Crossroads
The ACM Student Magazine

Architecting Trust-Enabled Peer-to-Peer File-Sharing Applications

By ***Girish Suryanarayana***, ***Mamadou H. Diallo***, ***Justin R. Erenkrantz***, and ***Richard N. Taylor***

Abstract

Decentralized peer-to-peer (P2P) resource sharing applications lack a centralized authority that can facilitate peer and resource look-ups and coordinate resource sharing between peers. Instead, peers directly interact and exchange resources with other peers. These systems are often open and do not regulate the entry of peers into the system. Thus, there can be malicious peers in the system who threaten others by offering Trojan horses and viruses disguised as seemingly innocent resources. Several trust-based solutions exist to address such threats; unfortunately there is a lack of design guidance on how these solutions can be integrated into a resource sharing application. In this paper, we describe how two teams of undergraduate students separately integrated XREP, a third-party reputation-based protocol for file-sharing applications, with PACE, our software architecture-based approach for decentralized trust management. This was done in order to construct trust-enabled P2P file-sharing application prototypes. Our observations have revealed that using an architecture-based approach in incorporating trust into P2P resource-sharing applications is not only feasible, but also significantly beneficial. Our efforts also demonstrate both the ease of adoption and ease of use of the PACE-based approach in constructing such trust-enabled decentralized applications.

Introduction

P2P file-sharing applications allow peers at the edges of the network to interact and exchange resources, such as documents or media, directly with other peers in the

system. Existing architectures for such applications either use some notion of a centralized authority or are completely decentralized. File-sharing applications like Napster [1] use a centralized index that helps coordinate the search and exchange of resources between peers.

In contrast, decentralized P2P file-sharing applications, such as those based on Gnutella [2], are characterized by the absence of such a central authority. In such applications, each peer directly queries its neighboring peers for resources. When a peer receives such a query from the user, it forwards the query to its neighboring peers. Each peer also checks for the existence of the requested file (resources) and returns the result to the requesting peer. Upon receiving this response, the original peer can elect to download the file from one of the responding peers.

Although such decentralized file-sharing applications offer significant benefits: no single point-of-failure, increased robustness, and allowing users at the edge of the network to directly share files with each other, they also can become host to several attacks by malicious peers. This is because such decentralized P2P file-sharing applications are open, meaning anyone can join and leave the system, without restrictions, at any time. Peers with malicious intent may offer tampered files or disguise Trojan horses and viruses as legitimate files to download. A study [3] in January 2004 reported that 45% of 4,778 executable files downloaded through the Kazaa [4] file-sharing application contained malicious code, such as viruses or Trojan horses. In addition to harming the host computer, malicious code can relay itself to other peers in the network.

Clearly, there is a pressing need for mechanisms that will help determine the trustworthiness of both peers and the resources offered by them. However, in the absence of a centralized authority that can regulate the entry, manage and coordinate the peers, and implement suitable trust mechanisms, it becomes the responsibility of each decentralized peer to adopt suitable measures to safeguard itself. Decentralized, reputation-based trust schemes are one such measure that helps each peer determine the trustworthiness of other peers and resources. Although such schemes have been investigated to some extent by researchers [5], there is little guidance on how these schemes can be incorporated into a decentralized peer-to-peer file-sharing application.

In this paper, we describe our software architecture-based approach for the construction of a trust-enabled decentralized P2P file-sharing application. Two teams of undergraduate students, working separately, have integrated XREP [5], a third-party reputation protocol for file-sharing applications, into the PACE (Practical Architectural

approach for Composing Egocentric trust) [6] architectural style to create two separate trust-enabled file-sharing applications. Our observations have revealed the feasibility and benefits of the PACE guidelines in properly constructing a trust-enabled decentralized application as well as demonstrated the ease of use and adoption of the PACE style.

The rest of the paper is structured as follows: relevant related work is discussed next, followed by descriptions of the PACE architectural style and the XREP reputation protocol. We then discuss the designs and compare the approach followed by the two teams. We conclude with a summary of our evaluations and a brief discussion.

Related Work

This section discusses relevant research in the area of trust management and P2P file-sharing applications. Existing research in trust and reputation management can be roughly classified into two main topics: models that encapsulate algorithms and techniques to establish trust relationships between entities, and architectures that facilitate the assimilation of these trust models into an application.

Trust and Reputation Management

The earliest decentralized trust management systems [7] regulated access control using credential verification and pre-defined application policies. Examples of such systems that verify a user's credentials against pre-defined application policies to determine whether the user can access a system [8] include PolicyMaker [9] and REFEREE [10].

Reputation-based trust management systems establish trust relationships between entities based on their reputations [11]. Abdul-Rahman and Hailes define reputation as an expectation about an individual's behavior, based on information about, or observations of, its past behavior [12]. Thus, a reputation might be derived either through personal past experience, experience of other entities in the system, some other means, such as word-of-mouth, out-of-band communication, or even a combination of all the above. Websites such as eBay [13] and Amazon [14] enable users to record their experience with other users, which serves as reputation information for other users. Given that this data is stored on servers maintained by a single centralized authority, such systems are not truly decentralized. Instead, all users are completely dependent upon this authority, and, as such, they are expected to

completely trust the centralized authority. Although this works well for websites such as eBay and Amazon, it poses a serious problem in applications where either such a centralized infrastructure is unavailable or peers do not trust the centralized authority. In a decentralized reputation-based system, peers store their own past experience and exchange this with other peers in order to make better informed trust-related decisions.

File-sharing

Napster and Gnutella were the first file-sharing applications. Freenet [15] and other P2P-based resource sharing applications soon followed. Given the nature of these applications, interest and attention in the context of these file-sharing applications was mostly focused on the issues of scalability, privacy, anonymity, robustness, efficient use of bandwidth, and free-riding. The essential issue of safeguarding against malicious peers and harmful content was mostly overlooked.

However, the spread of viruses and Trojans, along with the presence of unreliable resources and peers, have highlighted the need for secure file sharing. Several reputation-based trust schemes for decentralized file-sharing applications have therefore been recently explored. XREP is a protocol for gathering and evaluating reputation information in file-sharing applications. XREP involves using the reputation of not only the resource being offered, but also that of the peer offering the resource. PET [16] is another personalized trust model for file sharing that has two main parts, namely, reputation and risk evaluation. Reputation represents the cumulative assessment of the long-term behavior. Risk evaluation refers to the opinion of the short-term behavior. PET is thusly different from other reputation models that solely assess reputation without risk analysis. Other reputation-based models for decentralized file-sharing applications include PeerTrust [17], EigenRep [18], Reputation Management Model [19], and SupRep [20].

This paper focuses on describing our efforts at integrating the XREP file-sharing protocol within the boundaries of the PACE software architectural style. However, it should be noted that the PACE-based architectural approach is model-independent and provides an effective basis for integrating different types of reputation-based models, including those discussed above, within a decentralized file-sharing application.

PACE Architectural Style

PACE addresses the concerns of trust management in open decentralized applications [6]. PACE is designed to facilitate the incorporation of trust models into the architectures of decentralized peers. In particular, PACE provides comprehensive guidance about the components that should be included inside a peer, their arrangement within a peer, and their interactions.

PACE imposes a set of constraints on the structure and behavior of components in the system to address trust management. Digital identities are used to identify peers in the system. This allows a single user to pose as multiple peers by using multiple electronic identities. It also affords a group of users within an organization to pose as a single peer by using a single identity. Trust information about each digital identity is separately determined and maintained, irrespective of the physical identities it represents. Given that the information received from other peers in a decentralized system may possibly be faulty or incomplete, PACE makes a specific distinction between the beliefs internal to a peer and the beliefs communicated externally by other peers. Hence, PACE explicitly separates the storage of internal and external beliefs through the use of distinct internal and external information repositories.

PACE requires trust to be explicit between components in the peer's architecture. Perceived trust as well as published trust need to be shared among internal components that make up a peer because they are responsible for local decision-making. PACE recognizes four essential functional aspects of any decentralized peer: communication, data, trust, and application. The communication aspect is responsible for interaction with other peers. The data aspect is responsible for storing application and trust-specific data. The trust aspect deals with trust-specific functionality including trust computation. Finally, the application aspect deals with application-specific functionality. The dependencies of these functional aspects among each other are as follows: communication does not depend upon any other aspect, the trust aspect depends upon the data aspect for trust computation, and the application aspect requires the services of all the other aspects.

PACE is built on top of C2 [21], an event-based architectural style. The C2 architecture is composed of components and connectors that are organized into layers. Components in a layer are only aware of components in the layers above and have no knowledge about components in layers below. Components communicate with each other using two types of asynchronous event-based messages: requests and notifications. Request messages travel up the architecture, while notification messages travel down the architecture.

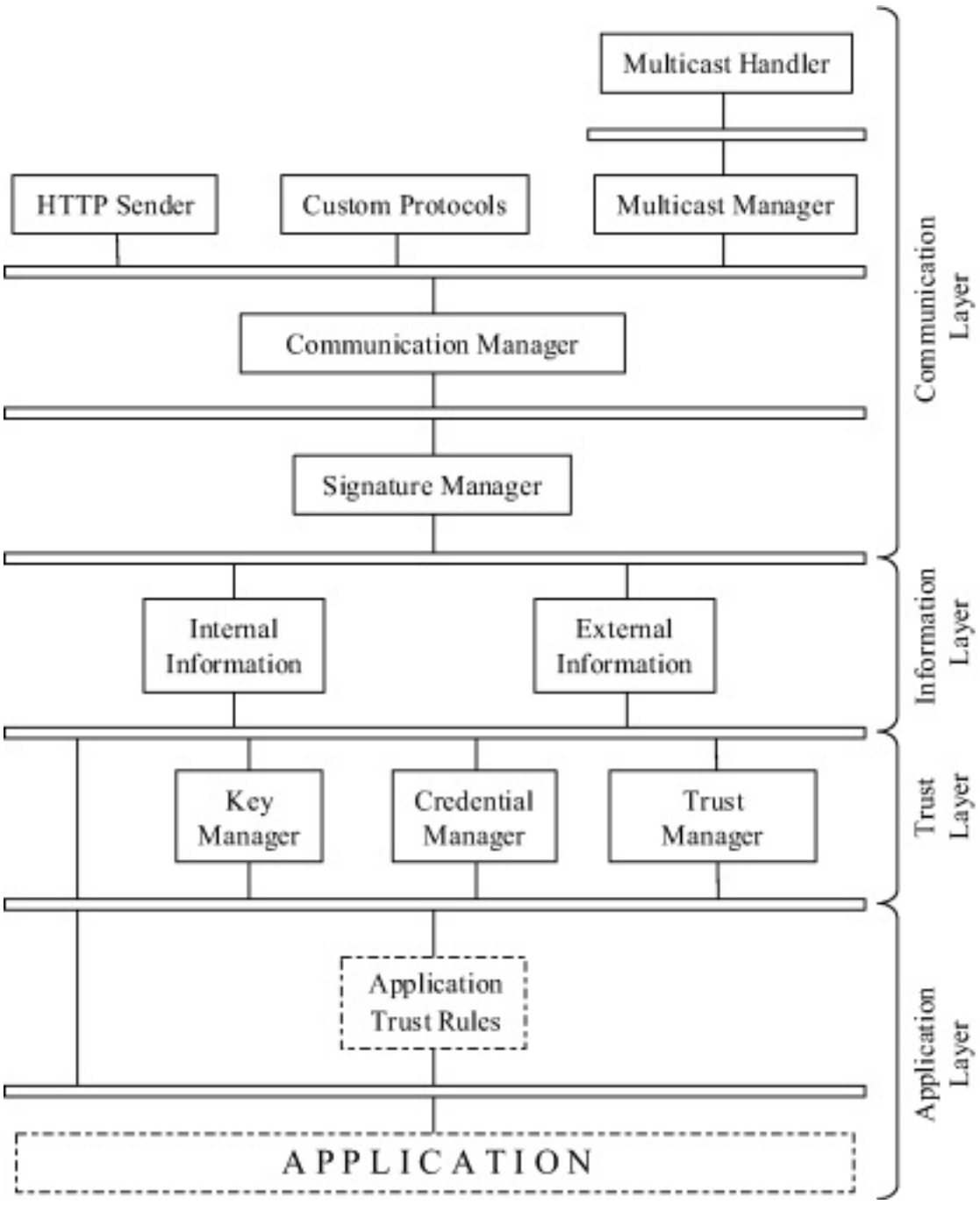

Figure 1: Internal Architecture in PACE

As mentioned earlier and as shown in Figure 1, PACE divides the components of a peer's architecture into four layers: Communication, Information, Trust, and Application. A detailed description of each of these layers and the components belonging to these layers is presented in "PACE: An Architectural Style for Trust Management in Decentralized Applications" [6]. The Communication layer is responsible for handling the communication between peers in the system. It is mainly

composed of three components: the Protocol Handler, the Communication Manager, and the Signature Manager. The Protocol Handler enables multiple network communications that are responsible for translating internal events into the format understood by the associated external protocol and vice-versa. The Information layer is used for storing data in the system and is composed of two components. The Internal Information component stores request messages, whereas the External Information component stores notification messages. The Trust layer's function is to hold credential and reputation-based trust models that manage the trust for peers locally. This layer is composed of the Key Manager, the Credential Manager, and the Trust Manager. The Key Manager is application independent, whereas the Credential and Trust Manager need to be implemented according to the trust models used. The Application layer encapsulates application-specific functionality. Although components in the other layers can be reused across different applications, components in the Application layer are application dependent and hence not reusable across application domains.

PACE is only concerned with trust relationships between peers, not with trust relationships between components in the internal architecture. Further, because external beliefs are communicated through notifications sent from the Communication layer, notifications arising from the Communication layer are not implicitly trusted. Internal beliefs originally communicated through explicit request messages are trusted implicitly. In order to facilitate the comparison of trust values, PACE also requires that trust values be represented numerically, without imposing any constraint on their semantic.

The adoption of the PACE architectural style is not binding upon all peers participating in the system, nor is it an absolute requirement for the system to function. In other words, peers built in the PACE style can still communicate and interact with other peers that are not built in the PACE style. However, adoption of the PACE style offers several significant benefits that can be leveraged, such as structured guidance for the design of decentralized systems, principles that guide the incorporation of trust management, and reusability of components across applications and trust models. Additionally, adopting the PACE style does not require the sub-architecture of PACE components to necessarily adhere to the PACE style. It is noteworthy that PACE does not provide a complete trust management solution by itself. Instead, it acts as a placeholder for different trust and reputation models such as those mentioned in the Related Work section.

The PACE architectural style has been used to build two decentralized applications: a decentralized auctioning system, and a common operational picture that is used by organizations in an emergency response scenario to share real-time data with each other [6]. PACE components exist for both these applications and differ primarily in the implementation of the application layer.

XREP Reputation Model

The XREP [5] approach primarily focuses on P2P file-sharing applications. In addition to modeling peers' reputations in the system, each peer also evaluates resources accessed from peers. A distributed polling algorithm is used to allow these reputation values to be shared among peers, so that a peer requesting a resource can assess the reliability of the resource offered by a peer prior to using it.

Each peer in the application is termed a "servent" because of its ability to play the roles of both server and client. Each servent maintains its own experience information on resources and other peers that it can share with other servents upon request. This information is stored in two repositories: a resource repository that associates a unique resource ID with a binary reputation value, and a servent repository that associates the number of successful and unsuccessful downloads with each unique servent ID.

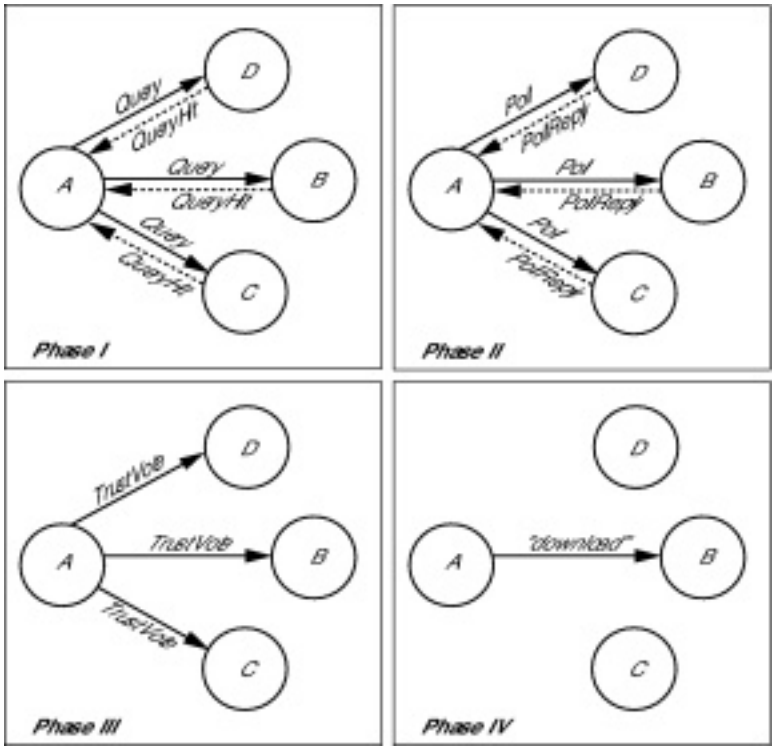

Figure 2: Phases in XREP

XREP is a distributed protocol that allows these reputation values to be maintained and shared among the servents. It consists of the following phases as illustrated in Figure 2: resource searching, resource selection and vote polling, vote evaluation, best servent check, and resource downloading. Resource searching is similar to the technique used in Gnutella and involves a servent broadcasting to all its neighbors a Query message containing search keywords. When a servent receives a Query message, it responds with a QueryHit message. In the next phase, upon receiving QueryHit messages, the originator selects the best matching resource among all possible resources offered. At this point, the originator polls other peers using a Poll message to enquire their opinion about the resource or the servent offering the resource. Upon receiving a Poll message, each peer may respond by communicating its votes on the resource and servents using a PollReply message. These messages help distinguish reliable resources from unreliable ones, trustworthy servents from fraudulent ones.

In the third phase, the originator collects a set of votes on the queried resources and their corresponding servents. Then it begins a detailed checking process that includes verification of the authenticity of the PollReply messages, , using cluster computation to guard against the effect of a group of malicious peers acting in tandem, and sending TrustVote messages to peers that request confirmation on the votes received from them. At the end of this checking process, based on the trust votes received, the peer may decide to download a particular resource. However, because multiple servents may be offering the same resource, the peer still needs to select a reliable servent. This is done in the fourth phase where the servent with the best reputation is contacted to check the fact that it exports the resource. Upon receiving a reply from the servent, the originator finally contacts the chosen servent and requests the resource. It also updates its repositories with its opinion on the downloaded resource and the servent who offered it.

Approach

In this section, we describe our approach towards the construction of an XREP-based file-sharing application. We organized two teams, each consisting of four undergraduate students, which separately constructed two file-sharing applications. Each team first designed a file-sharing peer in the PACE architectural style and integrated XREP components into the peer architecture. Next, each of these file-sharing applications was subjected to several test scenarios and results observed were

used to draw various conclusions about PACE and XREP.

Peer Design

In order to distinguish the two teams, we named the two undergraduate teams Zot-n-Share and AntShare. Based on the same requirement document, both teams independently designed the architecture of the trust-enabled file-sharing application. As mentioned earlier, the XREP protocol consists of five phases: resource searching, resource selection and vote polling, vote evaluation, best servent check, and resource downloading. A significant aspect of each of these phases is the explicit message passing operations between peers. Both teams, therefore, based their architectures primarily on the different messages that need to be exchanged among peers in the system.

The teams also followed the guidelines set by the PACE architectural style. In PACE, the Communication, Information, and Trust layers are application independent, whereas the Application layer needs to be specifically designed for the particular application. As a result, the design effort for the file sharing was mainly concentrated on the Application layer. It should be noted that although components of the top three layers already exist as part the PACE framework, users have the freedom to replace them with their own components.

Team Zot-n-Share

The Zot-n-Share team chose to modify several existing PACE components while still adhering to the PACE style. As described earlier, the main design effort was devoted towards the design of the Application layer. Team Zot-n-Share's design approach involved decomposing the application layer into seven different components organized into three sub-layers as illustrated in Figure 3. The first sub-layer contains only the Application Trust Rules component.

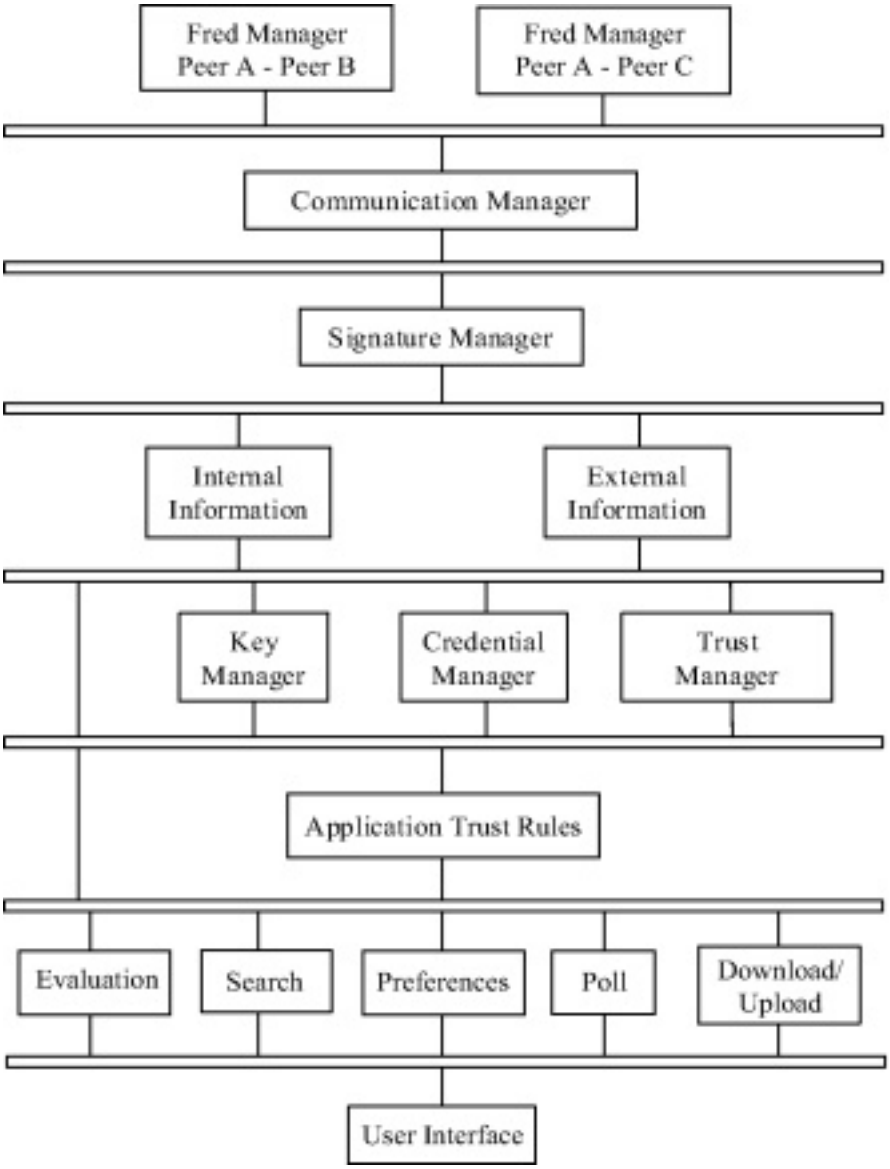

Figure 3: Zot-n-Share Peer Architecture

The second sub-layer is composed of the following five components: Search, Poll, Evaluation, Download/Upload, and Preferences. The Search component is responsible for issuing queries for resources, displaying query results to the user through the user interface, and saving query results in the Internal Information component. Once queries about a resource are received and aggregated by the Search component, the Poll component sends poll messages to other peers to acquire their opinion about the resource and the servents offering that resource. Poll reply messages are automatically stored in the External Information component and are also displayed to the user through the user interface. The Evaluation component is responsible for checking the authenticity of poll reply messages and analyzing the trust information received in the

form of poll reply messages. The Download/Upload component is used for downloading the resource from the servent selected by the user. In addition, this component is used for uploading resources to peers that are trying to download the resource. The Preferences component enables the user to specify whether he/she wants to connect to the P2P network, the message hop count, the number of uploads and downloads allowed, and whether he/she wants to enable or disable resource sharing. These preferences and configuration information are stored in the Internal Information.

The third sub-layer contains only one component, the User Interface. This component is basically a graphical user interface that enables the transfer of information entered by the user to the components in the second sub-layer and vice-versa.

Team Antshare

Unlike the first team, Team AntShare decided to reuse all the existing PACE components from the Communication, Information and Trust layers without any modifications. The only exception was the Trust Manager component, which was modified to enable the evaluation of poll results in the XREP model. The Application layer was decomposed into eleven different components organized into three sub-layers as shown in Figure 4.

The first sub-layer contains only the Application Trust Rules component. The second sub-layer consists of the following six components: Library, Poll, Transfer, Preferences, Evaluation, and Search. The Library component maintains the list of files that have been downloaded and that can be shared with other peers. However, these files are persistently stored in the Internal Information. The Search component is responsible for issuing resource and Poll queries and displaying received responses to those queries through the user interface. The Poll component responds to Poll query messages by sending PollReply messages. The Transfer component is responsible for uploading and downloading files, displaying uploaded and downloaded files to the user interface, saving downloaded files to the Internal Information storage, and deleting files from the Internal Information. The Preferences component manages login information, and enables the user to specify whether to automatically connect to the P2P network, the number of hops, the number of permissible uploads and downloads, and the destination for the library folder. The Evaluation component is responsible for checking the authenticity of PollReply messages and analyzing peer votes received about resources.

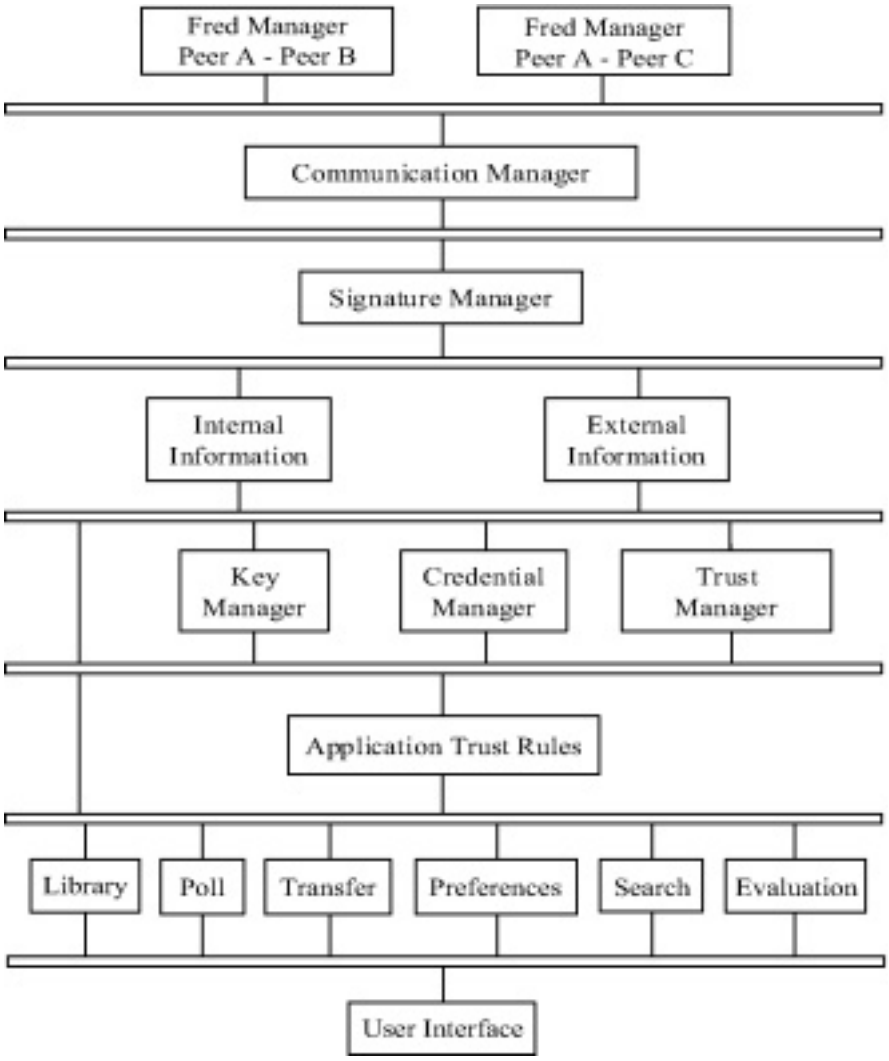

Figure 4: AntShare Peer Architecture

Evaluation

The goal behind incorporating the XREP trust model within a decentralized file-sharing system was to safeguard users against the threat of malicious resources as well as peers offering such resources. Consequently, peers in the system can share trust information with each other using the XREP trust model and autonomously devise their own protection by using their trust information to guard against malicious resources and peers. Below, we describe the file-sharing applications built by each team and discuss their various functional features.

Team Zot-n-Share

The Zot-n-Share system included a complete implementation of the XREP protocol. It conforms to the requirement specification, except for checking the authenticity of PollReply messages. The GUI is composed of three tabs that cover the main functionality of the system. These tabs are Search, Download/Upload, and Evaluate.

The Search tab is composed of the following commands: Search, Poll, Calculate, and Download. To search for a resource in the system, the peer types the file name in the space provided and initiates the Search command. The Search command broadcasts the search query to the neighboring servents. Upon receiving this query, the neighboring servents respond if they have the requested file. Otherwise, they forward the query to their neighbors. Upon receiving responses to the query, the requestor peer can use the Poll command to query trust information about the peers that offer the resource. Based on the responses in the form of votes received from queried peers, the peer requestor can evaluate the reputation of the provider peers using the Calculate command. The requestor can then choose a servent peer for downloading the resource based on this reputation determination.

The Download/Upload tab displays a list of files downloaded from other peers and a list of files uploaded to other peers. The Evaluate tab shows a list of servents and resources with their reputations, and it also provides a Set Evaluation Information command used to manually modify the servents and resources reputations. The peer uses the Preferences command from the File menu to set an upper threshold on the number of files that can be downloaded or uploaded simultaneously.

Team AntShare

Similar to the Zot-n-Share system, the AntShare system also included a successful implementation of the XREP model, except for checking the authenticity of received PollReply messages. The AntShare GUI is composed of the following five tabs that encapsulate the functionality of the system: Search, Transfer, Library, Preferences, and About.

The Search tab consists of the Search, Poll, and Download commands. To search for a file, the requestor peer enters the name of the file in the space provided and executes the Search command. The search command sends out a request for the resource to the

neighboring servents. After gathering the information about the servents that possess the requested resource, the requestor peer uses the Poll command to send Poll messages to other servents. This is done to gather their votes regarding the resource as well as their votes regarding the servents that offer the resource. Based on the PollReply responses, the requestor can decide upon a servent to download the resource. Once a good servent is selected, the peer executes the Download command to download the resource. Unlike the Zot-n-Share system, PollReply responses received were not aggregated before computing a single reputation value for a resource or a servent. Instead, with each incoming response, the corresponding reputation values of servents and resources were recomputed and new values displayed to the user.

The Transfer tab displays a list of downloaded files and a list of uploaded files. The Library tab shows a list of files shared by the peer. The Preferences tab is divided into two sections: general preferences and preferences related to the file-transfer operations. The general preferences section allows peers to set the login preferences and specify the number of hops for the polling process. The preferences for file-transfer operations enables the specification of the destination folder for download, and the maximum number of files that can be downloaded or uploaded at the same time. The About tab contains the information about the file-sharing system and its developers.

Discussion

Our experience with building two decentralized file-sharing application prototypes by integrating XREP within the PACE architectural style and their subsequent experimentation helped us draw several conclusions. First, the successful integration of XREP with PACE to construct a trust-enabled decentralized file-sharing application underlined the applicability and feasibility of the PACE architectural style and its guidelines for composing trust-aware decentralized applications.

Second, the different architectures adopted by the two teams in the design phase showed that the PACE style is not overly constraining, in spite of imposing a disciplined approach to composing trust management. In fact, several alternative architectures for the same application are possible. Additionally, because team AntShare built new implementations of some existing PACE components, it demonstrated that adhering to the PACE style does not restrict developers from using existing PACE components. In fact, off-the-shelf components built in other architectural styles can be easily plugged in as well. The bottom-line is that as long as the PACE style is adhered to, the benefits provided by the PACE style are leveraged regardless of whether new or off-the-shelf

components are used. Further, team Zot-n-Share reused existing PACE components while building their prototype, thus demonstrating that PACE components could be reused across multiple application domains.

Third, the successful completion of the project by two teams of undergraduate students in 10 weeks highlights the ease of adoption and use of the PACE style. (It should be pointed out that neither team implemented the authentication and confirmation of the PollReply messages in XREP, due primarily to the short timeline.) Discussions with team members of both teams revealed that the primary technical problem they faced lied in comprehending how event-based communication works. Coming from a procedure call-based communication perspective, understanding and using an event-based asynchronous messaging paradigm required a considerable length of time. Similarly, it took a significant length of time to acquire a general understanding of software architectural styles and, in particular, the constraints of the PACE style. However, once the concepts of event-based communication and the PACE architectural style were understood, the team members found it easy to understand and adopt the PACE style and build the file-sharing prototypes within the 10-week period. This was also facilitated through the utilization of reusable PACE components that made it easy to quickly build trust-enabled applications from scratch. Based on the above testimony of the undergraduate team members, we believe that, for developers who are familiar with event-based messaging, adopting the PACE style will take considerably less time.

It should be pointed out that an additional goal of the project was to make the two PACE-based file-sharing prototypes interoperable. Our objective was to investigate whether the Zot-n-Share and AntShare clients could directly interact and share files with each other, and, if so, what the constraints that would restrict this interoperability were. However, due to non-technical issues such as team coordination and personnel problems, this objective could not be realized during the course of this project. Nevertheless, we believe that it is certainly feasible to make independently built PACE-based prototypes interoperable, and we plan to explore this issue in the future.

Acknowledgements

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

References

1

Napster. URL: **http://www.napster.com/**

2

Gnutella. URL: **http://www.gnutella.com/**

3

Zetter, K. *Kazaa Delivers More than Tunes.* 2004.

4

Kazaa. URL: **http://www.kazaa.com/**

5

Damiani, E., et al. Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (Nov. 18-22, Washington DC) ACM Press, 2002, pp. 207-216.

6

Suryanarayana, G., et al. PACE: An Architectural Style for Trust Management in Decentralized Applications. In *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture* (Jun. 12-15, Oslo, Norway) IEEE Press, 2004, pp. 221-230.

7

Blaze, M., et al. Decentralized Trust Management. In *Proceedings of the IEEE Symposium on Security and Privacy* (May. 6-8, Oakland, Calif.) IEEE Press, 1996, pp. 164-173.

8

Grandison, T. and Sloman, M. A Survey of Trust in Internet Applications. *IEEE Communications Surveys & Tutorials*, 3, 4 (2000), 2-16.

9

Blaze, M. and Feigenbaum, J. Managing Trust in an Information Labeling System. *European Transactions on the Telecommunications*, 8 (1997), 491-501.

10

Chu, Y., et al. REFEREE: Trust Management for Web Applications. *World Wide Web Journal* (1997) pp. 127-139.

11

Resnick, P., et al. Reputation Systems. *Communications of the ACM.* 43, 12 (2000) pp. 45-48.

12

Abdul-Rahman, A. and Hailes, S. A Distributed Trust Model. *In Proceedings of New Security Paradigms Workshop* (Sept. 23-26, Great Langdale, Cumbria, UK) ACM Press, 1997, pp. 48-60.

13

eBay. URL: **http://www.ebay.com/**

14

Amazon. URL: **http://www.amazon.com/**

15

Langley, A. Freenet, In *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, A. Oram, Editor. 2001, O'Reilly. pp. 123-132.

16

Liang, Z. and W. Shi. PET: A Personalized Trust Model with Reputation and Risk Evaluation for P2P Resource Sharing. In *Proceedings of the Hawaii International Conference on System Sciences* (Jan. 3-6, Waikoloa Village, Hawaii) 2005, pp. 201b-210b.

17

Xiong, L. and L. Liu. PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. *IEEE Transactions on Knowledge and Data Engineering* 16, 7 (2004), 843-857.

18

Kamvar, S., et al. EigenRep: Reputation Management in P2P Networks. In *Proceedings of the Twelfth International World Wide Web Conference* (May. 20-24, Budapest, Hungary) 2003.

19

Iguchi, M., et al. Managing Resource and Servent Reputation in P2P Networks. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences* (Jan 5-8, Waikoloa Village, Hawaii) 2004.

20

Chhabra, S., et al. A Protocol for Reputation Management in Super-Peer Networks. In *Proceedings of the 15th International Workshop on Database and Expert Systems Applications* (Aug. 30-Sept. 3 Zaragoza, Spain) 2004.

21

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

Biography

Girish Suryanarayana is a PhD student in the Department of Informatics at the University of California, Irvine. His research interests include software architectures for decentralized peer-to-peer applications and decentralized trust and reputation management. Suryanarayana has a BE in electrical and electronics engineering from the Birla Institute of Technology and Science, Pilani and an MS in information and computer science from the University of California, Irvine. He is a member of the ACM

and the IEEE. Contact him at sgirish@ics.uci.edu.

Mamadou H. Diallo is a second year Masters student in the Department of Informatics at the University of California, Irvine. His research interests include software architectures and ubiquitous and pervasive computing. Diallo holds a Bachelor degree in information and computer science from the University of California, Irvine. Contact him at mdiallo@ics.uci.edu.

Justin R. Erenkrantz is a PhD student in the Department of Informatics at the University of California, Irvine and is a director of the Apache Software Foundation. His research interests include software architectures, specifically representation state transfer-based application architectures. He has also contributed to the development of the Apache HTTP Server and Subversion. Erenkrantz has an MS in information and computer science from the University of California, Irvine. He is a member of the ACM and the IEEE. Contact him at jerenkra@ics.uci.edu.

Richard N. Taylor is a professor of information and computer sciences at the University of California, Irvine, a member of UCI's Department of Informatics, and the director of the Institute for Software Research, which fosters innovative basic and applied research in software and information technologies through partnerships with industry and government. His research interests center on software architectures, especially event-based and peer-to-peer systems, and how they scale across organizational boundaries and decentralized applications. Taylor has a PhD in computer science from the University of Colorado, Boulder. He is a fellow of the ACM. Contact him at taylor@ics.uci.edu.