OpenKnowledge

FP6-027253

Plug-in component supporting trust[1]

Juan Pane[1], Carles Sierra[2], Adrián Perreau de Pinninck[2], Pavel Shvaiko[1]

[1] University of Trento
{pane;pavel}@dit.unitn.it
[2] IIIA(CSIC) Barcelona
{adrianp;sierra}@iiia.csic.es

Report Version: final
Report Preparation Date: 21/12/07
Classification: deliverable 4.8
Contract Start Date: 1.1.2006 Duration: 36 months
Project Co-ordinator: University of Edinburgh (David Robertson)
Partners: IIIA(CSIC) Barcelona
Vrije Universiteit Amsterdam
University of Edinburgh
KMI, Open University
University of Southampton
University of Trento

[1]The originally planned title of this deliverable as from the project proposal was "Trust and reputation evaluation plug-in". However, this new title better reflects the current contents of the deliverable and needs of the project, and therefore, is used here.

Abstract

This document provides a brief documentation of the Trust component within the OpenKnowledge (OK) project. In particular, it discusses: (*i*) the Trust algorithms, (*ii*) the Trust component architecture, (*iii*) the graphical user interface, and (*iv*) a concrete case study scenario.

1 Introduction

This deliverable describes the plug-in implementation of the trust component described in Deliverable 4.5 [1]. The deliverable is divided in two major sections. Section 2 explains the decisions taken when implementing the algorithms in Deliverable 4.5. The currently implemented version is a stand-alone one. This section also explains what visual interfaces have been implemented to interact with the plug-in. Section 3 explains the working of the algorithms in a concrete negotiation scenario.

2 Implementation

2.1 Trust model implementation

The following web site `http://www.few.vu.nl/OK/wiki/` provides the OpenKnowledge (OK) client installation guidelines, while the source code is available at the project revision (subversion) control system - SVN: `http://fountain.ecs.soton.ac.uk/ok/repos/modules/trust/trunk`[2].

The Trust model has been implemented as a separate module from the Open Knowledge Kernel [2]. This decision was taken considering that the computation of Trust values requires a fair amount of computational resources, so leaving the kernel as lightweight as possible. Only those peers that require the sophisticated computation of trust because, for example, they are running *sensitive* interaction models (IM) [3] like buying and selling goods, or they run on mobile devices, can choose to use the trust module as a service.

The interface of the trust module is:

```
public interface ITrust {
//Trust interface
    public double getTrust(String peerID, Commitment commit);
}
```

[2]Authorization required, contact David Dupplaw (dpd@ecs.soton.ac.uk) for an account set up.

where the first argument `peerID` of the `getTrust` function is the identity of the peer of which we want to assess the trust about a particular yet to be signed commitment[3] `commit`. The commitment `commit` is a 4-tuple $(\beta, \alpha, \langle im, r, m, \varphi, c_m \rangle, t)$ where $\langle im, r, m, \varphi, c_m \rangle$ is the `context`. This means that β commits to α to play role r in interaction model im and, as part of playing the role r, to send a particular message m instantiated as φ, in a manner determined by the set of constraints c_m, the commitment being made at time t, see [1] for details.

Each peer α has a local history M_α of interactions with other peers βs. Each entry μ in M_α is a 4-tuple $\langle C, \varphi'', g, d \rangle$, where C is the previously defined commitment, φ'' is the observation of what actually was done by β, g is in 0,1 and can denote correct "1" or failed "0" execution of the im, and d is α's rating, this is, how happy the user was with the actual observation of the commitment.

Trust is calculated *on demand* following Algorithm 1 in this section (imported from Deliverable 4.5 [1] for the purpose of completeness of the deliverable). The main steps in the algorithm are:

1. *(lines 1 to 6)* build the *Focus* set, this is, the set of all terms φ' from the local Ontology O that are similar enough (over a defined threshold ν) to the current commitment φ. Where the similarity function is defined as:

$$\text{Sim}(\varphi, \varphi') = \begin{cases} 1 & \text{if } \varphi = \varphi' \\ e^{-\kappa_1 l} \cdot \frac{e^{\kappa_2 h} - e^{-\kappa_2 h}}{e^{\kappa_2 h} + e^{-\kappa_2 h}} & \text{otherwise} \end{cases} \tag{1}$$

   where l is the length (i.e; number of hops) of the shortest path between the concepts, h is the depth of the deepest concept subsuming both concepts, and κ_1 and κ_2 are parameters balancing the contribution of the shortest path length and the depth, respectively [4].

2. *(lines 7 to 9)* calculate the decay distribution D. Where D is the default probability distribution when there are no relevant experiences.

3. *(lines 10 to 13)* calculate H_β, which is the set of commitments that the peer β has successfully fulfilled, according to the user rating d of the observation φ''.

4. *(lines 14 to 21)* measure the ability of β of doing φ, considering the set H_β of what he has successfully done in the past.

[3]We follow Deliverable 4.5 in that a commitment is any ontological expression that can be *observable*. Not all ontological expressions are observable. In that sense, a concept (element in the vocabulary, e.g. Cow) in the ontology can not be the subject of a commitment, but a contract or a proposition whose truth can be verified can be (e.g. a cow is a mammal).

Algorithm 1 function Trust(α, β, $\langle im, r, m, \varphi, c_m \rangle$)

```
Require: $O$ {the peer finite local ontology}
Require: $\kappa_1, \kappa_2 : Real$ [Default 1.0] {parameters of the similarity function}
Require: $\eta : [0,1]$ [Default 0.8] {min. sem. similarity in the computation}
Require: $\nu : [0,1]$ [Default 0.95] {decay parameter}
Require: $\zeta : [0,1]$ [Default 0.7] {minimum satisfaction}
Require: $\omega : [0,1]$ [Default 0.1] {minimum overall similarity}
Require: $\lambda : [0,1]$ [Default 0.1] {minimum update relevance}
Require: $n_{ex} : \mathbb{N}$ [Default 6] {number of experiences to be highly confident}
Require: $Prefer : Terms(O) \times \{\varphi\} \to [0,1]$ [Default $Prefer(x, y) = if\ x = y\ then\ 1\ else\ 0$]
    {a prob. dist. for preference of terms over $\varphi$ represented as a vector}
Require: $M_\alpha \subseteq M$ {$\alpha$'s log of experiences sorted by time}
Ensure: $Trust(\alpha, \beta, \langle IM, r, m, \varphi, c_m \rangle) \in [0,1]$
 1: $Focus \leftarrow \emptyset$
 2: for all $\varphi'$ in Terms(O) do
 3:     if $Sim_\tau(\varphi', \varphi, \kappa_1, \kappa_2) \geq \eta$ then
 4:         $Focus \leftarrow Focus \cup \{\varphi'\}$
 5:     end if
 6: end for{we assume finiteness of Terms(O)}
 7: for all $\varphi'$ in Focus do
 8:     $D(\varphi' \mid \varphi) \leftarrow 1/size(Focus)$
 9: end for
10: $H_\beta = \emptyset$
11: for all $\mu = \langle PC, \varphi'', 1, d \rangle$in $M_\alpha$
        and $\langle Commit(\beta, \_, \langle im, r, m, \varphi', c' \rangle, t) \rangle \in PC$ with $d \geq \zeta$ do
12:     $H_\beta = c' \cup H_\beta$
13: end for
14: $Match \leftarrow 1$
15: for all $\phi \in c_m$ do
16:     $MAX \leftarrow 0$
17:     for all $\phi' \in H_\beta$ do
18:         $MAX \leftarrow \max\{MAX, Sim(\phi, \phi')\}$
19:     end for
20:     $Match \leftarrow MAX \cdot Match$
21: end for
22: $t \leftarrow 0$; $P^t = D$
23: for all $\mu = \langle PC, \varphi'', 1, d \rangle$ in $M_\alpha$
        and $\langle Commit(\beta, \_, \langle im, r, m, \varphi_c, c_m \rangle, t) \rangle \in PC$ do
24:     if $Sim(\varphi_c, \varphi) \geq \omega$ then
25:         for all $\varphi'$ in Focus do
26:             $S \leftarrow (1- \mid Sim(\varphi'', \varphi_c) - Sim(\varphi', \varphi) \mid) \cdot Sim(\varphi_c, \varphi)$
27:             if $S \geq \lambda$ then
28:                 $Q(\varphi' \mid \varphi) \leftarrow Match \cdot (P^t(\varphi'|\varphi) + 1/(n_{ex} \cdot |Q|) \cdot S \cdot (1 - P^t(\varphi'|\varphi)))$ {Constraints
                    to satisfy}
29:             end if
30:         end for
31:         $P^{t+1} \leftarrow (1-\nu)D + \nu \cdot MRE(P^t, Q)$ {MRE is the Min. relative entropy from $P^t$
            satisfying $Q$}
32:     end if
33:     $t \leftarrow t + 1$
34: end for
35: $T \leftarrow 0$
36: for all $\varphi'$ in Focus do
37:     $T \leftarrow T + Prefer(\varphi', \varphi) * P^{t-1}(\varphi' \mid \varphi)$
38: end for
39: return $T$
```

5. *(lines 22 to 34)* update the probability distribution P, considering the entries in the local history M_α that are similar enough (over a given threshold ω), to the current commitment.

6. *(lines 35 to 39)* calculate the final trust value considering the probability of the terms in the set $Focus$.

2.1.1 Packages in the Trust Module

The implementation of the trust module is divided in 3 main packages:

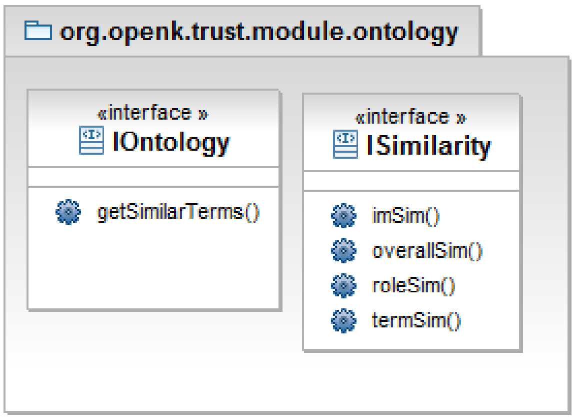


Figure 1: Class diagram for Ontology package.

1. `org.openk.trust.module.ontology`: This package interfaces with the local representation of the ontology in the peer. Figure 2.1.1 shows the class diagram of this package. The interfaces that should be implemented are:

   - `IOntology`: which is required to obtain all the similar terms needed by Algorithm 1 and defined by the `getSimilarterms()` function in the interface;
   - `ISimilarity`: defines the similarity functions also needed by Algorithm 1. `imSim()` computes the similarity between interaction models, `roleSim()` computes the similarity between roles, `termSim()` computes the similarity between terms that a peer can commit to, and `overallSim()` defines an aggregation function of all the other similarity functions, in order to obtain a unique overall similarity score, for more details see Deliverable 4.5 [1].

2. `org.openk.trust.module.history`: This package interfaces with the local representation of history M_α of the peer. Each peer is free to

implement its local history as preferred, and then implement the interfaces in this package to interact with the so defined local representation. Figure 2 shows the class diagram of this package; the classes in this package are:

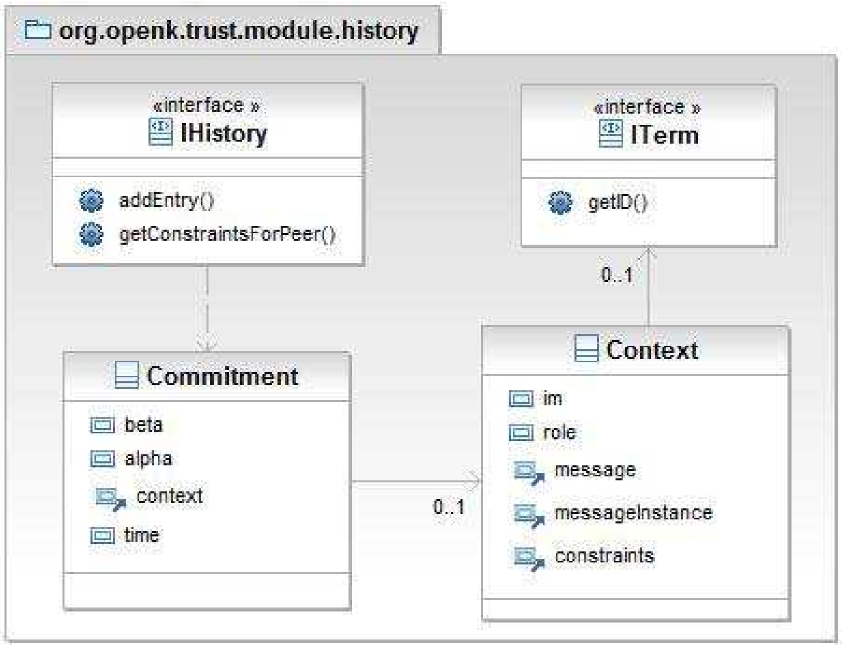


Figure 2: Class diagram for History package.

- `IHistory`: is the bridge between the local implementation of the history of the peer and the Trust module, it should at least provide functionalities for inserting (`addEntry()`) and retrieving entries (`getConstraintsForPeer()`) from M_α.
- `ITerm`: represents the capabilities or constraints that one peer can commit to, and therefore, can later be observed. The `getID()` function obtains the identifier of the Term.
- `Commitment`: is the commitment C defined in Section 2.1. This is, `beta` commits to `alpha` to fulfill some action defined in `context` in time `t`.
- `Context`: is the `context` a particular peer commits to fulfill as defined in Section 2.1. This means that a peer commits to another peer to play `role` in interaction model `im` and, as part of playing the role, to send a particular `message` instantiated as `messageinstance`, in a manner determined by the set of `constraints`, see [1] for details.

3. `org.openk.trust.module.statistical`: This package defines the statistical functions that are the base of the Trust calculation. Figure 2

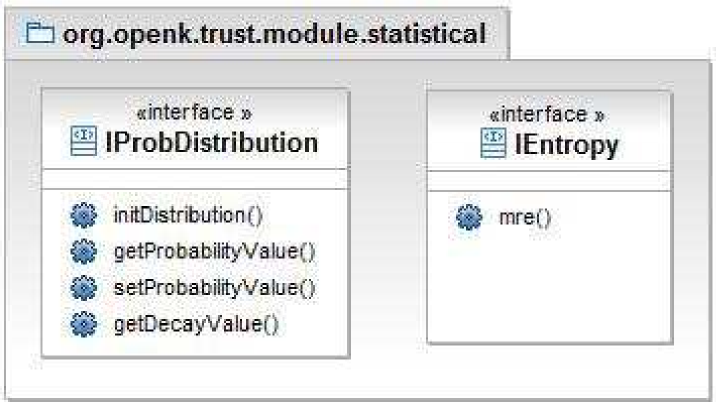


Figure 3: Class diagram for package Statistical package.

shows the class diagram of this package. The interfaces in this package are:

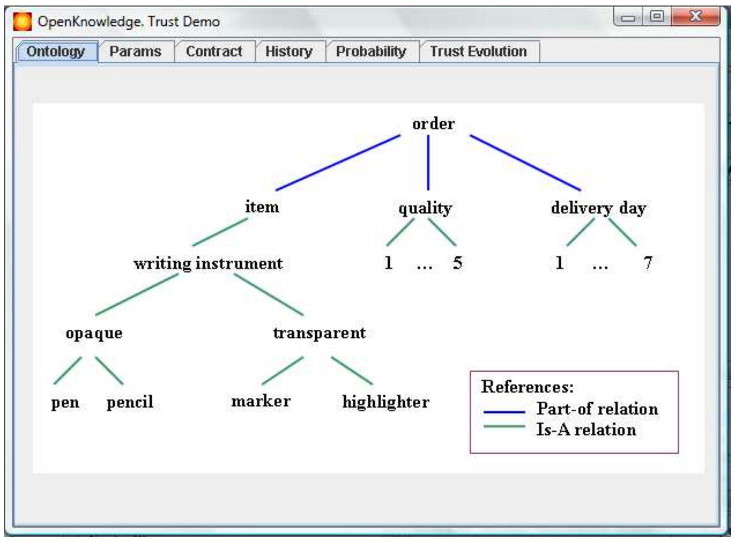


Figure 4: Ontology of a local peer.

- `IProDistribution`: represents the probability distribution P that changes during time. It manages the conditional probabilities defined in Deliverable 4.5 [1] and the decay D, see Algorithm 1. The defined functions are: `iniDistribution()` that initializes the probability and decay distribution, `getProbabilityValue()` returns the probability for a given term, `setProbabilityValue()` sets a new probability for a given term, and `getDecayValue()` returns the decay value for a given term in the probability distribution.

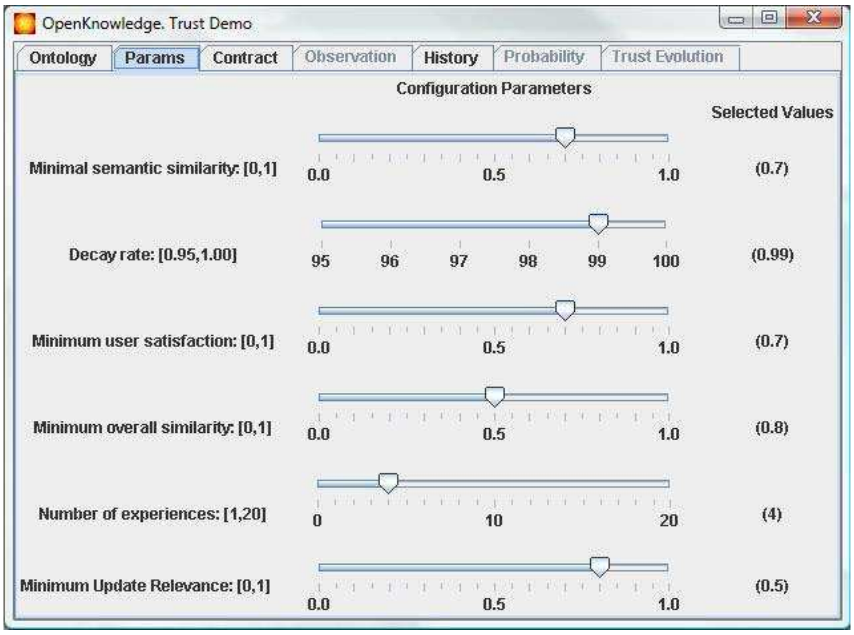
- `IEntropy`: defines the update function `mre()` (minimum relative entropy) for the probability distribution P^t used in Algorithm 1.

2.2 Visual interfaces

The implementation of the Trust module includes a Graphical User Interface (GUI) to show the capabilities and the different components of the module. The source code of the Visual interface is located at org.openk.trust.visual within the SVN project. The GUI is composed of several tabs, which intend to show the parameters, behaviour and results. The tabs are:

1. **Ontology**: shows a graphical representation of the local ontology of the current peer (Figure 4). It defines possible terms that can be used in the interaction models (IM).

2. **Parameters**: shows the parameters and the default values of the Trust module, which can be customized in order to change its behaviour (Figure 5). The configuration parameters are:



Configuration Parameters

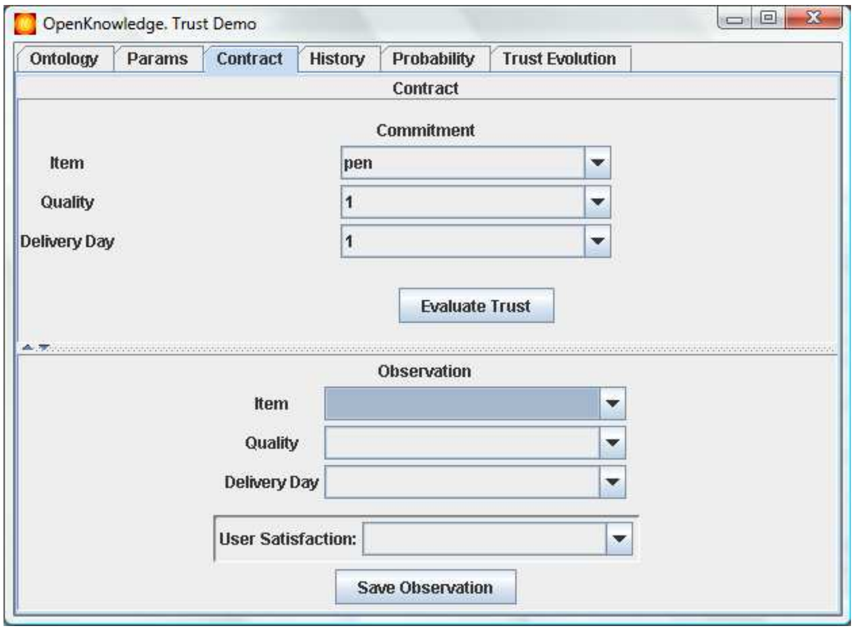
		Selected Values
Minimal semantic similarity: [0,1]	0.0 0.5 1.0	(0.7)
Decay rate: [0.95,1.00]	95 96 97 98 99 100	(0.99)
Minimum user satisfaction: [0,1]	0.0 0.5 1.0	(0.7)
Minimum overall similarity: [0,1]	0.0 0.5 1.0	(0.8)
Number of experiences: [1,20]	0 10 20	(4)
Minimum Update Relevance: [0,1]	0.0 0.5 1.0	(0.5)

Figure 5: Parameters of the Trust Module.

- ν: *minimum semantic similarity* required between a term of the ontology and the commitment under analysis to consider the term to be a part of the *Focus*.
- v: (*decay rate*) sets how slow, $1 - v$, P goes back to the default probability distribution D. This is, if no observation happens during a unit interval of time, the new probability distribution is

calculated as: $P^{t+1} = (1 - \upsilon) \cdot D + \upsilon \cdot P^t$, see Deliverable 4.5 [1] for details.

- *minimum user satisfaction* required to consider that an entry in the history, for a given peer β, was good enough, and therefore should be considered in order to assess the capabilities of β. The peer will be considered capable of fulfilling those contracts close enough to commitments in the past with a degree of satisfaction over the value of this parameter.
- *minimum overall similarity* between commitments in the history M_α, and the current commitment, to consider that one commitment in M_α is meaningful enough to affect the assessment of P.
- N_{ex}: *number of experiences* with a peer β about a certain commitment φ, necessary to fully consider the influence of a new experience to assess P.
- λ: *minimum update relevance*, i.e; minimal semantic similarity between the current commitment, the expected and observed terms in one entry of the history, and one term in the focus, to consider that the considered term in the focus should change the probability distribution.

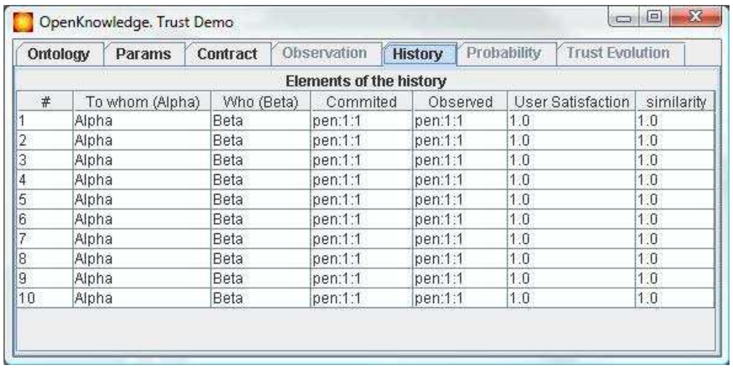


Item: pen
Quality: 1
Delivery Day: 1

Item:
Quality:
Delivery Day:
User Satisfaction:

Figure 6: A contract to be tested.

3. Contract: shows the commitment φ that a peer β has signed, or is about to sign, and for which assessment we can use the Trust module. This tab (Figure 6) also lets the user define the observation of the



Elements of the history

#	To whom (Alpha)	Who (Beta)	Commited	Observed	User Satisfaction	similarity
1	Alpha	Beta	pen:1:1	pen:1:1	1.0	1.0
2	Alpha	Beta	pen:1:1	pen:1:1	1.0	1.0
3	Alpha	Beta	pen:1:1	pen:1:1	1.0	1.0
4	Alpha	Beta	pen:1:1	pen:1:1	1.0	1.0
5	Alpha	Beta	pen:1:1	pen:1:1	1.0	1.0
6	Alpha	Beta	pen:1:1	pen:1:1	1.0	1.0
7	Alpha	Beta	pen:1:1	pen:1:1	1.0	1.0
8	Alpha	Beta	pen:1:1	pen:1:1	1.0	1.0
9	Alpha	Beta	pen:1:1	pen:1:1	1.0	1.0
10	Alpha	Beta	pen:1:1	pen:1:1	1.0	1.0

Figure 7: History of the observed contracts for the current peer.

commitment φ'. Once the user defines φ', the tuple $\langle \varphi, \varphi' \rangle$ can be stored in the history M_α of the observed commitments for β.

4. History: shows a subset of attributes of each entry in the history M_α, mainly the index or number of entry, the current peer α, the committing peer β, the committed term φ, the observed term φ', the user satisfaction d, and the similarity between the committed and observed terms.

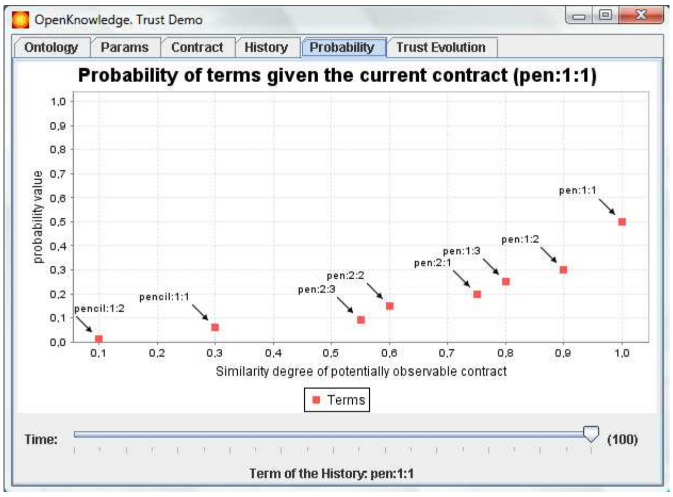


Figure 8: Evolution in time of the probability distribution.

5. Probability Distribution (P): shows a chart of the Probability of the terms in the *Focus* (Figure 8). The terms are arranged considering their similarity value with respect to φ. At each time step P changes considering the commitment and observation of the corresponding entry in M_α. The time line represents the evolution of P, by changing its value on the sliding bar, it is possible to observe the corresponding P for a particular moment in time.

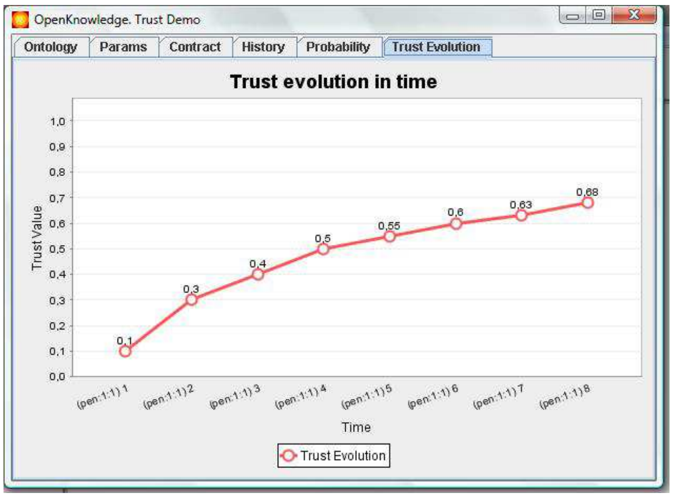


Figure 9: Evolution in time of the trust for the current commitment.

6. Trust Evolution: shows a chart with the evolution of the Trust, for the given φ (written in the contract tab) in time (Figure 9). The time evolution is defined by the entries in M_α as before. The last point in the chart represents the current Trust value for the given φ.

3 A negotiation scenario

This section illustrates the monitoring of the trust plug-in in a concrete buying/selling scenario. The lifecycle of an interaction is described from the point of view of a particular peer, named $peer_1$.

1. $peer_1$'s user decides if he/she wishes to play the role of a buyer in a *buying-selling* interaction. He/she therefore inputs the following query into the discovery service (DS):

   Buying selling interaction

2. The DS returns a list of potentially suitable IMs, ranked according to various criteria such as (for details see Deliverable 4.5 [1]):

   - How well the query of the user matches the keyword description of the IM.
   - Some evaluation of how popular the IM is.
   - How many peers are already subscribed to the IM.

   This process would naturally return a ranked list of IMs to $peer_1$. Let us assume that the top ranked IM is the one shown in Figure 10.

```
$a(buyer, B) ::$
    $ask(C, Ma, Mo, Y) \Rightarrow a(seller, S) \leftarrow$
        $need(C : car, Ma : make, Mo : model, Y : year)\ then$
    $price(C, Ma, Mo, Y, P) \Leftarrow a(seller, S)\ then$
    $commit(B, S, ccIM, payer, pay, price = P)$
        $\leftarrow buy(C, Ma, Mo, Y, P) \Rightarrow a(seller, S) \leftarrow afford(C, P : Price)$
    $owns(C, Ma, Mo, Y) \leftarrow sold(C, P) \Leftarrow a(seller, S)$

$a(seller, S) ::$
    $ask(C, Ma, Mo, Y) \Leftarrow a(buyer, B)\ then$
    $price(C, Ma, Mo, Y, P) \Rightarrow a(buyer, B) \leftarrow instock(C, Ma, Mo, Y, P)$
    $buy(C, Ma, Mo, Y, P) \Leftarrow a(buyer, B)\ then$
    $commit(S, B, crrIM, sndr, snd, car = C; make = Ma; mod = Mo; yr = Y)$
        $\leftarrow sold(C, P) \Rightarrow a(buyer, B)$
```

Figure 10: Buying-selling IM.

3. Given a concrete need that materialises into a desired contract φ that $peer_1$ would like to see signed with a seller. It looks for potential partners to play the selling role in the interaction model and then ranks them according to their relative trust with respect to the desired φ. Let us call the selected one, $peer_2$.

4. $peer_1$ runs the interaction model with $peer_2$ to try and reach φ. If the agreement is feasible the contract is signed and a commitment is made to run another interaction model later on, in this example: *execute*, in order to execute the contract.

5. $peer_1$ and $peer_2$ run the *execute* interaction model and then the actual φ' is observed.

The trust algorithm is used in step 3 to select the best potential seller. We illustrate the previous behaviour on a scenario where the orders, o_i, consist of: $o_i = (y_i, q_i, d_i)$, where: y_i is the type of item ordered (pens, pencils, markers, highlighters), q_i is the quality of the item (a number in [1..5]), and d_i is the number of days to delivery the product (a number in [1..7]). Finally, Figure 4 shows graphical representation of the peer ontology.

4 Conclusions and future work

This document has explained the implementation details and the application in a concrete scenario of the trust plug-in developed within the OpenKnowledge project. The current implementation is stand-alone in order to facilitate its validation. We developed a graphical user interface to facilitate the tuning of the trust parameters by the user of the plug-in as well as to help in the monitoring of the component execution. The requirements of the algorithms are easy to be satisfied by the kernel (including historical data and parameter values). Thus, the integration with the kernel is among the next steps. Also, the use of the Trust plug-in in the two major case studies is currently being studied.

Two reputation models are currently being studied as well: a centralized and totally distributed. The distributed model is based in the current Trust model, where gossip can occur in order to learn the behaviour of a particular peer we never had interacted with before. The centralized model just keeps track of user behaviour and can be consulted when needed. The results of this use will be reported in subsequent deliverables.

References

[1] Giunchiglia, F., Sierra, C., McNeil, F., Osman, N., Siebes, R.: *OpenKnowledge Deliverable 4.5: Good Enough Answer Algorithms.* `http://www.cisa.informatics.ed.ac.uk/OK/Deliverables/D4.5.pdf` (2006)

[2] Perreau de Pinninck, A., Dupplaw, D., Kotoulas, S., Siebes, R.: The openknowledge kernel. International Journal of Applied Mathematics and Computer Sciences (IJAMCS) **4** (2007) 162–167

[3] Joseph, S., de Pinninck, A.P., Robertson, D., Sierra, C., Walton, C.: *OpenKnowledge Deliverable 1.1: Interaction Model Language Definition.* `http://www.cisa.informatics.ed.ac.uk/OK/Deliverables/D1.1.pdf` (2006)

[4] Li, Y., Bandar, Z.A., McLean, D.: An approach for measuring semantic similarity between words using multiple information sources. IEEE Transactions on Knowledge and Data Engineering **15** (2003) 871 – 882