VTrust: A Trust Management System Based on a Vector Model of Trust

Indrajit Ray Sudip Chakraborty Indrakshi Ray

Colorado State University
Computer Science Department
Fort Collins, CO 80523, USA
{indrajit, sudip, iray}@cs.colostate.edu

Abstract. Trust can be used to measure our confidence that a secure system behaves as expected. We had previously proposed a vector model of trust [1]. In this work we address the problem of trust management using the vector model. We develop a new trust management engine which we call VTrust (from Vector Trust). The trust management engine stores and manages current as well as historical information about different parameters that define a trust relation between a truster and a trustee. We propose an SQL like language called TrustQL to interact with the trust management engine. TrustQL consists of a Trust Definition Language (TDL) that is used to define a trust relationship and a Trust Manipulation Language (TML) that is used to query and update information about trust relationships.

1 Introduction

Traditionally, security challenges in computing have been addressed through the use of techniques such as passwords, access control, program verification, intrusion detection, cryptographic protocols and so on. However, with the growing use of distributed open systems such as the Internet and pervasive computing environments, traditional approaches to security are often found lacking. For example, open systems applications such as e-commerce or e-government presents interesting problems to security. In such systems, human users and computational agents and services often interact with each other without having sufficient assurances about the behavior of the other party. There is often insufficient information for deciding how much access to authorize and how much information to share in a multi-user environment. These problems have led researchers to explore the potential of security mechanisms that are based on some aspects of social control.

The notion of *trust* has often played a crucial role for the proper formulation of security policies and in reasoning about expectations from agents and systems to work with confidentiality, integrity and availability. Thus, using trust to enable secure interaction among computational agents seems appropriate. Unfortunately there is no well accepted model for the specification of and reasoning about trust. For the most part, trust is considered to be a binary entity; confidence is measured in terms of either total trust or no trust. This had motivated us earlier to propose a new vector model of trust [1]. In this model trust is a quantitatively measurable entity which can have different

degrees. We define methods and algorithms to measure trust and to compare two trust relationships. To use this trust model however, we need a corresponding trust management system. The current work presents a step in that direction.

A trust management system is a comprehensive framework designed to facilitate the specification, analysis and management of trust relationships. It focuses on specifying and interpreting security policies, credentials, and relationships [2]. The trust management system also provides trust establishment, trust evaluation, trust monitoring and trust analysis service. Traditionally, trust managment has always focused on how one can make authorization and access control more efficiently [3]. Blaze, et al. first introduced the trust management problem as a distinct and important component of security in network services [4]. The *PolicyMaker* trust management system [4, 5] is a framework for expressing in a common language authorization policies, certificates and trust relationships. The *PolicyMaker* service appears to applications very much like a database query engine [4]. *KeyNote* [2] derives from PolicyMaker and was designed to improve some of its weaknesses. *KeyNote* provides a simple language for describing and implementing security policies, trust relationships, and digitally-signed credentials. KeyNote has a built-in credential verification system and a simple notation to express authorization predicates. Grandison [3] proposes *SULTAN* (Simple Universal Logic-oriented Trust Analysis Notation), an abstract, logic-oriented framework designed to facilitate the specification, analysis and management of trust relationships. The IBM Trust Management System [6] implements trust management on top of the Role Based Access Control model. The underlying trust model is binary. The XML based Trust Policy Language that is used to interact with the system is flexible and can be easily expanded. However, it often requires defining custom XML tags for complex situations.

In this paper, we present the VTrust (from Vector Trust model) trust management framework. Major components of the trust management system include a database engine to store and manage trust data, a trust specification engine for defining and managing trust relationships, a trust analysis engine to process results of a trust query, a trust evaluation engine for evaluating trust expressions and a trust monitor for updating trust relationship information in the database engine. We have also developed an SQL like language to interact with the trust management system. We call it TrustQL.

The rest of the paper is organized as follows. In section 2 we describe the main components of the vector trust model. We discuss the extensions to the model that are needed to implement it in the VTrust framework. In section 3 we describe the VTrust system architecture. Section 4 presents the conceptual entity-relationship model for the VTrust system. The section begins with a description of relational entities involved. We discuss inter-relationship of these entities. We then showcase the idea with a running example. Section 5 identifies components of a query based language, called TrustQL, to interact with the trust management system. In this section we also discuss the rationality for choosing such a language. We conclude the paper in Section 6 with some discussion on extensions that we are currently working on.

2 Overview of vector trust model

For the purpose of implementing the model in the trust management system we needed to introduce several modifications to the original model. In this discussion we include the extensions we have made. The interested reader is refered to [1] for the original model. We bgin by defining trust along the lines of Grandison and Sloman [7].

Definition 1. *Trust is defined to be the firm belief in the competence of an entity to act dependably and securely within a specific context.*

Definition 2. *Distrust is defined as the firm belief in the incompetence of an entity to act dependably and securely within a specified context.*

Although we define trust and distrust separately in our model, we allow the possibility of a neutral position where there is neither trust nor distrust.

We specify trust in the form of a trust relationship between two entities – the truster – an entity that trusts the target entity – and the trustee – the target entity that is trusted. This trust is always related to a particular context. An entity A needs not trust another entity B completely. A only needs to calculate the trust associated with B in some context pertinent to a situation. The specific context will depend on the nature of application and can be defined accordingly. Based on our current model, trust is evaluated under one context c only. The simple trust relationship $(A \xrightarrow{c} B)_t$ is a vector with three components – *experience*, *knowledge*, and *recommendation*. It is represented by $(A \xrightarrow{c} B)_t = [{}_AE_B^c, {}_AK_B^c, {}_\Psi R_B^c]$, where ${}_AE_B^c$ represents the magnitude of A's experience about B in context c, ${}_AK_B^c$ represents A's knowledge and ${}_\Psi R_B^c$ represents the cumulative effect of all B's recommendations to A from different sources.

To compute a trust relationship we assume that each of these three factors is expressed in terms of a numeric value in the range $[-1,1] \cup \{\perp\}$. A negative value for the component is used to indicate the *trust-negative* type for the component, whereas a positive value for the component is used to indicate the *trust-positive* type of the component. A 0 (zero) value for the component indicates *trust-neutral*. To indicate a lack of value due to insufficient information for any component we use the special symbol $\perp$.

2.1 Computing the experience component

We model experience in terms of the number of events encountered by a truster A regarding a trustee B in the context c within a specified period of time $[t_0, t_n]$. An event can be trust-positive, trust-negative or, trust-neutral depending on whether it contributes towards a trust-positive experience, a trust-negative experience or, a trust-neutral experience. Intuitively, events far back in time does not count as strongly as very recent events for computing trust values. Hence we introduce the concept of *experience policy* which specifies a length of time interval subdivided into non-overlapping intervals. It is defined as follows.

Definition 3. *An* experience policy *specifies a totally ordered set of non-overlapping time intervals together with a set of non-negative weights corresponding to each element in the set of time intervals.*

Recent intervals in the experience policy are given more weight than those far back. The whole time period $[t_0, t_n]$ is divided in such intervals and the truster A keeps a log of events occurring in these intervals.

If e_k^i denote the k^{th} event in the i^{th} interval, then $v_k^i = +1$, if $e_k^i \in \mathcal{P}$, $v_k^i = -1$, if $e_k^i \in \mathcal{Q}$ or $v_k^i = 0$, if $e_k^i \in \mathcal{N}$, where, $\mathcal{P}$ = set of all trust-positive events, $\mathcal{Q}$ = set of all trust-negative events and $\mathcal{N}$ = set of all trust-neutral events.

The *incidents* I_j, corresponding to the j^{th} time interval is the sum of the values of all the events, trust-positive, trust-negative, or neutral for the time interval. If n_j is the number of events that occured in the j^{th} time interval, then $I_j = \perp$, if there is no event in $[t_{j-1}, t_j]$, and $I_j = \sum_{k=1}^{n_j} v_k^j$, otherwise.

The *experience* of A with regards to B for a particular context c is given by ${}_A\mathrm{E}_B^c = \frac{\sum_{i=1}^{n} w_i I_i}{\sum_{i=1}^{n} n_i}$. where, w_i is a non-negative weight assigned to i^{th} interval.

2.2 Computing the knowledge component

The knowledge component has two parts - *direct knowledge* and *indirect knowledge (or, reputation)*. The truster A assigns two values to these two parts. Her *knowledge policy* regarding B in context c determines the weights to express relative importance between these two. Sum of the product of values and weights for the parts gives us a value for knowledge.

The *knowledge* of A with regards to B for a particular context c is given by

$${}_AK_B^c = \begin{cases} d, \text{if } r = \perp \\ r, \text{if } d = \perp \\ w_d \cdot d + w_r \cdot r, \text{ if } d \neq \perp, r \neq \perp \\ \perp, \text{if } d = r = \perp \end{cases}$$

where $d, r \in [-1, 1] \cup \{\perp\}$ and $w_d + w_r = 1$. d and r are the values to direct and indirect knowledge respectively and w_d and w_r are the corresponding non-negative weights.

2.3 Computing the recommendation component

Recommendation is evaluated on the basis of a *recommendation value* returned by a recommender to A about B. Truster A uses the "level of trust" he has on the recommender in the context "to provide a recommendation" as a weight to the value returned. This weight multiplied by the former value gives the actual *recommendation score* for trustee B in context c.

The *recommendation* of A with regards to B for a particular context c is given by ${}_\Psi R_B^c = \frac{\sum_{j=1}^{n} (\mathbf{v}(A \xrightarrow{rec} j)_t^N) \cdot V_j}{\sum_{j=1}^{n} (\mathbf{v}(A \xrightarrow{rec} j)_t^N)}$ where Ψ is a group of n recommenders, $\mathbf{v}(A \xrightarrow{rec} j)_t^N)$ = trust-value of j^{th} recommender and V_j = j^{th} recommender's recommendation value about the trustee B.

2.4 Trust vector

We next observe that given the same set of values for the factors that influence trust, two trusters may come up with two different trust values for the same trustee. We believe that there are two main reasons for this. First, during evaluation of a trust value, a truster may assign different weights to the different factors that influence trust. The weights will depend on the trust evaluation policy of the truster. So if two different trusters assign two different sets of weights, then the resulting trust value will be different. The second reason is applicable only when the truster is a human being and is completely subjective in nature – one person may be more trusting than another. We believe that this latter concept is extremely difficult to model. At this stage we choose to disregard this feature in our model and assume that all trusters are trusting to the same extent. We capture the first factor using the concept of a *normalization policy*. The normalization policy is a vector of same dimension as of $(A \xrightarrow{c} B)_t$; the components are weights that are determined by the corresponding trust evaluation policy of the truster and assigned to experience, knowledge, and recommendation components of $(A \xrightarrow{c} B)_t$. The normalization policy together with the experince policy and the knowledge policy form the truster's *trust evaluation policy*.

We use the notation $(A \xrightarrow{c} B)_t^N$, called *normalized* trust relationship to specify a trust relationship. It specifies A's *normalized* trust on B at a given time t for a particular context c. This relationship is obtained from the simple trust relationship – $(A \xrightarrow{c} B)_t$ – after combining the former with the normalizing policy. It is given by $(A \xrightarrow{c} B)_t^N = \mathbf{W} \odot (A \xrightarrow{c} B)_t$. The $\odot$ operator represents the normalization operator. Let $(A \xrightarrow{c} B)_t = [{}_AE_B^c, {}_AK_B^c, {}_\psi R_B^c]$ be a trust vector such that ${}_AE_B^c,\ {}_AK_B^c,\ {}_\psi R_B^c \in [-1,1] \cup \{\perp\}$. Let also $\mathbf{W} = [W_e, W_k, W_r]$ be the corresponding trust policy vector such that $W_e + W_k + W_r = 1$ and $W_e, W_k, W_r \in [0,1]$. The $\odot$ operator generates the normalized trust relationship as

$$\begin{aligned}(A \xrightarrow{c} B)_t^N &= \mathbf{W} \odot (A \xrightarrow{c} B)_t \\ &= [W_e,\ W_k,\ W_r] \odot [{}_AE_B^c,\ {}_AK_B^c,\ {}_\psi R_B^c] \\ &= [W_e \cdot {}_AE_B^c,\ W_k \cdot {}_AE_B^c,\ W_r \cdot {}_\psi R_B^c] \\ &= [{}_A\hat{E}_B^c,\ {}_A\hat{K}_B^c,\ {}_\psi \hat{R}_B^c]\end{aligned}$$

We next introduce a concept called the *value* of a trust relationship. This is denoted by the expression $\mathbf{v}(A \xrightarrow{c} B)_t^N$ and is a number in $[-1,1] \cup \{\perp\}$ that is associated with the normalized trust relationship. The special symbol $\perp$ is used to denote the value when there is not enough information to decide about trust, distrust, or neutrality. This value together with the vector now represents a trust of certain degree.

Finally, we investigate the dynamic nature of trust – how trust (or distrust) changes over time. We make a couple of observations. First, trust depends on trust itself; that is a trust relationship established at some point of time in the past influences the computation of trust at the current time. If an agent is positively trusted to begin with then negative factors are often overlooked (that is given less weightage) when trust is re-evaluated in the agent. Second, trust decays with time. This is owing to the effect of forgetfulness of the human mind. The second idea is captured by the equation –

$\mathbf{v}(T_{t_n}) = \mathbf{v}(T_{t_i})e^{-(\mathbf{v}(T_{t_i})\Delta t)^{2k}}$ where, $\mathbf{v}(T_{t_i})$, be the value of a trust relationship, T_{t_i}, at time t_i and $\mathbf{v}(T_{t_n})$ be the decayed value of the same at time t_n. We have developed a method to obtain a vector of same dimension as of $(A \xrightarrow{c} B)_t^N$ from this value $\mathbf{v}(T_{t_n})$. The effect of time is captured by the parameter k which is determined by the truster A's *dynamic policy* regarding the trustee B in context c. The current normalized vector together with this time-affected vector are combined according to their relative importance. Relative importance is determined by truster's *history_weight policy* which specifies two values α and β in $[0,1]$ (where, $\alpha+\beta=1$) as weights to current vector and the vector obtained from previous trust value. The new vector thus obtained gives the actual normalized trust vector at time t for the trust relationship between truster A and trustee B in context c. This is represented by the following equation

$$(A \xrightarrow{c} B)_{t_n}^N = \begin{cases} [{}_A\hat{E}_B^c, {}_A\hat{K}_B^c, {}_\psi\hat{R}_B^c] & \text{if } t_n = 0 \\ [\frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}] & \text{if } t_n \neq 0 \text{ and } {}_A\hat{E}_B^c = {}_A\hat{K}_B^c = {}_\psi\hat{R}_B^c = \perp \\ \alpha \cdot [{}_A\hat{E}_B^c, {}_A\hat{K}_B^c, {}_\psi\hat{R}_B^c] + \beta \cdot [\frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}] & \\ \qquad \text{if } t_n \neq 0 \text{ and at least one of } {}_A\hat{E}_B^c, {}_A\hat{K}_B^c, {}_\psi\hat{R}_B^c \neq \perp \end{cases}$$

where $[\frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}]$ is the time-effected vector and $\mathbf{v}(\hat{T}) = \mathbf{v}(T_{t_n})$.

3 The VTrust system architecture

The high level system architecture consists of the components as shown in the following figure 1. Values of the different parameters needed for the computation of trust relationships are maintained in the VTrust database. The truster interacts with the trust management system through the external interface. The communication is done using the language TrustQL that we have developed. The TrustQL language parser in the interface parses the command and sends it to the appropriate component in the next layer. This layer has the following major components. A *specification server* is managing and updating the trust database schema. The *analysis engine* processes all trust related queries. It interacts with specification server and an *evaluation engine*. The latter is responsible for computing trust related information according to the underlying model. The evaluation engine takes a parsed trust query string, finds the associated information and policy, and returns the final trust vector and value to the analysis engine. The *trust monitor* is responsible for acquiring relevant trust formulation parameters. It maintains the VTrust database, updates the trust data while truster and trustee interacts with each other and also updates periodically trust component values like experience and knowledge.

All these information (trusters, trustees, recommenders, policies and trust parameter information) are stored in the VTrust database. The database is implemented as described in section 4. Since TrustQL can not interact with the database directly, an SQL translator beneath the component layer does this job. The specification server, analysis engine and evaluation engine takes a trust operation specified in TrustQL and maps the command to an equivalent SQL command to interact with the underlying database. After receiving an answer from the database, each of those components again does a reverse mapping to output the answer in terms of TrustQL.

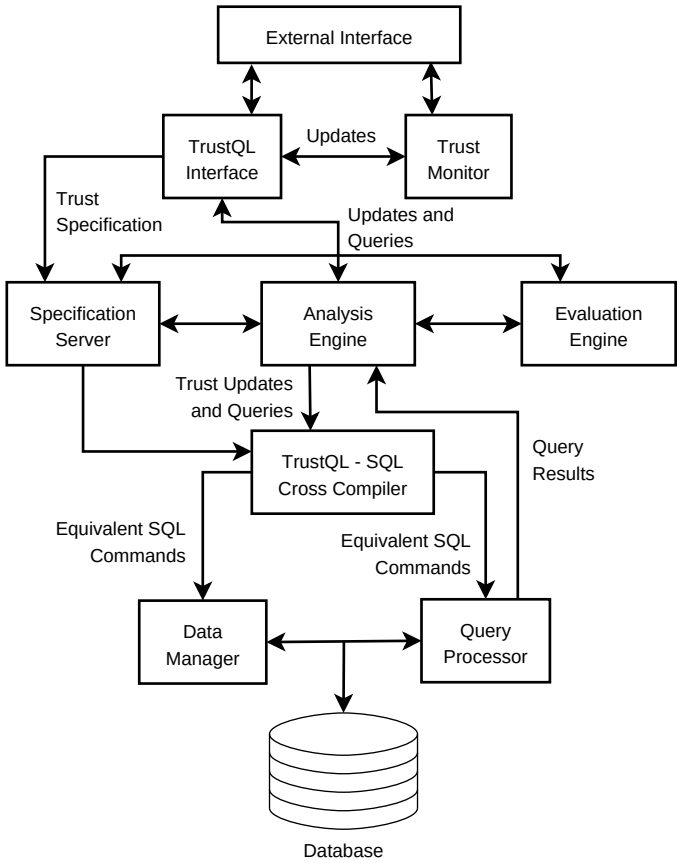

Fig. 1. Trust Management System Architecture

The following algorithm is used by a truster to compute the trust relationship with a trustee for a given context at any given time.

Algorithm 1 *1. If not already available, initialize the truster's trust evaluation policy corresponding to the trustee and the specific context. If needed update the same to reflect current circumstances.*

2. *Initialize dynamic policy and history_weight policy if not already available. Update as needed.*
3. *Compute truster's experience with trustee.*
   (a) *Determine last point in time when trust was evaluated for current trustee in the given context. If such a time exists call it* t_{last}.
   (b) *Read off experience values from database starting from most recent first till either* t_{last} *or start of experience table.*
   (c) *Apply experience policy to evaluate current experience value.*
4. *Compute truster's knowledge with trustee by applying knowledge policy to current direct knowledge and reputation values.*

5. *Compute recommendation value for trustee.*
6. *Compute truster's simple trust on trustee using values obtained in steps 3 - 5. Apply normalization policy as appropriate to the simple trust.*
7. *Let trust value at t_{last} be termed T_{last} (assuming available); compute decayed value for T_{last} by applying dynamic policy to it.*
8. *Combine trust values obtained in steps 6 and 7 using the history_weight policy to get the truster's current trust relationship with the trustee in the given context.*
9. *Record current time of trust evaluation as t_{last} corresponding to this truster, trustee and context.*

4 Conceptual trust model

We model the underlying trust components using Entity-Relationship techniques (see Figure 2). Both the entity sets and the relationship sets are then converted to tables in a relational database with columns representing the "attributes" of the entity and relationship sets. ACTOR is a generalization of three specific types – TRUSTER, TRUSTEE, and RECOMMENDER as *Role*. A TRUSTER has the the following relationships with a TRUSTEE: EVENTS, EXPPOL, KNOWLEDGE, KNOWLPOL, NORMPOL, DYNPOL, and HWTPOL. The relationship RECOMMENDATION involves all three types of ACTOR (i.e Truster, Trustee and Recommender). The TRUSTER calculates his 'experience' with a TRUSTEE on the basis of EVENTS and EXPPOL. The entity EVENTS is a log of events happened between the truster and the trustee in the context at certain time. Experience is calculated by summing up the net effect of events within some consecutive intervals of time. The EXPPOL specifies the length of that time interval. KNOWLEDGE returns a value which is evaluated based on KNOWLPOL which determines weights for direct knowledge (*DKnolWt*) and reputation (*RepuWt*). The truster assigns values for direct knowledge (*DirectKnol*) and repuation (*Repuation*). RECOMMENDATION (*RecommendationScore*) is evaluated based on the value returned by the recommender (*RecoValue*) and the recommender's weight (*RecommenderWt*) according to the truster. These three values (i.e., experience score, knowledge score and recommendation score) are normalized according to a normalization policy (NORMPOL). They are multiplied with their corresponding weights – *ExpWt*, *KnolWt*, and *RecoWt*. The DYNPOL determines the parameter *k* to get the current value of the last available trust value. A vector from this trust history is derived and HWTPOL specifies weights to be assigned to this vector and the current normalized vector. Composition of these two vectors results in actual trust vector with components Experience score, Knowledge score and Recommendation score and they, in turn, return trust value between the truster and the trustee in a context on a particular date.

We now use a hypothetical trust relationship example to descibe how the VTrust database works. Let Alice be developing a software that has several modules with diffrent functionality. She wants to get every module tested by an expert software engineer before she merges two modules. Assume that she assigns this testing responsibility to Bob. Thus, Alice wants to evaluate her 'trust' on Bob in the context of 'efficiency to test a software' (say, EST; acronym for the context) to decide her further course of action with Bob in the context EST. Alice sets up a trust-relationship with Bob in the

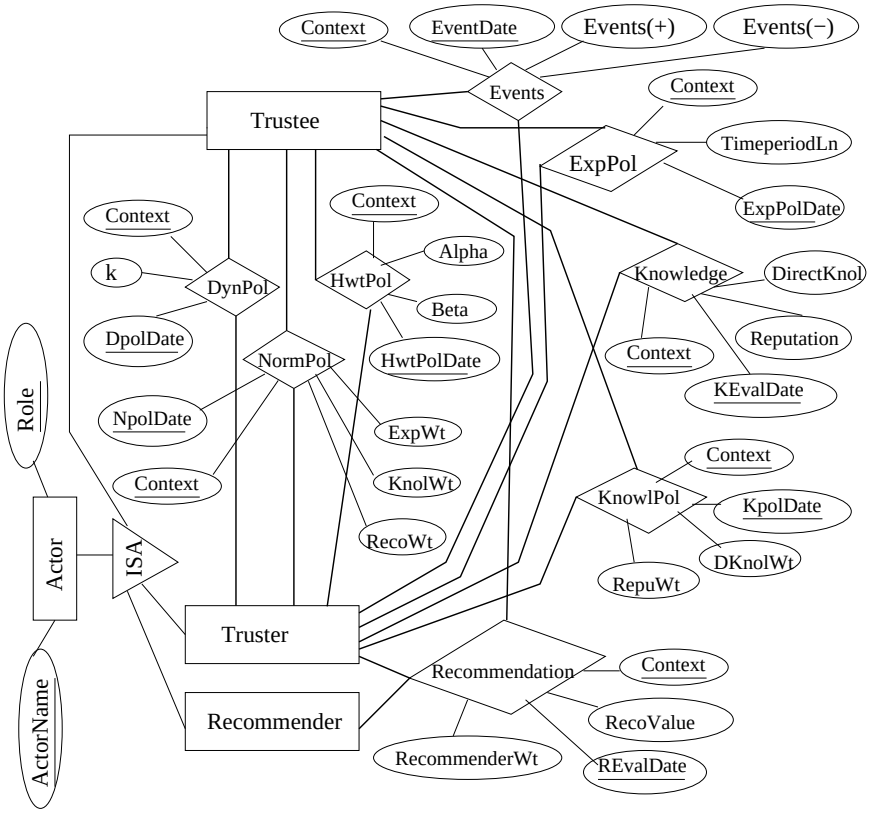

Fig. 2. ER-diagram of the VTrust system

context EST. She thinks of consulting Charlie, her friend who happens to know Bob, to get his view about Bob's efficiency in this context. To store the information Alice creates the table called ACTOR as shown in Table 1.

ACTORNAME	ROLE
Alice	Truster
Bob	Trustee
Charlie	Recommender

Table 1. Initial ACTOR table

Alice starts interacting with Bob from 1st January, 2004. She decides to keep track of events that occured between her and Bob on a monthly basis. Alice forms her EXPPOL as shown in Table 2. Alice also sets up her knowledge policy regarding Bob. She decides to assign 70% weight on direct knowledge and 30% to indirect knowledge she gets about Bob regarding EST. Thus, her KNOWLPOL table looks like that in Table 3. Alice can set her knowledge policy anytime before the first time she evaluates trust for Bob in EST. She can also set the normalization policy anytime prior to first evaluation

ACTOR.ACTOR-NAME1	ACTOR.ROLE1	ACTOR.ACTOR-NAME2	ACTOR.ROLE2	CONTEXT	EXOPOLDATE	Timeperiod Ln
Alice	Truster	Bob	Trustee	EST	01/01/2004	1 month

Table 2. Alice's experience policy

ACTOR.ACTOR-NAME1	ACTOR.ROLE1	ACTOR.ACTOR-NAME2	ACTOR.ROLE2	CON-TEXT	KPOLDATE	Dknol Wt	Repu-Wt
Alice	Truster	Bob	Trustee	EST	06/30/2004	0.7	0.3

Table 3. Alice's knowledge policy

ACTOR.ACTOR-NAME1	ACTOR.ROLE1	ACTOR.ACTOR-NAME2	Actor.Role2	CONTEXT	NPOL DATE	Exp Wt	Knol Wt	Reco Wt
Alice	Truster	Bob	Trustee	EST	10/31/2004	0.5	0.3	0.2

Table 4. Alice's normalization policy

of trust for Bob in EST. Let she have the NORMPOL as shown in Table 4. Now let us assume that Alice evaluates Bob's trust in EST for the first time on 31st December, 2004. On that day her EVENTS table looks like Table 5. Alice calculates the 'experience value' from the above log with the help of her 'experience policy'. The policy defines the time-period length as 1 month. Let us assume that Alice specify the start-date for calculation as 01/02/2004. Then the above log is devided in 30-day periods and net effect of positive and negative events are calculated within those periods. Thus Alice got her experience value as 0.1543. Alice next builds the KNOWLEDGE, and RECOMMENDATION databases. She assigns two values for direct knowledge and reputation for Bob in EST. During the year Alice possibly makes several visits to Bob's office to get idea about Bob's infrastructure; she checks tools and techniques used by Bob for testing. She hears about Bob's efficiencey in the job. Based on these information Alice assigns those two values according to her own policy. The knowledge value, which comes as 0.62, is calculated based on the two values she provides and their corresponding weights specified in Table 3. Finally, before evaluating trust in Bob Alice consults Charlie to get his view on Bob in the context of EST. Charlie returns his judgment about Bob as a recommendation value to Alice. Alice evaluates Charlie's recommendation on the basis of the trust she has on Charlie. Alice calculates recommendation score of Bob with these information as 0.44 and the RECOMMENDATION table is of the form of Table 7. Now Alice evaluates the actual trust vector as well as the trust value based on these information. All these component values are normalized before calculating the trust value with the values available from Table 4. These calculations are automatically computed by the system. The final trust vector and the trust value of Alice for Bob in the context EST as obtained on 31st December, 2004 shown in the Table 8.

Let us next assume that Alice again wants to evaluate Bob after 4 months. Therefore, on 30th April, 2005 she wants to have a trust for Bob in the same context EST. We assume that after evaluating trust on 31st December, she purges all events prior to that date and start keeping log afresh. Rationale is that at any later time, her decision

ACTOR.ACTOR-NAME1	ACTOR.ROLE1	ACTOR.ACTOR-NAME2	Actor.Role2	CONTEXT	EVENT DATE	Events(+)	Events(-)
Alice	Truster	Bob	Trustee	EST	01/06/2004	1	0
Alice	Truster	Bob	Trustee	EST	01/19/2004	1	1
...	...	...	...	...	...	...	...
Alice	Truster	Bob	Trustee	EST	07/27/2004	0	1
Alice	Truster	Bob	Trustee	EST	10/10/2004	0	1
Alice	Truster	Bob	Trustee	EST	11/07/2004	2	0

Table 5. Alice's EVENTS table on 31st December, 2004

ACTOR. ACTOR NAME1	ACTOR.ROLE1	ACTOR. ACTOR NAME2	Actor.Role2	CONTEXT	KEVAL DATE	Direct Knol	Reputation
Alice	Truster	Bob	Trustee	EST	12/31/2004	0.8	0.2

Table 6. Alice's knowledge value on 31st December, 2004

would be influenced by the previous trust value. She does not need the whole set of events to derive current trust value. Only the events after the previous evaluation are considered to evaluate current experience. We also assume that she has not changed any policy and nothing happened between her and Bob during these 4 months. Thus there will not be any change in Alice's EVENTS table on 30th April, 2005. Let us assume that Alice changes the values assigned to direct knowledge and repuation on the basis of her current judgment. So she adds a new entry to the KNOWLEDGE table and calculates the new knowledge value as 0.66 (say). Maybe also the trust relationship of Alice with Charlie on the context of "providing a recommendation" changes from 0.8 to 0.7 and this time Charlie returns a lower value 0.4 for Bob in the context EST. Hence the RECOMMENDATION table changes and (possibly) the new recommendation score for Bob is evaluated as 0.28. Now the trust value evaluated earlier (i.e., on 31st December, 2004) will have some effect on Alice's present decision. For that Alice has to form the dynamic policy which gives the current 'level' of the previous value. Alice can form this table DYNPOL anytime before 30th April, 2005. Let us assume that Alice set k in DYNPOL as 1 on 31st March, 2005. This is presented in Table 9.

To combine the vector having current value of the parameters with the vector derived from the time-affected value of trust, Alice needs to form HWTPOL on or before 30th April, 2005 to put relative weight on these two vectors. Let us assume that Alice put 60% weight to the vector with currently evaluated values and rest 40% to the vector derived from the time-affected value. It is shown in Table 10. The final trust vector and value on 30th April, 2005 is presented in the Table 11. Alice keeps on adding a new entry in the tables everytime she evaluates Bob's trust vector in EST.

5 TrustQL: The trust query language

Users of the trust management system need a language to interact with the system. The language should be able to interact with the database implementation of the model.

ACTOR.ACTOR NAME1	ACTOR.ROLE1	ACTOR.ACTOR NAME2	ACTOR.ROLE2	CONTEXT	REVAL DATE	ACTOR.ACTOR NAME3	ACTOR.ROLE3	Reco value	Recommenderwt
Alice	Truster	Bob	Trustee	EST	12/31/2004	Charlie	Recommender	0.55	0.8

Table 7. Alice's recommendation score on 31st December, 2004

ACTOR.ACTOR NAME1	ACTOR.ROLE1	ACTOR.ACTOR NAME2	ACTOR.ROLE2	CONTEXT	EVALUATION DATE	Experience Score	Knowledge Score	Recommendation Score	Trust value
Alice	Truster	Bob	Trustee	EST	12/31/2004	0.077	0.186	0.088	0.351

Table 8. Alice's trust on Bob in the context EST on 31st December, 2004

Therefore, we introduce a trust language similar to Structured Query Language (SQL). We call this language as Trust Query Language or TrustQL.TrustQL consists of Trust Definition Language (TDL) and Trust Manipulation Language (TML). TDL is used to create, alter and drop entities, policies, parameters and context. TML is used to add, modify and delete trust records as well as query the trust engine to get trust values. Trust Definition Language (TDL) consists of TrustQL keywords, Identifiers, Statements, and TrustQL convention. Trust Manipulation Language (TML) consists of commands like INSERT, UPDATE, DELETE, SELECT, and commands to query trust value after the trust management system has been set up using Trust Definition Language. TrustQL differs from general purpose procedural language such as C and Java in that users specify what they want instead of how to get the result. It is up to the VTrust engine to manipulate the data and present the final trust value to end users. From the user's point of view, this approach makes it easy to interact with the trust management system.

Some examples of TrustQL statements are shown in figures 3 and 4.

```
CREATE POLICY
    {policy_name}
WEIGHT
{(experience_weight, knowledge_weight, recommendation_weight)}
EXPERIENCE POLICY {experience_policy_name}
KNOWLEDGE POLICY {knowledge_policy_name}
RECOMMENDATION POLICY {recommendation_policy_name}
DYNAMICS POLICY {dynamics_policy_name}
HISTORY POLICY {history_policy_name}
```

Fig. 3. Defining trust evaluation policies using TrustQL

ACTOR. ACTOR NAME1	ACTOR. ROLE1	ACTOR. ACTOR NAME2	ACTOR. ROLE2	CONTEXT	DPOLDATE	k
Alice	Truster	Bob	Trustee	EST	03/31/2005	1

Table 9. Alice's dynamic policy

ACTOR. ACTOR NAME1	ACTOR. ROLE1	ACTOR. ACTOR NAME2	ACTOR. ROLE2	CONTEXT	HWTPOL DATE	Alpha	Beta
Alice	Truster	Bob	Trustee	EST	04/01/2005	0.6	0.4

Table 10. Alice's policy on assigning weights to previous trust value at current time

6 Conclusion and future work

The vector model of trust gives a technique to measure trust quantitatively on the basis of some parameters. The model has methods to specify policies to evaluate those parameters. Using this model we can define "multilevel" trust and distrust. In this paper we present a trust management framework, named as VTrust, based on vector-based trust model. In this framework, information regarding trust relationships are kept in a trust database. The trust relationship, the entities involved in it (e.g., truster, trustee, context etc.), the parameters to evaluate trust, and the policies to determine values are represented as relational entities. All these are translated to tables of the database and the attributes of these entities are expressed as columns in the tables. The working principle of the database system is explained with an example. The system architecture of VTrust, which contains a user interface, trust management components in the middle layer and the trust database as the lower layer is also introduced. We also introduce a query language, called TrustQL, to interact with the components of trust management system. We present some of the features of TrustQL with examples. The detail syntax and semantics of TrustQL are left out.

A lot of work remains to be done. We are currently extending the underlying trust model to define more operations on trust relationships. Presently we have single entity as truster or trustee. We want to incorporate the idea of a group of truster or a group of trustee. In our current representation, the user (i.e., the truster) needs to enter a lot of values. We are trying to minimize the number of user input by giving more power to the analysis engine. An efficient design of the underlying database and a good user interface are need to be developed. We believe that achieving above goals would result in a complete trust management framework based on the vector-based trust model.

Acknowledgment

This work was partially supported by the U.S. Air Force Research Laboratory (AFRL) and the Federal Aviation Administration (FAA) under contract F30602-03-1-0101. The views presented here are solely that of the authors and do not necessarily represent those of the AFRL or the FAA. The authors would like to thank Mr. Pete Robinson of the AFRL and Mr. Ernest Lucier of the FAA for their valuable comments and their support for this work.

ACTOR. ACTOR NAME1	ACTOR. ROLE1	ACTOR. ACTOR NAME2	ACTOR. ROLE2	CONTEXT	EVALUATION DATE	Experience Score	Knowledge Score	Recommendation Score	Trust value
Alice	Truster	Bob	Trustee	EST	12/31/2004	0.077	0.186	0.088	0.351
Alice	Truster	Bob	Trustee	EST	04/30/2005	0.0064	0.4024	0.1744	0.5832

Table 11. Alice's trust on Bob in the context EST on 30^{th} April, 2005

```
INSERT TRUST
BETWEEN {<truster>} AND {<trustee>}
CONTEXT {context_name}
[WHEN {some_date}]
[EXPERIENCE VALUES {(<experience_values>)}]
[KNOWLEDGE VALUES {(<knowledge_values>)}]
[RECOMMENDATION VALUES {(<recommendation_values>)}]

<truster> ::= {entity_name}
<trustee> ::= {entity_name}
<experience_values> ::= {time_interval, experience_value} [,...n]
<knowledge_values> ::= {direct_knowledge_value,
                        indirect_knowledge_value }
<recommendation_values> ::= {<recommender>, recommendation_value}[,...n]
<recommender> ::= {entity_name | group_name}
```

Fig. 4. Populating trust relationships using TrustQL

References

[1] Ray, I., Chakraborty, S.: A vector model of trust for developing trustworthy systems. In: Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS'04). Volume 3193 of Lecture Notes In Computer Science., Sophia Antipolis, Frech Riviera, France, Springer-Verlag (2004) 260–275

[2] Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.: The keynote trust management system (version 2). http://www.crypto.com/papers/rfc2704.txt (1999)

[3] Grandison, T.: Trust Specification and Analysis for Internet Applications. PhD thesis, Imperial College of Science Technology and Medicine, Department of Computing, London, UK (2001)

[4] Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In: Proceedings of 17th IEEE Symposium on Security and Privacy, Oakland, California, USA, IEEE Computer Society Press (1996) 164–173

[5] Blaze, M., Feigenbaum, J., , Strauss, M.: Compliance checking in the policymaker trust management system. In: Proceedings of the 2nd Financial Crypto Conference. Volume 1465 of Lecture Notes in Computer Science., Anguilla, Springer-Verlag (1998) 254–274

[6] Herzberg, A., Mass, Y., Mihaeli, J., Naor, O., Ravid, Y.: Access control meets public key infrastructure, or: Assigning roles to strangers. In: Proceedings of IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society Press (2000) 2–15

[7] Grandison, T., Sloman, M.: A survey of trust in internet applications. IEEE Communications Surveys and Tutorials **3** (2000) 2–16