# The KPI-Based Reputation Policy Language

Slim Trabelsi
SAP Research
Mougins, France
slim.trabelsi@sap.com

Luca Boasso
Politecnico Di Torino
Torino, Italy
luca.boasso@studenti.polito.it

***Abstract*—Trust policy languages are implemented to express the trust requirements of the users. These requirements are represented by a set of rules specifying the necessary conditions that should be fulfilled by an entity in order to gain the trust of the evaluator. Most of the known trust policy languages are designed to express credential, authorization and access control requirements for the trust establishment. The credential based approach represents only one aspect of trust. The other main aspects like reputation and recommendation are not covered by these policy languages. In this paper we propose a new policy language for expressing trust requirements for reputation models, and particularly for the KPI-based reputation model in a supply chain scenario.**

***Index Terms*—Trust, Reputation, Supply, Policy, Language, KPI,**

## I. INTRODUCTION

Trust is a subjective matter, it depends on a truster's subjective evaluation of past experiences and it depends on the characteristics of the trustee [1]. Still, for making trust usable, we need to be able to express it in terms of measurable quantities (trust metrics and reputation). These metrics will be clearly not able to capture the many subjective and context dependent facets of this complex sociological phenomenon, but still they may be used in a specific context to assess the reliability of the trustee and its capability to ensure privacy, security, and so on. Nowadays, the notion of trust does not rely only on the traditional trust infrastructure (based on certificate verification or recommendation) but it is strongly related to the behavior of the users and their virtual reputation. The traditional trustworthiness models implemented in the internet, such as Amazon or E-Bay, are relying on a subjective rating system in which users estimate the "quality" of the transaction over a numerical scale. Knowing that nobody is able to formalize and explain the difference between two successive values like a transaction rewarded at 9/10 and another one 10/10, we cannot really estimate the correctness and the objectivity of the trust and reputation value. In addition, such trustworthiness models are limited in terms of federation and adaptability; in fact it is very hard to adapt the perception of trust in different domains and conditions. For example the reputation of a transporter cannot be exported to packaging and storage domain, because there is no possibility to map the subjective trustworthiness values between two different domains with two different trust perceptions. In order to address the limitations above, we proposed a less subjective trust model taking into account quantifiable parameters for the computation of the trustworthiness value of an entity [11]. We called these parameters KPI for Key Performance Indicators. In the context of trust, the goal of using these metrics is to quantify the sources of trust and adapt them to the personal perception of each trusting entity. For example if a shipment and a distribution company were sharing the same KPI parameters for their reputation model, like transportation time, package quality, quality of goods, price etc. the federation of trust between these two domains can be handled in an easy way by adapting the reputation calculation according to the local perception of trust. Each trustee in the supply chain can configure a pattern for his trust model according to his objectives and his trust perception expressed through a formal language, for example the trusting entity that prioritizes the delivery time of a good, will obtain a different reputation value than another user that prioritizes the CO2 footprint.

In this paper, we propose a new reputation policy language for expressing easily the personalized trust requirements related to the KPI-based trust model. There are few languages describing the reputation-based trust models, and in the majority of the cases they are not designed for non-expert users, therefore they are far from being user friendly. On the other hand, an increasing number of people are starting to use trust models in the supply chain industry in order to evaluate the trustworthiness of the different nodes of the chains. Most of the time users in that domain are not necessarily security experts, and they encounter major obstacles in the configuration and personalization of reputation models. For this reason we propose here a user friendly policy language, able to express in a very simple way the trust requirements of a user who wants to evaluate the reputation of an entity according to its performance parameters. The long term goal of this language is not limited to KPI based trust model, but to support most of the behavioral trust models.

This paper is organized as follows: in Section 2, we discuss the related work in the domain of Trust and reputation policies, in Section 3, we present a brief use case scenario, in Section 4, we define the KPI-based reputation model, in Section 5, we present the policy language specification, then we briefly provide the implementation details, and finally we conclude our work.

 ISBN: 978-1-61208-144-1 

## II. RELATED WORK

In the literature, the two aspects of policy based and reputation based management are usually separated [9]. On one side, the policy based trust management focuses on problems related to authorization and access control in open systems; i.e., it determines whether or not an unknown user can be trusted, based on a set of credentials and on a set of policies. On the other side, reputation-based management assesses the trust relationships based on non-certified available information, like recommendations or previous experiences of other users. In this paper we show how we can merge these two approaches by providing a policy language that expresses the trust requirements from a reputation model.

Among the existing trust policy languages we can mention TPL (Trust Policy Language) [6] that is a XML-based language defining the relation between unknown entities to roles. It expresses a mechanism that allows a business to define a policy to map accessed users to roles, based on certificates received from the user and collected automatically by the system. The XML nature of the language makes it appropriate for automated processing, but less suitable for human users. Bonatti and Samarati [5] proposed the PSPL language to regulate service access and information release in large scale networks. This language is designed to express access and release policies in conjunction with a policy filtering mechanism, which allow the parties to exchange their requirements in a compact and privacy preserving way. PSPL has a Prolog-like syntax, in which one can define rules that take into account the elements of the trust model. Blaze et al. proposed the KeyNote policy language [8] that provides a simple notation for specifying both local security policies and security credentials that can be sent over a non trusted network. KeyNote policies and credentials, called "assertions", contain predicates describing the trusted actions permitted by the holders of specific public keys. KeyNote assertions are small and structured programs written in a simple notation based on C-like expressions and attribute/value pairs actions.

All these trust policy languages do not support the reputation models. And, to our knowledge there are very few studies trying to address the expressivity of the reputation requirements by a policy language like TriQL.P [10] that plans to propose a reputation dedicated language, but up to now few results are available from this project .

## III. USE CASE

To illustrate our approach, let us consider a simple supply chain use case. Let us consider an active transport tracking devices attached to returnable transport items, such as crates, rolling containers, pallets and shipping containers. Consider a shipment of milk as it travels from the farm near Rennes in France to a supermarket distribution center in Paris. After collecting the milk in the farm, the farmer has to use a small tank truck to carry his daily production to the local milk collecting center. There the milk is packaged then assembled to pallets and finally charged up to huge transportation trucks. The trucks chip the bricks of milk to the supermarkets in Paris. In order to monitor and evaluate the quality of the transportation process from the farm to the distribution center the supermarket quality manager will setup a KPI requirement list in which he defines the all the quantifiable thresholds that should be satisfied during the entire process. The metrics chosen by the quality manager are for example the transportation time between the farm and the local collecting center, the average temperature, the packaging time and cost, the transportation time between the packaging factory and the supermarket in Paris, the average temperature during the transportation etc. All these indicators are provided by tamperproof sensors. The quality manager usually defines KPIs that represent his business objectives and compute a *reputation* score for each actor in the chain according to the compliance with the requirements described above.. For example a *good* temperature average should be between 3 and 4 degrees Celsius. The transportation time between Rennes and Paris should be 5 hours (more is bad, less is good), etc.

For each delivery day the manager collects from different sensors the indicator values and integrates it to the reputation model in order to evaluate the score of each actor contributing to the chain. This example is quite trivial, and any manager can compute the score with a spreadsheet. The problem becomes serious when in real cases, some managers have to take into account a large number of KPIs. These performance indicator values may be gathered through different sensors located in different places and communicating via different protocols (for example in a remote database, in a XML file from a web service…). In that case the user should have a wide range of computer skills just to collect and convert the values in a suitable format and then to compute the reputation values. This is not always the case. For this reason we define in this paper a formal model to collect KPI values and compute the final score, as well as an easily accessible policy language that expresses these requirements.

## IV. KPI-BASED TRUST MODEL

We propose a KPI-based trust Model (KPITM) [11] as an approach in which trust evaluation is based on KPIs shared by different users.

### A. Repuation model

The KPI-based reputation model takes into account trust metrics based on KPI. In this approach, a user can express his trust preferences via an expressive language that specifies the sources of the performance indicators factors and how the factors should be combined to obtain a reputation score. According to his business objectives, the user is able to prioritize some indicators by setting a strong weight affecting the result of the trust score. These indicator values are then normalized (between 0 and 1) and then aggregated in order to obtain a unified reputation value.

The normalization rule is written as follows:

 ISBN: 978-1-61208-144-1 

$$\begin{cases} 1 & \text{if } K_i > K_{max} \\ 0 & \text{if } K_i < K_{min} \\ \dfrac{K_i - K_{\min}}{K_{\max} - K_{\min}} & \text{otherwise} \end{cases}$$

*Higher is better KPI normalization*

$$\begin{cases} 0 & \text{if } K_i > K_{max} \\ 1 & \text{if } K_i < K_{min} \\ \dfrac{K_{\max} - K_i}{K_{\max} - K_{\min}} & \text{otherwise} \end{cases}$$

*Lower is better KPI normalization*

Where $K_i$ is the measured performance indicator value$K_{min}$ and $K_{max}$ are the minimum and maximum values declared in the objectives scale. If lower values are better (e.g., the delivery time example) the value is reversed by subtracting it from 1.

In the KPI-based model, each item has a semantic meaning that explains the context of the measured value related to any performance parameter (e.g., delivery time, temperature, CO2 impact, etc.). The combination of different KPI items offers the possibility to the trustee to customize his trust evaluation by expressing complex semantic queries.

An entity that wants to connect to a KPITM system in order to evaluate the trust of another entity has to select three elements: first, the KPI items that are relevant for him, second, the location where to find the performance indicator value and, lastly, the weights of each KPI in order to prioritize some values during the trust evaluation. All this information must be contained in the core query sent to the KPITM engine that will automatically connect to the different sources, get the values of each item and compute the trust value.

The KPI-based trust model offers the possibility to quantify the trustworthiness according to some domain specific objectives (how should be the conservation temperature range for the milk ) and it permits to any trustee entity to determine, which tested element is more trustworthy according to an objective estimation. In particular, our KPI-based trust model allows a trustee to evaluate the weight of a recommendation by applying the business objective scale of the recommender.
More formally, the KPI-based trust model used is composed of three complementary layers:

- Performance Indicator Values: they are collected from the different sources providing the values related to the performance items
- Business Objectives Scale: it is defined by the trustee according to the performance indicators related to their business objectives. An interval of values (min and max) must be chosen for every performance indicator in order to normalize the measured value with a [0, 1] scale. Furthermore a weight factor must be defined to prioritize the performance indicators and to compute the final trust value.
- Trust Level Value: it is the aggregation of all the normalized performance indicators plus, possibly, some external values like the recommendation from other trusted entities.

For example in our scenario these layers are represented in Figure 1, the weight factors are within the circles, whereas the interval of values is represented by the double ended arrow in the same layer.

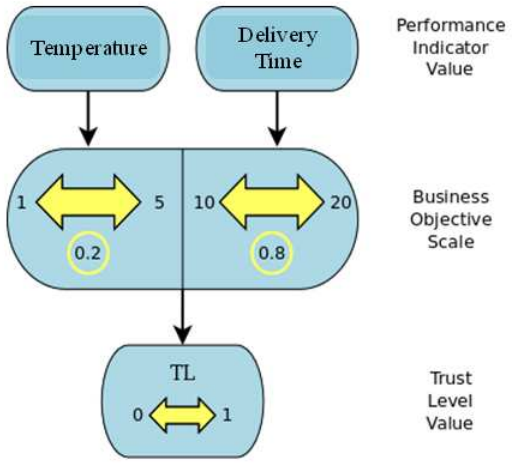


Fig1. KPI-based Trust model of our scenario

### B. *Architecture*

We proposed a loosely coupled architecture (Figure 2) for managing the KPI based trust in which we have three independent and complementary layers:

- The Indicator sources: we proposed two kinds of interfaces in order to collect the indicator values that should be used to compute the reputation: a database connector used as an interface to get access to any kind of local or remote database. A Web Service interface for collecting the indicator values published as Web Services
- The KPITM engine in charge of computing the reputation value according to the trust model described in the previous section. It interprets the queries sent by the user via a UI or the policy language, and then it uses the collected indicator values to compute the reputation.
- KPI-based reputation language engine: this component interprets the queries written in the policy language (human readable language used to express the reputation/trust requirements) and translates it into a remote call to the KPITM engine in order to calculate the reputation value.

The choice of this kind of decoupled architecture is motivated by the requirement to have a generic solution independent from the trust/reputation model and from the sources of trust. In this paper, we describe a specific case, where the trust model is based on the KPIs only, but in other

 ISBN: 978-1-61208-144-1 

cases, we may use another reputation model with other sources of trust, and we want to develop a language generic and flexible enough to be used with a wide range of trust and reputation models. Each layer of this architecture is independent and replaceable.

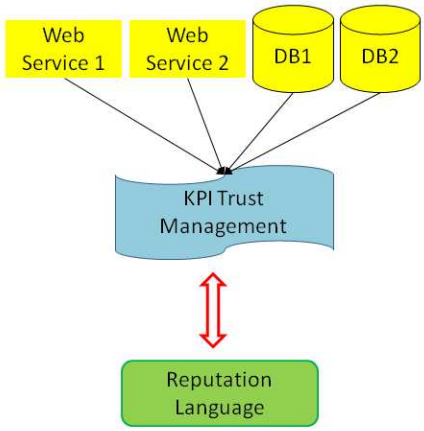


Fig2. Architecture

## V. KPI-BASED POLICY LANGUAGE

The reputation language allows a user to define queries for the KPITM engine in a simple and concise way. No programming knowledge is required since we propose a script based language.

Referring to our scenario, the following code calculates the trust value of an actor in the chain that we will cal FARMER for example:

```
? example
Actors[Farmer] -> Delivery (20:30:0.8)
               -> Temperature(3:5:0.2)
```

According to our trust model, this query specifies the KPI that are relevant to the manager (*time* and *temperature*), the location of the performance values (location Actors for the actor *Farmer*) and finally the range of acceptable values and the weight for each KPI ( 20:30:0.8, i.e. delivery time can vary between 20 and 30 minutes with a weight of 0.8).

### A. Language Specification

A formal language is a set of sequences of symbols. Elements of this set are called sentences. In the KPI language sentences are programs called scripts. The symbols originate from a finite set called the vocabulary. The set of programs (which is infinite) is defined by rules of their composition. Sequences of symbols that are composed by these rules are said to be syntactically correct or well-formed. The set of rules is the syntax of the language. The program (or sentence of the formal language) consists of parts called syntactic entities, such as declarations, statements or expressions.

Parentheses may be used to group factors or terms. The notation introduced here is known as Extended Backus-Naur Formalism (EBNF) [2].

Besides syntactic entities, denoted by identifiers, we need to substitute elements, also called tokens, taken from the formal language's vocabulary. The vocabulary of the KPI language consists of identifiers, numbers, strings, operators, delimiters and comments. They are called lexical symbols and are composed of sequences of characters. (Note the distinction between symbols and characters.)

In the EBNF notation non-terminal symbols are denoted by English words expressing their intuitive meaning. Terminal symbols are denoted by strings enclosed in quote marks.

### B. Lexical Analysis

The representation of terminal symbols in terms of characters is defined using the Latin-1 set. Terminal symbols include identifiers, numbers, strings, operators, delimiters and comments. Blanks and line breaks must not occur within symbols (except in comments and blanks in strings). They are ignored unless they are essential to separate two consecutive symbols. Capital and lower-case letters are considered as being distinct.The lexical rules are now considered in detail:

**i.** An identifier (*ident*) starts with an upper-case letter followed by a sequence of zero or more letters or digit or the special character "_":
Examples: Actors, Temperature, green_car10

**ii.** Numbers are of type real, a sequence of digit followed by an optional decimal part:

```
real = digit {digit} [ "." digit {digit} ].
```

Examples: 3.14, 8, 6.33.

**iii.** A string is a sequence of characters enclosed in quotation marks. A string cannot contain the delimiting quote mark:

```
string = '"' {character} '"' | "'" {character} "'".
```

Examples: "This", "is 'a'", 'short "string"'.

**iv.** Operators and delimiters are the special characters, character pairs or reserved words listed below. These reserved words cannot be used as identifiers.

**v.** Comments start with a hash character "#" that is not part of a string and ends at the end of the physical line.

### C. Syntax and Semantic

A script begins with an optional chart declaration followed by a sequence of statements:

```
script = [chart] statement {statement}.
```

There are two kinds of statements, assignment and query:

```
statement = assignment | query.
```

#### 1) Assignments

An assignment allows the creation of a variable with a value given by an expression:

```
assignment = "var" ident "=" expression.

expression = ["+"|"-"] term { ("+"|"-")
term}.

term = factor {("*" | "/") factor}.

factor =  real | ident |  "(" expression
")".
```

The above rules specify that an expression can use the mathematical operators for addition, subtraction, multiplication and division. These operators have the usual

 ISBN: 978-1-61208-144-1 

arithmetic precedence and they are left-associative. The parentheses are used to group expressions and they have the highest precedence. Variables can appear freely in an expression. For example the following assignments are correct and all the expressions evaluate to 8:

```
var a = 4 + 3 * 10 / 5 - 2
var b = 4 + 3 * (10 / 5) - 2
var c = (4 + 3 * 10 / 5) - 2
```

*2) Query*

A query allows the creation of a list of KPIs:

```
query = "?" ident  item {item}.
```

It starts with a question mark character followed by ident that represents the name of the list. A KPI is defined as an item:

```
item = ident "[" ident {"," ident } "]" type
{type}.

type = "->" ident "(" min_max ":" min_max  ":"
weight ")".

weight = expression | "/".

min_max =  expression | "MIN" | "MAX".
```

The above rules define the exact syntax of a KPI. Some examples from Table 1 will make these rules clear. Consider this script:

```
? Example1 Actors[Farmer] -> Delivery (20:30:1)
```

In order to take into account the KPI of type Delivery a second item is added:

```
? Example2

Actors[Farmer] -> Delivery (20:30:0.6)
Actors[Farmer] -> Temperature(3:5::0.4)
```

Notice that the sum of all the weights in the list must be always one. It is possible to add KPIs with different name, type, location, min, max and weight; they are fully customizable as shown in the following example:

```
? Example3

Actors[Farmer] -> Delivery(20:30:0.6)
Actors[Packaging] -> Delivery(40:60:0.2)
Finance[GOOG] -> Price(30:100:0.2)
```

In Example2 there are two KPIs with same name (Farmer) and location (Actors). That script can be written in this equivalent form:

```
? Example4

Actors[Farmer] -> Delivery(20:30:0.6)
                  -> Temperature(3:5:0.4)
```

Let's suppose that a company wants to evaluate the reputation of two Actors (Farmer and Packaging) based on their delivery time and temperature. A possible script can be:

```
? Example5

Actors[Farmer] -> Delivery(20:30:0.3)
                  -> Temperature(3:5:0.2)
Actors[Packaging] -> Delivery(20:30:0.3)
                     -> Temperature(3:5:0.2)
```

This last example can be rewritten also as:

```
? Example6

Actors[Farmer, Packaging] -> Delivery(20:30:0.6)
                             ->Temperature(3:5:0.4)
```

In fact the four KPIs share the location (Actors) and taken in pair they share also the type, the min, the max and the weight (see Example5). In this way it is enough to list the names inside the square brackets and write the shared part only once. It is important to notice the changes in the weights: the language will split 0.6 and 0.4 like in Example 6 automatically.

*3) Automatic weight and MIN MAX keywords*

Since the sum of the weights for all the items must be equal to one, it is possible to specify only the weights of interest and let the language to calculate the others. To achieve this result use the "/" symbol:

```
? Example7

Actors[Farmer] -> Delivery(20:30:0.6)
Actors[Packaging] -> Delivery(3:5:/)
Actors[Supermarket] -> Delivery(3:5:/)
```

The last two items have a weight of 0.2. So the Example 5 can be rewritten again like this:

```
? Example8

Actors[Farmer, Packaging] -> Delivery(20:30:0.6)
                             -> Temperature(3:5:/)
```

Sometimes can be convenient to use the keywords MIN and MAX:

```
? Example9

Actors[Farmer] -> Temperature(MIN:5:0.6)
                  -> Delivery(20:MAX:0.4)
```

The MIN will be replaced by the lowest value of type Delivery in the Actors location. Accordingly to the table defined before this value is 1. The same reasoning applies for MAX, its value is 45.

As discussed before a script contains a sequence of statements so more than one query (and so KPIs lists) can be written:

```
? Farmer
Actors[Farmer] -> Delivery(20:30:0.8)
                  -> Temperature(3:5:0.2)

? Packaging
Actors[Packaging] -> Delivery(20:30:0.8)
                     -> Temperature(3:5:0.2)
```

The implementation of the language displays a list of query's name sorted by their resulting trust value:

```
Farmer: 0.95
Packaging: 0.4
```

*4) Graphical charts*

In our prototype language implementation, we also support some essential graphic function. If a script starts with a chart declaration a graphical representation of the results will be displayed. The chart syntax is the following:

```
chart = "Charts" ":" chart_desc {chart_desc}.

chart_desc = "Pie"  [ "{" pie_option {pie_option}
"}" ] |
             "Bar"  ["{" bar_option {bar_option}
"}"] .
```

A chart declaration starts with the keyword "Charts" followed by ":" and a sequence of chart's descriptions (chart_desc). A chart_desc starts with the keyword "Pie" or

 ISBN: 978-1-61208-144-1 

"Bar" followed by an optional sequence of options. For a pie chart the possible options are:

```
pie_option =  "title" "=" string |
              "legend" "=" bool |
              "tooltips" "=" bool |
              "3d" "=" bool.
```

While for a bar chart are:

```
bar_option = "title" "=" string |
             "xlabel" "=" string |
             "ylabel" "=" string |
             "horizontal" "=" bool |
             "legend" "=" bool |
             "tooltips" "=" bool |
             "3d" "=" bool.
```

The following example will display the same charts as before but with a 3D effect and with other options enabled:

```
Charts: Pie  {title = "Pie Summary" 3d = true
legend = true tooltips = true}
        Bar  {title = "Bar Summary" 3d = true
xlabel = "Actors"
             ylabel = "Score" tooltips = true}
```

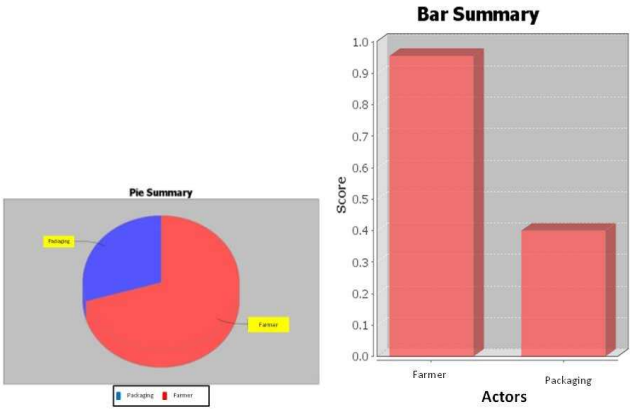


Fig3. Graphical pie and bar charts

### D. *Implementation Details*

We implemented a syntax directed interpreter that parses the script given as input and translate it in an internal data structure (a list of objects). In order to execute a script this list is further processed and finally the interpreter uses it to query the web service to collect the results and displays them.We used the ANTLR parser generator [3] to build the parser and the JFreeChart library to create and display the charts[4].

## VI. CONCLUSION

In this paper, we presented a new policy language for expressing reputation requirements in the context of supply chain scenarios. This language is now compatible with the KPI-based trust model but can be extended to the other reputation models. Using this new language one can easily specify the location of input sources (sensors) of the reputation model and configure its perception of trust. A visualization script is also integrated to the language in order to represent graphically the results. This language is in his initial phase and we are currently enhancing its capabilities. The first extension will be the support other reputation models than KPI like for example eBay/OnSale or Sporas & Histos [12] models.

## VII. ACKNOWLEDGMENT

This work is done in the context of the FP7 EU project TAS3.

## REFERENCES

[1] Gambetta, D. "Can We Trust Trust?", in Trust: Making and Breaking Cooperative Relations, Basil Blackwell, 1988
[2] Niklaus Wirth, "What can we do about the unnecessary diversity of notation for syntactic definitions?" CACM, Vol. 20, Issue 11, November 1977, pp. 822–823.
[3] http://www.antlr.org/
[4] http://www.jfree.org/jfreechart/
[5] P. Bonatti and P. Samarati, "Regulating Service Access and Information Release on the Web," 7th ACM Conference on Computer and Communications Security, Athens, Greece, November 2000.
[6] A. Herzberg, Mihaeli, Y. Mass, D. Naor, and Y. Ravid,"Access Control Meets Public Key Infrastructure, Or:Assigning Roles to Strangers," IEEE Symposium on Security and Privacy, Oakland, CA, May 2000.
[7] E. Bertino, S. Castano, and E. Ferrari, "On Specifying Security Policies for Web Documents with an XML-based Language," Sixth ACM SACMAT, Chantilly, Virginia, May 2001.
[8] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote Trust-Management System," RFC 2704,September 1999.
[9] P. Bonatti , C. Duma , D. Olmedilla , and N. Shahmehri, "An Integration of Reputation-based and Policy-based Trust Management" In proceedings of the Semantic Web Policy Workshop (in conjunction with 4th International Semantic Web Conference), Galway, Ireland, November 2005
[10] C. Bizer, R. Cyganiak, O. Maresch and T. Gauss, "TriQL.P - Trust Architecture" http://www4.wiwiss.fu-berlin.de/bizer/triqlp/
[11] K. Bohm, S. Etalle, J. den Hartog, C. Hutter., S.Trabelsi, D. Trivellato, and N. Zannone, "A Flexible Architecture for Privacy-Aware Trust Management" Journal of Theoretical and Applied Electronic Commerce Research ISSN 0718–1876 Electronic Version VOL 5 / ISSUE 2 / AUGUST 2010 / pp. 77-96
[12] Zacharia, G. and Maes, P., "Trust Management through Reputation Mechanisms", Applied Artificial Intelligence 14 (2000) pp. 881–907.

 ISBN: 978-1-61208-144-1 