A Model for Trust Metrics Analysis[⋆]

Isaac Agudo and Carmen Fernandez-Gago and Javier Lopez

Department of Computer Science, University of Malaga, 29071, Málaga, Spain
{isaac,mcgago,jlm}@lcc.uma.es

Abstract. Trust is an important factor in any kind of network essential, for example, in the decision-making process. As important as the definition of trust is the way to compute it. In this paper we propose a model for defining trust based on graph theory and show examples of some simple operators and functions that will allow us to compute trust.

1 Introduction

In the recent years trust has become an important factor to be considered in any kind of social or computer network. The concept of trust in Computer Science derives from the concept on sociological or psychological environments. Trust becomes essential when an entity needs to establish how much trust to place on another of the entities in the system.

The definition of trust is not unique. It may vary depending on the context and the purpose where it is going to be used. For the approach adopted in this paper we will define trust as the *level of confidence* that an entity participating in a network system places on another entity of the same system for performing a given task. We mean by a *task* any action that an agent or entity in the system is entitled or in charged of performing.

Trust management systems have been introduced in order to create a coherent framework to deal with trust. The first attempts for developing trust management systems were PolicyMaker [5], KeyNote [4] or REFEREE [7]. Since the importance of building trust models has become vital for the development of some nowadays computer systems the way this trust is derived, i.e., the *metrics*, becomes also crucial. Metrics become very important for the deployment of these trust management systems as the way of quantifying trust. The simplest way to define a trust metric is by using a discrete model where an entity can be either 'trusted' or 'not trusted'. This can also be expressed by using numerical values such as 1 for trusted and 0 for not trusted. The range of discrete categories of trust can be extended with 'medium trust', 'very little trust' or 'a lot of trust', for example. More complex metrics use integer or real numbers, logical formulae like BAN logic [6] or vector like approaches [9]. In the early nineties the first proposals for trust metrics were developed in order to support Public Key Infrastructure (for instance [13]). In the recent years the development of new networks or systems such as P2P or

[⋆] This work has been partially funded by the European Commission through the research project SPIKE (FP7-ICT-2007-1-217098), and the Spanish Ministry of Science and Education through the research project ARES (CONSOLIDER CSD2007-00004).

Ad-Hoc networks, or ubiquitous or mobile computing has led to the imminent growth of the development of trust management systems and consequently metrics for them. Most of the used metrics are based in probabilistic or statistics models (see [10] for a survey on this). Also due to the growth of online communities the use of different metrics has become an issue (see for example the reputation scores that eBay uses [1]). Flow models such as Advogato's reputation system [12] or Appleseed [16, 17] use trust transitiveness. In these type of systems the reputation of a participant increases as a function of incoming flow and decreases as a function of ongoing flow.

There are many different trust models in the literature. The model we present in this paper is a graph-based model that allows us to represent trust paths as matrices. Our intention is to characterize trust metrics that are more suitable to be used in any given case, depending on the nature of the system, its properties, etc. As a novelty we propose the definition of a *trust function* that allows us to do this. A classification of trust metrics has been done in [17] but more oriented to the semantic web environment.

The paper is organized as follows. In Section 2 we outline how trust can be modelled as a graph and give some definitions. These definitions will be meaningful for Section 3 where we introduce our trust evaluation. Those definitions are going to be used for the instantiations of different operators in Section 4. Section 5 concludes the paper and outlines the future work.

2 A Graph based Model of Trust

Trust in a virtual community can be modelled using a graph where the vertices are identified with the entities of the community and the edges correspond to trust relationships between entities. As we mentioned before, trust can be defined as the level of confidence that an entity s places on another entity t for performing a given task in a proper and honest way. The confidence level may vary depending on the task. Assuming that the level of confidence is a real number and that for each task there is only one trust value associated in our reasoning system, the trust graph is a weighted digraph.

Let us consider different tasks in our system. The trust graph will be a labelled multi digraph, i.e. there can be more than one edge from one particular vertex to another, where the label of each edge is compounded of a task identifier and the confidence level associated to it. That graph can also be modelled using a labelled digraph in which the labels consist of a sequence of labels of the previous type, each one corresponding to one edge of the multigraph. In this scenario we can distinguish two cases: (1) The simplest case where only one task is considered and (2) the average case where more than one task is considered.

The average case is quite easy to manage. For a fixed task identifier, we obtain a simple trust graph that can be inspected using techniques for the simplest case. The problem arises when there are dependencies among tasks. This could imply that implicit trust relationships can be found in the graph. An implicit trust relationship is derived from another one by applying some task dependency. For example, we can consider two dependent tasks, "Reading a file" and "Overwriting a file". Obviously they are trust dependant tasks, as trusting someone to overwrite some file should imply trusting him for reading that file too.

Those implicit trust relations depend on the kind of trust dependability that we allow in our system. The dependability rules have to be taken into account when reducing the trust graph for a given task. The dependency among tasks that we use in this paper is inspired in the definitions of the syntax of the *RT* framework, a family of Role-based Trust management languages for representing policies and credentials in distributed authorization [14]. In this work the authors define four different types of relationships among roles. If the relationships in the model are simple, these relationships can be modelled by using a partial order. This is the case for our purpose in this paper, a model of tasks, which are quite an objective concept. Next we will give some definitions.

Definition 1 (Trust Domain). *A trust domain is a partially ordered set* $(TD, <, 0)$ *where every finite subset of* TD *has a minimal element in the subset and* 0 *represents the minimal element of* TD*.*

Each entity in the system makes trust statements about the rest of the entities, regarding the task considered for each case. Those trust statements are defined as follows,

Definition 2 (Trust Statement). *A trust statement is an element* $(Trustor, Trustee, Task, Value)$ *in* $E \times E \times T \times TD$ *where,* E *is the set of all entities in the system;* T *is a partially ordered set representing the possible tasks, where the order established on tasks is* $\preceq$*; and* TD *is a Trust Domain.*

Let $G \subset E \times E \times T \times TD$ be a set of trust statements, and let x_0 be a fixed task in T, then G_{x_0} is defined as the set of trust statements of G such that the corresponding task is placed in an upper position in the task hierarchy, i.e.,

$$G_{x_0} = \{(s,t,x_0,v) \in E \times E \times T \times TD \text{ such that there exists } x \in T \text{such that } (s,t,x,v) \\ \in G \text{ and } x_0 \preceq x\}$$

Let now s_0 and t_0 be two fixed entities, then we can filter G in order to obtain a new set, G^{s_0,t_0}, as the trust statements of G such that they are part of a path from s_0 to t_0. We can combine the two filtering methods together to obtain a new set, $G_{x_0}^{s_0,t_0} = G^{s_0,t_0} \cap G_{x_0}$. We will see what these two sets are useful for in Section 3.

3 Trust Evaluations

If we want to establish trust between two entities in a system this trust should be measured somehow. A simple way to measure trust could be established by using a binary discrete model where the trust values are set as *a lot of trust*, for a very trusted entity, or *very little trust* if the trust placed in the entity measured is very low. More complicated systems could use integer numbers (Advogato's trust metric or FreeHaven [2]) or real numbers ([3, 15]).

A *trust evaluation* or trust metric is a function such that given a trust graph, G, and two entities s and t, called the source and the target of trust respectively (trustor and trustee are alternative names for those entities) returns the level of trust or confidence that s places on t.

As the same entities can be trusted in different ways depending on the task to perform, this function also takes into account as a parameter the task we are referring to, in case there is more than one.

3.1 Trust Functions

Definition 3 (Trust Evaluation). *A trust evaluation for a trust graph G is a function $\mathscr{F}_G : E \times E \times T \longrightarrow TD$, where E, T and TD are the sets mentioned in Definition 2.*

We say that a trust evaluation is *local* if for any tuple $(s,t,x) \in E \times E \times T$, $\mathscr{F}_G(s,t,x) = \mathscr{F}_{G_x^{s,t}}(s,t,x)$, i.e., only those trust statements in $G_x^{s,t}$ are relevant for the evaluation.

In this work we focus on local trust evaluations, in particular on those trust evaluations that can be decomposed in two elemental functions: the Sequential Trust Function and the Parallel Trust Function. By decomposed functions we mean that the trust evaluation is computed by applying the Parallel Trust function to the results of applying the Sequential Trust Function over all the paths connecting two given entities.

Definition 4 (Sequential Trust Function). *A sequential trust function is a function, $f : \bigcup_{n=2}^{\infty} \overbrace{TD \times \cdots \times TD}^{n} \longrightarrow TD$, that calculates the trust level associated to a path or chain of trust statements, such that $f(v_1,\ldots,v_n) = 0$ if, and only if, $v_i = 0$ for any $i \in \{1,\ldots,n\}$, where $v_i \in TD$ and TD is a trust domain.*

Each path of trust statements in G is represented as the chain, $t_1 \xrightarrow{v_1} t_2 \xrightarrow{v_2} \cdots \xrightarrow{v_{n-1}} t_n \xrightarrow{v_n} x_{n+1}$, where t_i are entities in E and v_i are respectively the trust values associated to each statement.

The sequential trust function, f, may verify some of the following properties:

- Monotony (Parallel Monotony): $f(v_1,\ldots,v_n) \leq f(v'_1,\ldots,v'_n)$ if $v_i \leq v'_i$ for all $i \in \{1,\ldots,n\}$.
- Minimality: $f(v_1,\ldots,v_n) \leq min(v_1,\ldots,v_n)$
- Sequential monotony: $f(v_1,\ldots,v_{n-1},v_n) \leq f(v_1,\ldots,v_{n-1})$
- Preference Preserving: $f(v_1,\ldots,v_i,\ldots,v_j,\ldots,v_n) < f(v_1,\ldots,v_j,\ldots,v_i,\ldots,v_n)$ if $v_i < v_j$.
- Recursion: $f(v_1,\ldots,v_n) = f(f(v_1,\ldots,v_{n-1}),v_n)$

When defining a recursive sequential function we have to take into account that it is enough to define it over pairs of elements in TD, since by applying the recursion property we could obtain the value of the function for any tuple.

We call *generator sequential function or sequential operator* to the function f restricted over the domain $TD \times TD$. We represent it by $\odot$. Thus,

Definition 5 (Sequential Operator). *A Sequential Operator or Generator Sequential Function is defined as a function $\odot : TD \times TD \longrightarrow TD$ such that $a \odot b = 0$ if and only if $a = 0$ or $b = 0$. $\odot(a,b)$ or $a \odot b$ are used indistinctively for representing the same, whatever is more convenient.*

Given a recursive sequential function, f, the associated sequential operator $\odot_f$, can be defined as $a \odot b = f(a,b)$. Viceversa, given a sequential operator, the recursive inference sequential function can be defined as $f_{\odot}(z_1,\ldots,z_{n-1},z_n) = f_{\odot}(z_1,\ldots,z_{n-1}) \odot z_n$.

Note that a recursive sequential function verifies the reference preserving property only if the associated sequential operator, $\odot_f$, is not commutative.

Moreover, if $a \odot b \leq min(a,b)$, for any a and b, we could conclude that f verifies the minimality property.

Definition 6 (Parallel Trust Function). *A parallel trust function is used to calculate the trust level associated to a set of paths or chains of trust statements. It is defined as,* $g : \bigcup_{n=2}^{\infty} \overbrace{TD \times \cdots \times TD}^{n} \longrightarrow TD$*, where TD is a trust domain and*

1. $g(z_1, \ldots, z_{i-1}, z_i, z_{i+1}, \ldots, z_n) = g(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n)$ *if* $z_i = 0$
2. $g(z) = z$

g may verify the following desirable properties:

- Idempotency, $g(z, z, \ldots, z) = z$.
- Monotony, $g(z_1, z_2, \ldots, z_n) \leq g(z'_1, z'_2, \ldots, z'_n)$ If $z_i \leq z'_i$ for all$i \in \mathbb{N}$.
- Associativity, $g(g(z_1, z_2), z_3) = g(z_1, z_2, z_3) = g(z_1, g(z_2, z_3))$.

The *generator parallel function*, or the *parallel operator*, $\oplus$ for the function g, is defined analogously as the operator $\odot$.

Definition 7 (Parallel Operator). *A Parallel Operator or Generator Parallel Function is defined as a function,* $\oplus : TD \times TD \longrightarrow TD$*, such that* $a \oplus 0 = 0 \oplus a = a$

We say that the two operators $\oplus$ and $\odot$ are distributive if $(a \oplus b) \odot c = (a \odot c) \oplus (b \odot c)$.

In the case where there are no cycles in the trust graph, the set of paths connecting two any given entities is finite. Then, given a sequential operator $\odot$ and a commutative parallel operator $\oplus$, i.e. $a \oplus b = b \oplus a$, the associated trust evaluation, $\widehat{\mathscr{F}_G}$, is defined as follows,

Definition 8. *Let $S_x^{s,t}$ be the set of all paths of trust statements for task x starting in s and ending in t. For each path $p \in S_x^{s,t}$ represented as $s \xrightarrow{v_1} \cdots \xrightarrow{v_n} t$ let z_p be $v_1 \odot \cdots \odot v_n$, then $\widehat{\mathscr{F}_G}(s,t,x)$ is defined as $\bigoplus_{p \in S_x^{s,t}} z_p$.*

Given a fixed sequential operator, for any parallel operator that verifies idempotency and monotony properties then, $z_* = min_{p \in S_x^{s,t}} z_p \leq \bigoplus_{p \in S_x^{s,t}} z_p \leq max_{p \in S_x^{s,t}} z_p = z^*$. Therefore, the maximum and minimum possible trust values associated to a path from s to t are the upper and lower bounds for the trust evaluation $\widehat{\mathscr{F}_G}$.

Fortunately, we do not need to compute the trust values of each path in order to compute those bounds, i.e. z_* and z^* . In this case we can use an algorithm, adapted from the Dijkstra algorithm [8], to find for example, the maximum trust path from a fixed entity s to any other entity on the system. The minimum trust path can be computed in an analogous way.

This is a particular case of a trust evaluation where we use the maximum function as a parallel function. Unfortunately we can not generalize this kind of algorithms for other combinations of parallel and sequential functions as it heavily relies on the properties of the max and min functions.

3.2 Performing Trust Computations using Matrices

Let us first assume the case where there are no cycles in the trust graph. We could model the trust network as a matrix, A where each element a_{ij} represents the trust level that node i places on node j. If we replace the scalar addition and multiplication in matrices by the operators $\oplus$ and $\odot$ respectively, then by iterating powers of the trust network we can compute the node to node trust values of the network. Thus, the trust evaluation could be defined through $\odot$ and $\oplus$ applying the generalized matrix product algorithm. It is then defined as $A \otimes B = \bigoplus_{k=1}^{n}(a_{ik} \odot b_{kj})$.

The generalized product is associative from the left hand side, i.e., $A \otimes B \otimes C = (A \otimes B) \otimes C$. Therefore, we can define the generalized power of A as $A^{(n)} = \bigotimes_{k=1}^{n} A$.

Last, we can define the operator $\oplus$ over matrices as $A \oplus B = (a_{ij} \oplus b_{ij})$, which can be used as the summation of matrices. Then, given a matrix A of order n, the matrix $\widehat{A}$ can be defined as $\widehat{A} = \bigoplus_{k=1}^{n} A^{(n)}$.

These definitions become more relevant when the aforementioned functions are distributive and associative in the case of the parallel function, and recursive in the case of the sequential function. However, they are still valid if these properties do not hold, although they may not be that meaningful.

Next, we will show that under the conditions mentioned above, the element (i,k) in the matrix $\widehat{A}$ is the value of the trust evaluation of all the trust paths from i to j. In particular, the first row in this matrix will give us the distribution of trust in the system.

First, we will show that the kth generalized power of the trust matrix A, $A^{(k)}$ contains the trust through paths of length k. We will prove this by induction where the base case holds by the definition of matrix A.

We assume that for $k \in \mathbb{N}$ (i,j) in $A^{(k)}$, $a_{ij}^{(k)}$, represents the value of the trust evaluation of all the trust paths of length k. We will then show that this is also the case for length $k+1$.

Let $C_{ij} = \{c_{ij}^{1}, \ldots, c_{ij}^{m_{ij}}\}$ be the set of all the paths of trust values from all the paths of length k from i to j. Then since $A^{(k+1)} = A^{(k)} \otimes A$, each element of the matrix can be obtained by using the function g as $a_{ij}^{k+1} = \bigoplus_{l=1}^{n}(a_{il}^{(k)} \odot a_{lj}) = g(c_{i1}^{1} \odot a_{1j}, \ldots, c_{i1}^{m_{i1}} \odot a_{1j}, \ldots, c_{in}^{1} \odot a_{nj}, \ldots, c_{in}^{m_{in}} \odot a_{nj})$.

Let now $c \equiv i \xrightarrow{z_1} c_1 \xrightarrow{z_2} \cdots c_n \xrightarrow{z_k} l \xrightarrow{z_{k+1}} j$ be a $k+1$ length path from i to j. This path can also be expressed as the path c' of length k from i to l linked with the path from l to j, (l,j). Therefore, since the sequential function is recursive, any path of length $k+1$ from i to j can be obtained adding a link to any path of length k. Thus, a_{ij}^{k+1} represents the value of the trust evaluation of all the paths of length $k+1$ from i to j. The number of elemental operations (sequential and parallel operations) for computing $\widehat{A}$ is

$$\frac{n(n^3 - 2n^2 - n + 2)}{12} \tag{1}$$

The important issue is that the order of operations is $\mathscr{O}(n^4)$.

3.3 The Problem with Cycles

If there are cycles in the trust graph the previous definitions and algorithm are not valid, therefore we need and extra algorithm to compute the trust values for this case. In fact, the new algorithm we are going to introduce is only needed when we are computing the trust value of a node involved in a cycle.

The key aspect of this new algorithm is to remove redundant graphs from the trust graph in such a way that the set of paths connecting two given entities remains finite.

Let i and j be two nodes in the system and m a natural number, then we can define the set S_{ij}^m as the subset of the permutation group S_n containing all the cycles, σ, of length $m+1$ such that $\sigma^m(i) = j$.

The cardinality of the set S_{ij}^m is $\mid S_{ij}^m \mid = \frac{(n-2)!}{(n-m-1)!}$. Thus, the number of elements is of the order $\mathscr{O}(n^{m-1})$.

The intuition behind the new algorithm is the same as for the previous one except by the fact that we only modify the way we compute the elements of the matrices $A^{(m)}$. In the case of nodes which are not involved in any cycle the two algorithms provide the same result.

For the new algorithm $a_{ij}^{(1)} = a_{ij}$ and $a_{ij}^{(m)} := \sum_{\sigma \in S_{ij}^m} a_{i,\sigma(i)} \odot \cdots \odot a_{\sigma^{m-1}(i)j}$.

The number of parallel operations performed by this algorithm is $(n^2)\frac{(n-2)!}{(n-m-1)!}$. This number is of the order of $\mathscr{O}(n^{m+1})$. As the length of the trust path is m, each of the components in the previous operations need $m-1$ sequential operations therefore the total number of sequential operations is of the order of $\mathscr{O}((m-1)n^{m+1})$. Thus, adding up these two number of operations, we can conclude that the total number of operations is $\mathscr{O}(mn^{m+1})$ only for the matrix $A^{(m)}$. Therefore, if we compute all the trust paths for all $m \in \mathbb{N}$ the amount of operations will grow enormously. If we compare this number with the number of operations in Equation 1 we can conclude that this latter number is much bigger for any $m > 3$.

As we can see by observing the number of operations for both algorithms, they are higher for the new algorithm, therefore it will be convenient to avoid cycles, if possible. We might need to apply some techniques for this, for example, we can include a timestamp in each trust statement and remove the problematic edges depending on when they were created.

4 Examples of the Model

The properties of the system are going to be derived from the properties of the operators $\odot$ and $\oplus$. Depending on these properties we could classify or outline the systems.

As we will only consider recursive functions we will deal directly with operators instead of functions. For simplicity and for showing the model purposes, we will consider the initial trust domain to be the interval $[0,1]$, where 0 stands for null trust and 1 for full trust. However, other more complex, non-scalar domains can be used.

We only consider two possibilities for the sequential operator: Product and Minimum; and for the parallel operators Maximum, Minimum and Mean. Regarding the sequential operators, their definitions are straightforward. Both of them trivially verify monotony, minimality and sequential monotony properties. The product also verifies what we could call “Strict Sequential Monotony”. This means that $v_1 \odot v_2 < 1$ if $v_2 < 1$. The preference preserving property does not hold for any of them.

Note that in order to be able to perform the computation of trust in a distributed manner we need the operators to be distributive. Then the problem of defining a parallel operator become harder as we have to make them compatible with the definition of the sequential operators.

4.1 Completing the Model Instances

In this section we will concentrate on parallel operators.

Maximum and Minimum The Maximum and Minimum are two examples of parallel operators which verify the associativity property as well as idempotency and monotony properties. The Minimum operator however does not verify the definition of parallel operator strictly as the minimum of any set containing 0 will be 0. This problem can be solved by erasing the 0s of such a sets before applying the function. The resulting operator with this new domain is called $\oplus_{min^*}$.

$$v_1 \oplus_{min^*} v_2 := \begin{cases} min\{v_1, v_2\} & \text{if } v_1 \neq 0 \text{ and } v_2 \neq 0 \\ v_1 & \text{if } v_2 = 0 \\ v_2 & \text{if } v_1 = 0 \end{cases}$$

Both operators, $\oplus_{min^*}$ and $\oplus_{max}$ verify the distributivity property with respect to the sequential operator minimum, i.e.,

1. $(v_1 \oplus_{max} v_2) \odot_{min} \lambda = (v_1 \odot_{min} \lambda) \oplus_{max} (v_2 \odot_{min} \lambda)$
2. $(v_1 \oplus_{min^*} v_2) \odot_{min} \lambda = (v_1 \odot_{min} \lambda) \oplus_{min^*} (v_2 \odot_{min} \lambda)$

and distributivity with respect to the sequential operator product, i.e.,

1. $(v_1 \oplus_{max} v_2) \odot. \lambda = (v_1 \odot. \lambda) \oplus_{max} (v_2 \odot. \lambda)$
2. $(v_2 \oplus_{min^*} v_2) \odot. \lambda = (v_1 \odot. \lambda) \oplus_{min^*} (v_2 \odot. \lambda)$

As we mentioned in Section 3.1 the maximum and minimum functions are the upper and lower bounds of any parallel function that verifies the idempotency and monotony properties. The difference between the highest trust and the lowest trust will be the range of the variation of trust for any other election of the parallel operator. This range of values will give us an average of the deviation of trust.

Mean The mean verifies idempotency and monotony properties but, however, does not verify the associativity property. We could solve this problem by using a different trust domain $\mathscr{T} := [0,1] \times \mathbb{N}$ and defining the operator as follows,

$$\begin{array}{rcl} \oplus_{Mean*} : ([0,1] \times \mathbb{N}) \times ([0,1] \times \mathbb{N}) & \longrightarrow & ([0,1] \times \mathbb{N}) \\ ((v_1, n), (v_2, m)) & \longmapsto & \begin{cases} \left(\dfrac{v_1 \cdot n + v_2 \cdot m}{n+m}, n+m \right) & \text{if } v_1 \neq 0 \text{ and } v_2 \neq 0 \\ (v_1, n) & \text{if } v_2 = 0 \\ (v_2, m) & \text{if } v_1 = 0 \end{cases} \end{array}$$

where '$\cdot$' is the usual product in $\mathbb{R}$.

With this definition the operator is associative and still verifies idempotency and monotony properties. The distributivity property only holds for the sequential operator product defined in the trust domain $[0,1] \times \mathbb{N}$ as $(v_1, n_{v_1}) \odot (v_2, n_{v_2}) = (v_1 \cdot v_2, n_{v_1} \cdot n_{v_2})$.

The generalized product of matrices can be applied to the combination $(\odot, \oplus_{Mean^*})$. This will allow us to calculate the mean of the trust values of any entity by using the matrix $\widehat{A}$ in the domain $[0,1] \times \mathbb{N}$. Note that the first component of a trust value in this domain represents the actual trust level, whereas the second one represents the number of paths considered for this computation.

4.2 Summary of the Examples

We have proposed the following combination of operators of our model based on the operators defined along the paper.

1. $(Min, Min*)$. In this case the minimum is the sequential function, Min, and the minimum of the non-null elements, $Min*$, is the parallel function.
2. (Min, Max). The function minimum is the sequential function and the maximum, Max is the parallel function.
3. $(Product, Min*)$. The function product is the sequential function and $Min*$ is the parallel function.
4. $(Product, Max)$. The function product is the sequential function and Max is the parallel function.
5. $(Product, Mean*)$. The product is the sequential function and the mean of the non-null elements is the parallel function. The trust domain in this case is not the interval $[0,1]$ as in the other cases but $[0,1] \times \mathbb{N}$.

5 Conclusions and Future Work

We have introduced a general model of trust that splits the process of computing trust in two sub-processes. In order to do this we have introduced the sequential and the parallel operators, which will be the parameters used in our model, together with a trust domain that can be any set used for defining trust values. Depending on those parameters, the trust values computed by the model will vary.

We assume that trust between entities is directional and can be partial. By partial we mean that it can be related to some specific task but not all of them, i.e. one can trust someone else to perform a certain task in an honest manner but not for the other tasks. Those tasks can be related, therefore it could be useful to order or classify them in a hierarchical diagram, in a way that trusting some entity for a certain task will imply trusting this entity for all the tasks that are lower in the hierarchy. How to order tasks is out of the scope of this paper although we consider it is an important aspect of the model that remains for future work.

We have defined the model by using a labelled trust graph where the label is of the form (t, x_0), where t is the value of trust and x_0 is a task. When performing trust evaluations, we set a fixed task in order to be able to remove this parameter from the labels.

The resulting weighted graph is then processed by using the sequential and parallel operators. We have also presented some sample uses of our model with simple operators and a simple trust domain (the interval [0,1]) to show that the model is useful when defining new trust evaluations.

In the future we intend to investigate the possibility of using different trust domains other than $[0,1]$. We are particularly interested in investigating the trust domain that the Subjective Logic by Jøsang uses [9]. Analysis of a trust network using subjective logic have been already carried out [11]. Our intention is to analyze the operators defined for it them according to the properties that we have defined and therefore, how our model could be suitable for them. We also intend to define new operators and study the properties that they may verify.

References

1. http://www.ebay.com.
2. http://www.freehaven.net/.
3. http://www.openprivacy.org/.
4. M. Blaze, J. Feigenbaum, and A. D. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures (position paper). *Lecture Notes in Computer Science*, 1550:59–63, 1999.
5. M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *IEEE Symposium on Security and Privacy*, 1996.
6. M. Burrows, M. Abadi, and R. M. Needham. A Logic of Authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.
7. Y.H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust Management for Web Applications. *Computer Networks and ISDN Systems*, 29:953–964, 1997.
8. E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
9. A. Jøsang and R. Ismail. The Beta Reputation System. In *15th Bled Electronic Commerce Conference e-Reality: Constructing the e-Economy*, Bled, Slovenia, June 2002.
10. A. Jøsang, R. Ismail, and C. Boyd. A Survey of Trust and Reputation Systems for Online Service Provision. *Decision Support Systems*, 43:618–644, 2007.
11. Audun Jøsang, Ross Hayward, and Simon Pope. Trust network analysis with subjective logic. In *ACSC '06: Proceedings of the 29th Australasian Computer Science Conference*, pages 85–94, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.
12. R. Leiven. *Attack Resistant Trust Metrics*. PhD thesis, University of California at Berkeley, 2003.
13. R. Leiven and A. Aiken. Attack-Resistant Trust Metrics for Public Key Certification. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, USA, January 1998.
14. Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.
15. S. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, Department of Computer Science and Mathematics, University of Stirling, 1994.
16. C. N. Ziegler and G. Lausen. Spreading Activation Models for Trust Propagation. In *IEEE International Conference on e-Technology, e-Commerce, and e-Service (EEE'04)*, Taipei, March 2004.
17. C.-N. Ziegler and G. Lausen. Propagation Models for Trust and Distrust in Social Networks. *Information Systems Frontiers*, 7(4-5):337–358, December 2005.