pairs, the java.security framework was used in the key manager and signature manager components. For the purposes of this application, we used a multicast UDP protocol handler provided in the c2.fw framework within the Communication layer to broadcast a message sent by a peer to all other peers in the system.

The Information layer contained separate internal and external information components which stored data in a relational database to allow storage and queries. The trust manager in the Trust layer incorporated a simplistic trust model that divides trust into two dimensions: the buying and selling of items.

In order to facilitate interaction with the user, a graphical user interface was implemented. This interface enabled the user to create regular and reverse auctions, bid for advertised items, and also review previously perceived actions by displaying all recorded events stored in the Information layer. Furthermore, it allowed the user to directly modify the trust levels of a particular message or a peer to address out-of-band knowledge.

Additionally, our example was able to validate that following the PACE architectural style can address several other threats discussed in the previous section. In particular, the signature manager tags a message as verified if and only if there is a valid digital signature of the message. Since all received messages must pass through the signature manager, all attempts at impersonation are exposed. However, we used a trust model which did not implement transitivity, so we were not able to verify whether the detection of misrepresentation of trust works in practice.

8. Discussion

There is an abundance of technologies that can help address the 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 the architectural level 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 as long as the constraints introduced by the PACE style are strictly followed and a suitable trust model is adopted, threats of decentralization can be addressed.

We have also implemented a decentralized auction system using a generic reusable framework constrained by the PACE architectural style. Experiments with this prototype have illustrated the feasibility of the PACE architectural style as a guide to integrating trust into decentralized applications with explicit benefits.

9. Acknowledgements

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

10. 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. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the 9th International Conference on Cooperative Information Systems.* Trento, Italy, September 5-7, 2001.

[3] 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.

[4] 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. <http://www.cse.msu.edu/~cse870/Materials/Frameworks/tosem-92.pdf>.

[5] Carzaniga, A., *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.

[6] Chen, R. and Yeager, W. *Poblano: A Distributed Trust Model for Peer-to-Peer Networks.* <http://www.jxta.org/docs/trust.pdf>.

[7] Damiani, E., *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.

[8] Dashofy, E.M., *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.

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

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

[11] Dragovic, B., *et al. XenoTrust: Event-based distributed trust management.* Report, <http://www.cl.cam.ac.uk/~ek247/research/XenoTrust-DEXA.pdf>.

[12] Fielding, R.T. *Architectural Styles and the Design of Network-based Software Architectures.* Ph.D. Thesis. Information and Computer Science, University of California, Irvine, 2000. <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

[13] Freedman, M. and Morris, R. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security.* Washington, DC, November, 2002.

[14] Gray, E., *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. <http://www.cs.tcd.ie/publications/tech-reports/reports.02/TCD-CS-2002-66.pdf>.

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

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

[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] Mui, L., *et al.* A Computational Model for Trust and Reputation. In *Proceedings of the 35th Hawaii International Conference on System Sciences.* Hawaii, 2002.

[21] 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.

[22] 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.

[23] 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. <http://citeseer.nj.nec.com/perry92foundation.html>.

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

[25] Shaw, M. Architectural issues in software reuse: it's not just the functionality, it's the packaging. In *Proceedings of the 1995 Symposium on Software Reusability.* 20, August, 1995.

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

[27] 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.

[28] Taylor, R.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.

[29] Yu, B. and Singh, M. An Evidential Model of Distributed Reputation Management. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems.* p. 294-301, 2002.

[30] Zacharia, G. and Maes, P. Collaborative Reputation Mechanisms in Electronic Marketplaces. In *Proceedings of the 32nd Hawaii International Conference on System Sciences.* Hawaii, 1999.