A Framework for Building Reputation Systems

Phillip J. Windley, Ph.D.
Dept. of Computer Science
Brigham Young University
Provo, UT
http://xri.net/=windley

Kevin Tew
Dept. of Computer Science
Brigham Young University
Provo, UT
kevin@tewk.com

Devlin Daley
Dept. of Computer Science
Brigham Young University
Provo, UT
devlin.daley@gmail.com

ABSTRACT

This paper introduces a set of principles for governing the design and operation of online reputation systems. We also introduce the design, architecture, and implementation of a flexible, general-purpose framework, called *Pythia*, for building reputation systems. We give a sample application of our framework to illustrate its use.

Categories and Subject Descriptors

H.4.3 [**Information System Applications**]: Communications Applications; K.4.2 [**Computers and Society**]: Social Issues; H.4.m [**Information System Applications**]: Types of Systems; Decision support

Keywords

reputation, identity, framework, blogging

1. INTRODUCTION

Most online interactions are devoid of many of the cues that people use in the physical world to make judgments about the character, stability, reliability, etc. of people, systems, and other entities. Yet these cues are critical to sophisticated interactions and transactions. Online reputation systems attempt to remedy this situation.

Reputation has been studied in the context of philosophy [13], psychology [3], economics [17], biology [28], and sociology [29]. Much of that work is informative of how reputation works in the physical world and can guide our thinking about reputation online.

Online reputation can be accessed informally or formally. Informal assessments are made by people from their experience on the Internet. For example, I may not trust email I get from the domain `hotmail.com` because I've received substantial Spam from addresses in the domain in the past. Formal systems collect and process online information about specific identifiers in an attempt to calculate reputation scores in a more systematic manner.

Computational models of reputation are quite recent[24], yet diverse strategies have been employed to meet specific needs. In *Reputations Systems: Facilitating Trust in Internet Interactions*,[22], Resnick et al. define a reputation system as a system that "collects, distributes, and aggregates feedback about participants' past behavior."

WWW2007, May 8–12, 2007, Banff, Canada.

The primary domains where computational reputation has been explored are in artificial intelligence, peer-to-peer networking, and electronic commerce. A number of systems have been devised to provide reputation information for commercial Web sites. [8] Most of these systems have limited scope and are built for a single context. [20] They are effective in their specific domain, but lack generality.

Recent advances in user-centric identity systems such as CardSpace, OpenID, LID, SXIP and i-names provide a firm foundation for the establishment of general-purpose, online reputation systems. [21, 5, 27, 32] This paper presents a set of principles that we believe should govern the operation of such systems based on our own work and the work of others.

The paper also presents the design and architecture of a reputation framework that is consistent with these principles and that can be specialized to particular tasks. The framework, called *Pythia*, has been designed and implemented using a rules engine and plug-in architecture to make it flexible and useful in a variety of contexts. We present one such application, reducing blog-comment Spam, as a means of demonstrating *Pythia*'s operation and flexibility.

It is not our goal to present a novel computational model for reputation. Rather, our goal is to provide a framework upon which such computational models can be built and evaluated.

2. REPUTATION PRINCIPLES

We believe there are important principles that apply to reputation systems and should govern their design and operation. Understanding these principles is vital to the success of any computational reputation system. We base these principles on our review of the literature, our own research, and a group discussion from the Berkman Identity Mashup in June 2006. [31]

Reputation is one of the factors upon which trust is based. [22, 16] People often confuse trust and reputation in casual conversation. Trust is "the expectation of one person about the actions of others that affects the first person's choice when an action must be taken before the actions of others are known."[6] Reputation is a basis for trust.

Mui et al. presents a computational model of reputation that explicitly accounts for trust. [19] Mui's model uses reputation and the history of encounters between agents to calculate a value representing the expectation that one agent will reciprocate another agents actions.

Trust is important in many online systems, implicitly or explicitly[30], but is beyond the scope of this paper. Consequently, we are not concerned with what factors beside

reputation go into trust, how trust is built, or how trust is exchanged–we are focused solely on reputation.

The expectation of future reciprocity or retaliation creates an incentive for good behavior in the present. [22] Obviously the party making the reputation request is doing so in an attempt to gain information that will guide some action. However, measuring reputation also affects the behavior of the targets of reputation requests. Axelrod calls this the "shadow of the future." [2]

When present actions will affect future judgments that others make, present behavior will change. For a reputation system to be effective, recorded feedback on past interactions must influence the actions of entities in the future. Put another way, entities must pay some attention to reputations in the system during their decision making process or it will cease to induce desired behaviors.

Reputation is personal. Fundamental to the operation of a reputation system is the notion that agents compute the reputation for other agents based on certain factors. [19] While one can control some of the factors upon which reputation is based, one cannot control how those factors are used by another agent in assessing reputation.

This principle leads to the conclusion that reputation computations should be personalized, so that each agent can compute a different reputation for another based on the information that it has available. In Sabater and Sierra's words, "we cannot talk about the trust/reputation of an individual x, we have to talk about the trust/reputation of an individual x from the point of view of individual y. [24]

Reputation is a currency. While you can't change reputation directly, reputation can be used as a resource. For example, Paul Resnick *et. al.* has shown that a positive eBay reputation has real economic value. [23]

Reputation is narrative. Put another way, reputation varies with time. Reputation is dynamic because the factors that affect it are always changing. One of the problems with many existing reputation models is that reputation is treated as uniform across time. [19]

Reputation is based on identity. Reputation isn't part of an identity, but is *about* an identity. Resnick et al. states that entities in the system *must* be long lived in order to ensure a expectation of future interactions. [22] We distinguish between *identity* and *identifiers*. Any given entity may be known by any number of handles or identifiers.

Linking these identifiers brings together all of the information associated with them for subsequent reputation calculations. This linkage reduces or even prevents identity switching [12] where subjects of bad reputation judgments simply create a new pseudonym to avoid any negative consequences. By using broad-based identity, with linked identifiers, as a foundation for the reputation calculation, the cost of switching is increased.

Reputation is based on verified claims and transactions. Resnick et al. discuss how trust grows in social situations where by interacting with a person over time, a history of past interactions is built that characterizes an individual's character, abilities, and disposition. [22] Reputation systems collect and record feedback concerning interactions. This information is distributed to entities in the future.

We used these terms in the following ways. *Claims* are facts about an identity that can be verified. A claim may be something as simple as an email address or a URL. A claim could also be a fact from some online resource. The number of years that a domain name has been registered, the ratio of inbound to outbound email from an account, or the classification of a URL by a content analysis engine are examples.

Transactions refer to records of interactions between any two entities. Transactions facilitate both the direct and observed interactions that Sabater and Sierra discuss. [24] Most computational reputation systems take direct interactions into account, only a few allow for observed interactions. Note that observed interactions are not the same as opinions. Observed interactions are more reliably objective since they record facts about an interaction, not judgments about it.

Since reputation is personal, how these factors are used in determining reputation is individually determined. Individuals may use various evidence in making claims or proposing a certain rating or endorsement. The penalty for making false claims or giving false endorsements varies from context to context.

Lin et al. assert that trust should be based on verifiable identification of the subject entity, qualification of the entity to perform the requested service, and the subject entity's consistency of performance. [16] Their system, which is built for peer-to-peer reputation management, bases reputation on reports of various performance metrics from a service's peers.

Reputation is based on opinion (indirect information from other witnesses). [24] A reputation isn't just based on facts, but is also based on other's beliefs about the subject of the reputation calculation. These beliefs are signaled to others in various ways depending on the context. Studies of reputation in social networks show the value of including the opinions of other entities. [29]

As we use the term, *opinions* are the indirect information that other entities may supply about the reputation subject. These may be ratings, endorsements, or other subjective judgments about an entity or a past transactions involving an entity. Naturally, a certain amount of uncertainty surrounds this information. Using it involves making more complicated judgments about its veracity. This makes reputation computations using opinions more costly.

Resnick et al. explain some of the challenges reputation systems face–many of those challenges are directly related to the use of opinion. [22] There is a human reluctance to give negative feedback. In order to mitigate losses and failures, individuals turn to negotiation and arbitration in place of giving negative feedback. In some systems, fear of retaliation leads to an acceptance of mediocrity. Finally, disgust and desires to avoid further confrontation may cause individuals to leave bad service in the past and move on without leaving negative feedback.

Reputation exists in the context of community. Any given context will have specific factors for what is important in determining reputation. Community provides a context for evaluating opinions and indirect information.

Sabater et al. discuss this issue under the heading of "visibility types" and classify the reputation of an individual as "global" or "subjective."[24] Global reputation is a single, universal value calculated from all the transactional data available to the system. Subjective reputation is calculated for pairs of individuals based on their interactions and the *opinions* of others.

Global reputation is often insufficient because of a lack of personalization. But Sabater et al. make the point that subjective systems require strong links between individuals so that information is shared frequently and a large body of information upon which to based reputation decisions can be collected.

But the need to collect large amounts of direct interactions between pairs of individuals can be mitigated through the increased use of observed interactions. That way the global set of transactions involving the reputation subject is available to any one individual, greatly increasing the probability that a sufficiently large set of data will be available for a reputation calculation.

Reputation exists in a particular context. When we speak of reputation we say things like "she has a reputation for ..." The fact that someone is a good plumber doesn't mean they will be a reputable babysitter. One of the failings of many current computational models for reputation is that they fail to take context into account or are only useful in a single domain and cannot be repurposed. [19]

There is a natural tradeoff between reputation and privacy. As noted above, reputation is calculated, in part, from a record of past interactions. These interactions record information that individually or in aggregate could threaten the privacy of the subject. People make the trade-off between reputation and privacy in the physical world every day–giving up some privacy to get a credit report, for example. Reputation systems should provide users with information and choice about this tradeoff.

The quality of a reputation calculation should be regularly assessed. As we've stated, reputation calculations are based on a number of factors. However, simply returning a score can lead to false confidence. For example, suppose that reputation is based on a history of past interactions between two parties. If meta data about the transactions is not available, such as number, type, and so on, then a good, or bad, score should not carry as much weight as it otherwise might.

In *A Reputation and Trust Management Broker Framework for Web Applications*, Lin et al. describe a multi-level hierarchy for calculating reputation in a distributed way in peer-to-peer networks. [16] Examining Lin's work leads us to two conclusions:

1. Reputation response should have an accompanying audit trail of external brokers and services used to calculate the reputation.

2. Confidence measures such as the quantity of feedback transactions used during a reputation calculation can be used to evaluate the quality of a reputation framework response.

The reputation framework architecture presented in this paper is designed to be true to the foregoing principles.

3. DESIGN PHILOSOPHY

Beyond the principles discussed in the last section, we also created a design philosophy that guided the system architecture. The design philosophy is intended to be consistent with the principles espoused in the last section and to put stakes in the ground on specific design issues.

***Pythia* supports multiple computational models for reputation.** There has been much research in the area of computational reputation models over the last decade. Sabater et al. gives a review of a dozen such systems. [24] Our goal in building *Pythia* was to create a system that supported many of these models and thus to provide a platform for future research in computational reputation.

***Pythia* pragmatically supports computing reputation from information already available online or information that is easily gathered**. Many of the available reputation systems are designed to work in a closed or specialized environment and thus exploit methods of interaction and observation that are not common interaction idioms on the Internet. [1, 9, 25, 10, 26, 4] In keeping with our goal to build a broad-based, general-purpose framework where reputation calculations can be performed on the emergent behavior of a large number of participants, *Pythia* employs common online identifiers such email addresses and URLs, uses online resources to verify those claims and generate additional facts about them, and captures transactional data automatically from systems through a Web-based API.

***Pythia* supports reputation computations for people**. As mentioned above, our focus has been leveraging emerging wide-area identity systems as a foundation for reputation. This goal influenced *Pythia* design choices in a way that favors reputation calculations for human subjects. The choice of which identifiers to support and how they're verified is an example of such a decision.

In the discussions that follow, we will refer to two groups of entities with respect to *Pythia*. The first is the group we call *users*. Users are the subjects of a reputation request. As we'll see, a reputation request instigates a context-sensitive reputation calculation. *Pythia* aggregates information about users and uses that information in processing reputation requests.

The second group is a subgroup of the users that we call *relying parties*. A relying party is a user making a reputation request about another user. Any user can be a relying party without any additional provisioning in the system.

***Pythia* uses aggregation rather than game-theoretic or stochastic methods to calculate reputation**. Reputation systems typically use either what Sabater and Sierra call "cognitive" or "game-theoretical" conceptual models. [24] Cognitive methods take into account the "mental states that lead [one agent] to trust another agent or assign a reputation." At present, *Pythia* does not use stochastic computation to any significant degree. Rather reputation scores are based upon the "weight of evidence" represented by the aggregated information in the database.

Pythia's use of aggregate data is similar to that of Sierra. Sierra is the reference implementation of OpenPrivacy.org's Reputation Management System. [14] Sierra uses data called opinion objects, which are roughly comparable to the transactions in *Pythia* to calculate reputation. "OpenPrivacy's reputation management system can assemble a set of related opinions into a bias. ... Often, a bias may consist of opinions from multiple nyms."[15] Biases are similar by rule sets (see below) in the *Pythia* system.

Users are identified by multiple identifiers. Multiple identifiers (email addresses, URLs, phone numbers, etc.) can be associated with a single user. The identifiers are validated through mechanisms like email challenges. As discussed above, linking these identifiers is an important method for increasing the body of information available about any given user for a reputation calculation.

Reputation is a personalized function of claims, transactions, and opinions. For our purposes, reputation can be represented as pseudo-mathematical expression:

$$R_u = F_{rp}(I_u, Tx_{u,u'}, O_{u,u'})$$

where

Rp is the relying party's identifier
u is the user id
u' is all users
I_u is a vector of verified identifiers and other claims for u
$Tx_{u,u'}$ is a vector of transactions between u and every other user in the system
$O_{u,u'}$ is a vector of opinions about u

F_{rp} is personalized to each relying party. One of our design goals it to create a system that supports each relying party creating their own F_{rp}. In practice, each relying party can define multiple F_{rp} functions and chose which is used at the time a reputation query is made.

The choice of vectors in the foregoing expression over sets is meant to indicate that these values are ordered according to the time of their collection. F_{rp} can use all of the information in any given vector or can exclude values from vectors by employing filters to select appropriate values for the calculation.

Pythia does not require that reputation be a single number. It's possible that the function could return a vector of values if that was needed. This allows a single reputation query to return reputation "fitness for purpose" parameters for multiple contexts.

The vector I_u corresponds to the notion of *claims* from the last section. Any number and type of identifiers can be used in a reputation calculation. In addition, properties of identifiers, where available, can also be used. For example, a domain name that has been in use for a longer period of time may indicate a more trustworthy Web site since there the owner has a large stake in its future viability. An email address with a ratio of inbound to outbound email messages significantly less than 1 might indicate an address that is used to send Spam.

Transactions and opinions we discussed in the last section. Since opinions are subject to manipulation "it is far more complex for trust and reputation models to use [them]. ...[W]itnesses manipulate or hide pieces of information for their own benefit". [24] Because of this complexity the current implementation of *Pythia* does not use opinions in a significant way. Future work will incorporate opinions into the reputation computation.

Transactions are jointly owned by the parties to the transaction. The exact nature of the transactions in the system depends on the particular use to which it's being put. Transactions always contain an identifier for each party to the transactions and a timestamp in addition to the facts pertinent to the transaction as defined by the domain.

Ownership implies that all parties to a transaction can *see* it, but not change it. Other users, not party to the transaction, can use it in a computation, but cannot inspect the details of the transaction.

Transactions are persistent and immutable. Once the system has recorded a transaction, it cannot be deleted by any of the parties to the transaction. A later transaction may record that it was nullified, but cannot remove it.

Transparency. *Pythia* is designed to favor transparency wherever possible. Consequently,

- Users can see any information the system knows about them (e.g. transactions they are party to)
- Users can see any reputation queries about them, what the results were, and how the result was arrived at.
- As noted, users who are *not* party to a transaction cannot see it, but can use it to calculate reputation about any of the parties to the transaction.

This opens the possibility of information leakage and a subsequent, unintended loss of privacy. Transaction details could be "tapped out" through repeated interactions with the system. We believe that this danger is mitigated by the benefits that transparency offers.

Online resources should be consulted whenever possible to garner reputation information. As we've noted, claims about identifiers are validated in various ways. For example, an email address can be validated by sending a unique URL to the address that the user clicks on to validate receipt. URLs can be validated by having the user embed a unique code in the HTML for the page. [18]

In addition, data stores such as the `whois` database and Netcraft[1] can be used to gather information about domains and Web sites respectively. These online sources of information provide facts that can be used as evidence in a reputation computation. Facts provide a means of assessing reputation that is based on information that is vouched for by a third party.

***Pythia* is centralized.** We made this decision primarily for the sake of simplicity. In Lin's distributed framework [16], for example, a reputation broker first attempts to answer a user's reputation requests based on the broker's local reputation database. If a broker lacks sufficient local feedback to make a recommendation it contacts peer brokers for reputation information. If the peer brokers lack sufficient feedback to return a reputation, a broker can contact institutional reputation authorities before resulting to untrusted sources. Consequently, Lin's hierarchy of diminishing trust is complex:

```
user ->
  broker ->
    peer brokers ->
      reputation authority
```

In contrast, *Pythia*'s model is much simpler, having only users and relying parties both interacting with a centralized reputation authority. The *Pythia* server answers reputation queries based only on the feedback that it has collected and publicly available online information. *Pythia*'s collected feedback may be thousands of transactions, ten or twenty transactions, or none at all.

The result of this decision has all the attendant costs of any centralized system including reliability concerns associated with a single point of failure, security and privacy concerns for having a large collection of information all in one place, and the scaling issues surrounding any large system

[1] `http://www.netcraft.com`

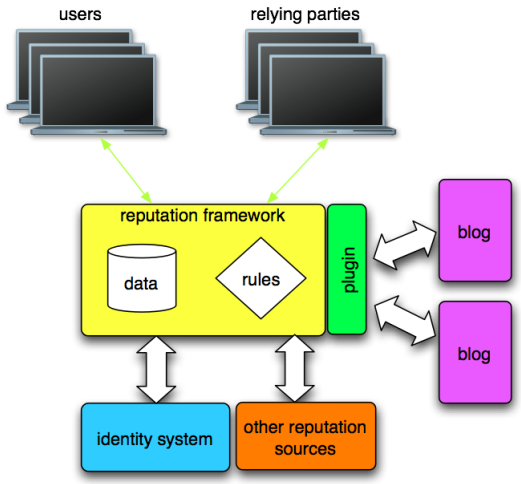

Figure 1: Reputation Framework Architecture

dependent on a database. We anticipate that these concerns can be mitigated in the usual ways.

On the other hand, centralization has some benefits in addition to simplicity including ease of management and performance gains due to the lack of multiple network requests cascading from an single query.

4. ARCHITECTURE

The architecture of *Pythia* is based on the foregoing principles and design philosophy. Consequently,

- *Pythia* is identity system neutral and stores multiple identifiers for any user.
- *Pythia* uses a rules engine to allow reputation calculations to be personalized by each relying party according to their individual needs.
- *Pythia* implements a plug-in architecture to customize the context for reputation computation.

Figure 1 shows the high-level block diagram architecture for *Pythia*.

4.1 Identity Subsystem

Pythia is not an identity system, but is meant to rely on one or more existing identity systems for authentication and user IDs. As implemented, *Pythia* uses OpenID as an authentication mechanism, but other authentication systems could also be used.

The framework allows users to claim and verify identifiers from identity systems of all kinds. At present, these identifiers are limited to email addresses and URLs, but other types of identifiers could be supported without changing the underlying system significantly. Email addresses are verified by sending the email address a challenge message asking the user to click on a URL to claim the email address in *Pythia*.

URLs are verified using MicroIDs [Mic06], a standard for claiming online resources. A MicroID is a secure hash of an email address and a URI. A MicroID is not used to prove someone owns a resource (since anyone with control of the resource could insert the MicroID), but rather to verify claims to a resource.

Pythia generates a MicroID for any claimed URL based on a verified email already in the system for that user. The system instructs the user to embed the MicroID in the page returned by the resource. In this way, the user proves that they have the ability to control the content of the resource identified by the URI.

4.2 Reputation Computation Engine

We support each relying party having the ability to calculate the reputation of users in a personalized manner. That means that reputation calculations are based on information (claims, transactions, and opinions) specific to the user, using an computation that is customizable by the relying party.

These computations are specified in a rule language designed to facilitate reputation calculations in *Pythia*.

Reputation computations are performed using rule-sets, groupings of rules that are executed to carry out the calculation. Relying parties can store as many rule sets in the system as they desire. Each rule in the rule-set comprises an optional filter, a condition, and an action.

Filters operate on transactions and opinions. Some example filters would be *transactions where the customer satisfaction score is greater than* 5 or *transactions where a blog comment was rejected*.

Conditions operate on information in the system as well as other reputation information from the Internet. For example, we have added an interface to the Akismet API,[2] a system for tracking URLs associated with Spam and a content classification engine.

Conditions are simple boolean expressions on information in the system. Permitted boolean expressions are $<$, $>$, $==$, $<=$, $>=$. *Pythia* includes a set of aggregate functions that can be applied to the filtered data. Implemented aggregate functions are `count`, `max`, `min`, `sum`, `average`, and `sd` (standard deviation). An example condition would be, *if the number (count) of transactions is less than* 10... or *if the maximum customer satisfaction value is less than* 3...

Actions modify the reputation score when their associated conditions are satisfied on the filtered data. An action may add, subtract or multiply any fixed amount or an aggregated amount (per above) to the current reputation score. If multiple rule conditions in a rule-set are satisfied, all of the corresponding rule actions will be executed.

Examples of complete rules (combining filters, conditions, and actions) that could be expressed using the rule language include the following:

- *if the average customer satisfaction value* $>$ 2.2 *in transactions where the customer satisfaction value is not null, then increase the reputation score by* 10%.
- *if the user has more than* 2 *verified email addresses in the system, increase the reputation score by* 1.2.
- *if any of the user's claimed URLs can be categorized as "commercial" then decrease the reputation score by* 5.
- *if the number of interactions where this users comment was rejected is* $>$ 3 *then decrease the reputation score by* 2.

[2] `http://akismet.com/development/api/`

One of the dangers of our system is that users might concoct rule sets that simply don't work well or that developing "good" rule-sets might be more difficult than can be managed by casual users of the system. This can be mitigated in two ways.

First, *Pythia* provides a set of system-defined rules can be used as-is or customized by relying parties for their particular purpose.

Second, future systems will include methods for conducting A/B testing of rule sets so that the effect of a rule change can be clearly determined. The use of A/B testing presupposes that the number of reputation queries by the relying party is sufficient to provide statistically meaningful results in the testing.

Users create rule sets using the relying party Web interface, which is available to any user. Since we anticipate that relying parties will not necessarily be programmers, we chose to develop a Web front end for managing rule sets. Building rules using a Web form limits the expressive power of the rule language, but we deem this an acceptable trade-off for ease of use.

4.3 Plug-In Architecture

Our goal was to build a general-purpose reputation framework, but the base system necessarily needs to be specialized to a specific domain for most uses. We determined to develop a plug-in architecture so that this specialization could be accomplished without modifying the code for the base system.

The base system supports user provisioning, the underlying identity system, identifier claims and verifications, building rule-sets for that information, and reputation queries.

To specialize the framework for a particular use, a plug-in specifies what communication and transaction messages are necessary for each integrating system. For example, an online forum might want to categorize transactions reported for posted comments using an enumeration of the status of members (e.g. "veteran") and use this information in calculating reputation. This same transaction attribute would not necessarily be useful in a blog system.

A plug-in is defined in an XML-formatted file that specifies a transaction format. Since plug-ins are stored as files that are loaded by the system at startup, installing, managing and transfering plug-ins to other installations is easy.

The transaction format specifies the type of transactions and their attributes. Transaction attributes are enumerated in the definition. Each attribute definition include a name, a type (text, numeric, boolean) and a regular expression for determining valid values before type conversion.

In addition to the type of transactions for the domain, the plug-in also specifies the API that relying parties will use to record transactions in the system. Transactions are serialized as XML and communicated to the system using the relying party API built into the framework. The API is "RESTful" [11]; in other words, the API transfers plain-old XML (POX) over standard HTTP. Specifics about name-value pairs for the domain are given in the plug-in.

A plug-in can also add to the default set of operators available in the rule language so that domain-specific operators can be added to the language. For example, a plug-in for specifying transactions from an e-commerce site might specify an operator that allows transactions to be easily filtered by region or proximity to a certain zip-code. These operators are specified in Ruby, the language used to build *Pythia*.

Plug-ins are where the default rule sets described in the last section are defined. As mentioned, these rule sets can be cloned by users of the system and subsequently modified.

One limitation of the current system is that the base system can be specialized to only one domain at a time using one plug-in. This keeps the reputation calculations simple. Future versions of *Pythia* will explore how multiple domains interact and what problems this causes.

4.4 Using Pythia

Relying parties interact with *Pythia* in two distinct ways: they make reputation queries for users and they send transactions to the system that record their interactions with the user.

Reputation queries can be made about unauthenticated identifiers. A reputation query includes the identifier of the relying party, an identifier for the subject, and a rule-set identifier. Any identifier stored in the system, email address or URLs, can be the subject of a reputation request. The system uses the rule-set identifier to select a rule set from those previously defined by the relying party and carries out the calculation, using the subject identifier to gather the right data for the rule set to operate on.

Transactions can only be reported about authenticated subjects to avoid users purposely contaminating the transaction store in order to bias future reputation calculations. Any transaction feedback is done in an authenticated manner that records identifiers for the relying party and the user involved in the transaction.

5. IMPLEMENTATION

The *Pythia* framework is implemented in Ruby on Rails. The system implements Web interfaces for users, relying parties, and administrators. The user interface allows users to claim and validate identifiers, view transactions to which they were a party, and view any reputation requests that were made about them.

As previously mentioned, authentication for *Pythia* is provided using OpenID. The Ruby libraries for OpenID are from OpenID Enabled[3].

The underlying rules engine used by *Pythia* is Rools. [4] Rule sets can be defined by plug-ins or through the Web interface by relying parties.

6. AN EXAMPLE APPLICATION

As part of the development and testing, we created a plug-in that allows *Pythia* to be used in blog comment moderation activities. Using this system, a relying party's blog system can query the reputation of people leaving comments and use that score in determine whether to moderate or junk a comment.

In addition any actions taken by the relying party with respect to the comment (e.g. deleting it, approving it, etc.) are automatically sent to the system and recorded as transactions. For purposes of the demonstration, we chose the MovableType blogging platform, but any blogging system could be used.

[3] `http://openidenabled.com`

[4] `http://rools.rubyforge.com`

Customizing the general purpose framework for comment moderation required that we create a plug-in for the reputation framework. In addition, we built a module for MovableType that automated the relaying of actions taken by the blog owner as transactions to *Pythia* and making reputation queries about commenters.

The plug-in defines its own schema for legal transaction types and transaction values specific to blog comments. The framework uses the plug-in's schema definition to receive and store blog comment transactions.

Figure 2 shows the configuration screen in MovableType for the reputation-based comment moderation system. The MovableType plug-in we built to integrate with *Pythia* is called RepKept.

The blog owner, acting as the relying party, specifies the URL of the reputation server and gives authentication information for the reputation server. The relying party also selects which rule set they want to use to calculate the reputation score and gives thresholds for various actions. The rule-sets in the drop-down box are automatically populated from *Pythia* over the relying party API and represent rule sets the relying party has created.

In our test, the blogging software asks users to authenticate before they comment. Once the commenter has authenticated, the blog uses that identifier for requesting a reputation score and later submitting feedback. In our test, blogs use OpenID as the authentication mechanism, but theoretically, any identifier that was associated with the user in *Pythia* could be used.

Once a user submits a comment, a reputation request is made and a reputation value is computed according to the rule set the relying party has selected in the blog configuration. Depending on the value returned and the threshold values set by the relying party, the blog either automatically publishes the comment, holds it for moderation, or deletes it. Each of these generates feedback events.

Feedback events contain the digital identifier of the relying party sending the feedback event and the authenticated digital identifier of the subject of the feedback event. The payload of a feedback event is simply a list of transaction records each represented by a key-value pair. The key is the transaction type and the value is a valid transaction value as defined by the plug-in schema.

In the blog comment case a transaction type could be the `Commenter's IP Address` and the corresponding transaction value could be `192.168.0.1`. A transaction can also be represented as an enumerated value. In this case the transaction type contains the enumerated value and the transaction value is empty.

The transactions in the blog comment example are predominately enumerated values. These values represent actions taken by the relying party given the reputation of the subject. The blog comment example has four types of feedback. In this particular example the feedback transaction types are mutually exclusive. However this need not be the case in general. The transaction types are generated according to the following scenarios:

- When the reputation for a subject exceeds the auto-publish threshold of the relying party the comment immediately published and a `AutoPublished` feedback event is sent to the reputation server.
- In the case where a subject's reputation falls below the relying party's auto-rejection threshold the comment is immediately deleted and a `AutoDeleted` feedback event is sent.
- If a subject's reputation lands between the auto-reject and auto-publish thresholds the comment is placed in a queue for manual moderation. When a human takes action on a comment in the manual moderation queue either a `ManualPublish` or a `ManualDelete` feedback event is sent to the reputation server.

7. FUTURE WORK

The framework we have described is still under development. While most of the key ideas are represented in the system, there is much to be done to complete our vision for them.

One of the most important additions to *Pythia* would be to expand the set of actions and conditional operators available in the rule language. Having more expressive operations would allow various computational models to be more easily implemented. Many of the extant models use sophisticated statistical functions well beyond the `average` and `sd` (standard deviation) operators now available.

The addition of built-in support for A/B testing would allow rules to be more easily evaluated for efficacy. Such A/B testing should be fine-grained so that effects of individual rules can be reported on.

We're confident that the current framework can be used in any single domain with the addition of the appropriate plug-in to customize the type of transactions stored in the system. However, the use of information from multiple contexts (e.g. blog comments and online purchases) in a single system and the interaction of the transaction and opinion data in that system promises to be a fruitful are of study.

Right now, *Pythia* makes use of online data sources such as the `whois` database or Akismet by hard-coding support for the respective API into the framework. It should be possible to create plug-in style architecture for adding these online resources so that many more of then can be made available to *Pythia* applications.

The system could calculate audit data about the transactions and other information in the system and make that available to the rule language. In so doing, relying parties would be able top create rule sets that better take into account the quality of the data before proceeding with the calculation.

Dellarocas discusses the kind of gaming and unfair bias that can be present in online reputation systems. [7] While the issues surrounding security and data contamination were something we discussed consistently throughout the design of *Pythia*, we have not conducted a formal security audit of the system in order to elicit as many of these scenarios as possible and respond to them.

One part of our design that has yet to be implemented is allowing users to express opinions, ratings, and endorsements about other users. For this to be useful some way of recording interuser trust is necessary to measure the confidence that should be placed in a particular opinion. Otherwise, there's no way for the system to know whose ratings to use and whose to ignore.

Interuser trust can be defined explicitly by users, but might also be calculated or otherwise inferred. One way to infer interuser trust is by tapping into social cues that

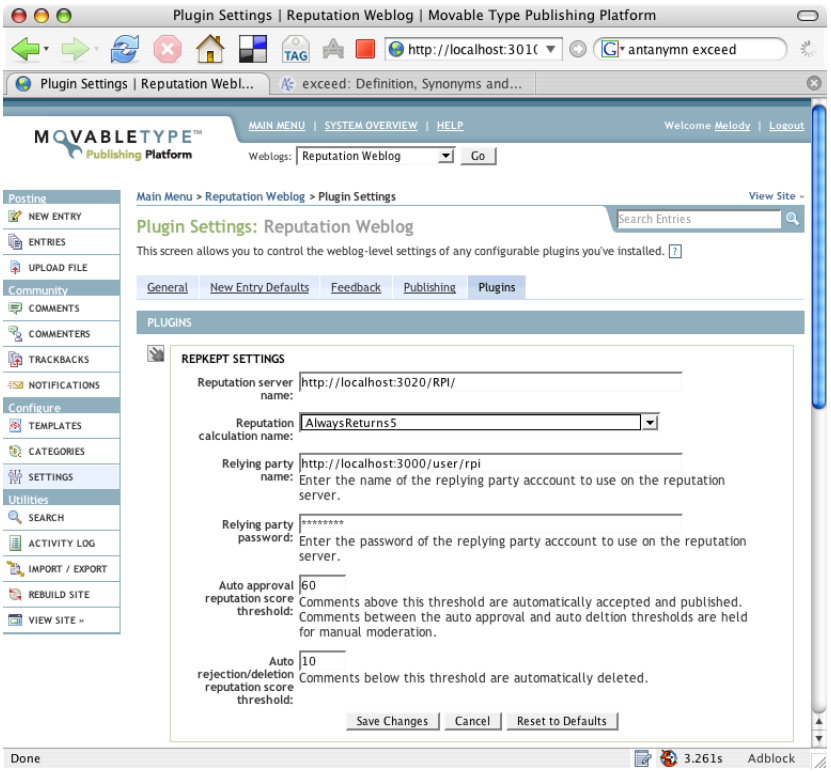

Figure 2: RepKept Configuration Screen

are already on the Web. For example, allowing users to claim OPML files from a site like Bloglines would give an indication of the URLs of other users whose blogs they read. FOAF (Friend of a Friend) data or other data from social networking sites could also be used.

Another avenue for future research is using the system as an identifier exchange service. The system already validates claims to identifiers and associates those identifiers together. *Pythia* could authoritatively state that a particular URL is controlled by the same person who controls a particular email address, for example. Identifier exchange services are useful in aggregating Web services where a client supplies one kind of identifier, but the server requires a different kind.

8. CONCLUSIONS

The emergence of identity systems such as CardSpace, OpenID, LID, SXIP and i-names that are independent of any particular site, provides an opportunity to build reputation systems that are based on those identifiers and have broad application. This paper has described such a system that also employs a rules engine and plug-in architecture to make it malleable to a variety of tasks.

The flexibility of our framework makes it a good platform for further studies in reputation since the effect of various policies in various settings can be more easily explored than when the entire system must be rebuilt from scratch to change the computation model.

9. ACKNOWLEDGMENTS

This work grew out of a class project at Brigham Young University in the Winter of 2006. Class participants were Devlin Daley, Jared Laney, Harshwardhan Nagaonkar, Roger Pack, Sterling Porter, Jason Read, Michael Reynolds, Nathan Stocks, Kevin Tew, and Terry Wilcox.

10. REFERENCES

[1] Amazon auctions. `http://auctions.amazon.com`, 2006.

[2] R. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, 1984.

[3] D. B. Bromley. *Reputation, Image and Impression Management*. John Wiley & Sons, 1993.

[4] J. Carter, E. Bitting, and A. A. Ghorbani. Reputation formalization within information sharing multiagent architectures. *Computational Intelligence*, 2(5):45–64, 2002.

[5] D. Chappel. Understanding Windows CardSpace. `http://msdn2.microsoft.com/en-gb/library/aa480189.aspx`, April 2006.

[6] P. Dasgupta. *Trust: Making and Breaking Cooperative Relations*, chapter Trust As a Commodity, pages 49–72. Department of Sociology, University of Oxford, 2000.

[7] C. Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the* 2^{nd} *ACM Conference on Electronic Commerce*. ACM, 2000.

[8] C. N. Dellarocas. The digitization of word-of-mouth: Promise and challenges of online feedback mechanisms. *SSRN eLibrary*, 2003.

[9] eBay. `http://www.ebay.com`, 2006.

[10] B. Esfandiari and S. Chandrasekharan. On how agents

make friends: mechanisms for trust acquisition. In *Proceedings of the Fourth Workshop on Deception, Fraud and Trust in Agent Societies 2001*, pages 27–34, 2001.

[11] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[12] E. Friedman and P. Resnick. The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10(2):173–199, 2001.

[13] D. Hume. *A Treatise on Human Nature*. Penguine Classics (1975), 1739-1740.

[14] F. Labalme and K. Burton. OpenPrivacy.org. `http://www.openprivacy.org/`.

[15] F. Labalme and K. Burton. Enhancing the internet with reputations. Technical Report 0.7, OpenPrivacy.org, March 2001.

[16] K.-J. Lin, H. Lu, T. Yu, and C.-e. Tai. A reputation and trust management broker framework for web applications. In *International Conference on e-Technology, e-Commerce, and e-Services*, pages 262–269. IEEE, April 2005.

[17] P. R. Milgrom and J. Roberts. Predation, reputation, and entry deterrence. *Journal of Economic Theory*, 27:280–312, 1982.

[18] J. Miller. MicroID - small, decentralized, and verifiable identity. `http://microid.org/`.

[19] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation. In *Proceedings of the 35th Hawaii International Conference on System Sciences*. IEEE, IEEE, 2002.

[20] L. Mui, M. Mohtashemi, and A. Halberstadt. Notions of reputation in multi-agent systems: A review. In *Proceedings of the First International Conference on Autonomous Agents and MAS*, pages 280–287, Bologna, Italy, July 2002. ACM.

[21] OpenID specification. `http://openid.net/specs.bml`, 2006.

[22] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. *Communications of the ACM*, 43(12):45–48, December 2000.

[23] P. Resnick, R. Zeckhauser, J. Swanson, and K. Lockwood. The value of reputation on eBay: A controlled experiment. *Experimental Economics*, 9(2):79–101, June 2006.

[24] J. Sabater and C. Sierra. Review on computational trust and reputation models. *Artificial Intelligence Review*, 24(1):33–60, September 2005.

[25] M. Schillo, P. Funk, and M. Rovatsos. Using trust for detecting deceitful agents in artificial societies. *In Applied Artificial Intelligence, Special Issue on Trust, Deception and Fraud in Agent Societies*, 14(8):825–848, September 2000.

[26] S. Sen and N. Sajja. Robustness of reputation-based trust: boolean case. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 288–293, New York, NY, USA, 2002. ACM Press.

[27] SXIP 2.0 protocol specification. `http://sxip.net/index.php/Specifications_and_Documents`, 2006.

[28] R. L. Trivers. The evolution of reciprocal altruism. *Quarterly Review of Biology*, 46:35, 1971.

[29] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[30] P. J. Windley. *Digital Identity*. O'Reilly Media, 2004.

[31] P. J. Windley. Principles of reputation. `http://www.windley.com/archives/2006/06/principles_of_r`, June 2006.

[32] XRI and XDI explained. `http://www.xdi.org/xri-and-xdi-explained.html`, 2006.