
UNIVERSITY
OF TRENTO

**DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY**

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it


# EVALUATING GOOD ANSWERS IN OPEN KNOWLEDGE


Fausto Giunchiglia, Fiona McNeill, Mikalai Yatskevich,
Carles Sierra and Jordi Sabater



January 2007

Technical Report # DIT-07-030

# Evaluating Good Answers in Open Knowledge

Fausto Giunchiglia, Fiona McNeill, Mikalai Yatskevich,
Carles Sierra and Jordi Sabater

January 26, 2007

## 1 Good Enough Matching in OpenKnowledge

The nature of the OpenKnowledge system makes insisting on perfect matching an unfeasibly stringent requirement. If we wish to facilitate the interactions of an unlimited number of disparate peers without prior agreement, it will often be the case that peers can *more or less* perform a given role, but will find that the role description does not fit perfectly with its own description of its abilities. To make the OpenKnowledge system usable, we need to be able to look for peers that can exactly perform a given role but, if, as will often happen, this search fails, we need to find peers that can *approximately* perform such roles, and we need some way of estimating how good this approximation is.

When a peer wishes to initiate an interaction, there are two aspects of good enough matching that must be considered: firstly, how can the peer find an interaction model that is a good enough description of the interaction it wishes to initiate; secondly, how can it determine which peers would be most suitable for taking on the roles in that interaction. If we consider that the motivation for a peer initiating an interaction would normally be that it wishes to play a role (or possibly roles) in that interaction, we can reduce this to a single problem: that of determining whether a given peer can perform a given role to an appropriately high level of approximation. Thus when a peer wishes to initiate an interaction, the way in which it can evaluate potentially suitable interaction models is by considering which role it wishes to play and then evaluating its own ability to perform that role. This involves matching the constraints of that interaction, which tell the peer to what degree it will successfully perform that role - how well do the *constraints* match the peer's *abilities* - and the consequences tell it how much it wants to perform the role - how well do the they match its *goals*. If the initiating peer receives poor scores for either of these matchings tasks, it is unlikely to choose this particular IM. If it receives good scores for both, it will then need to ascertain to what degree the other roles will be suitably fulfilled.

Note that there are two different requirements for a peer performing a role. The first is whether the peer can match the constraints to its own abilities and can therefore have the potential to fulfil them. The second is whether, for the given instantiations of those constraints, the peer is actually able to satisfy them. For example, Figure 1 shows an IM which includes a constraint $accept(Pr, M, N_2)$, where $Pr$ refers to *price*, $M$ to the make of wine and $N_2$ to the number of bottles required. Before interaction, a peer must

determine whether it has an ability that matches this: that is, does it have an ability with a name similar to *accept* and with attributes that approximately match *price*, *make* and *No_bottles*. However, prior to interaction, the peer cannot tell how $Pr$, $M$ and $N_2$ will be instantiated: it can only tell the *type* of thing they will be instantiated as. So the match between abilities may be very good, or even perfect, but during the interaction the peer will still find itself unable to satisfy the constraint.

The first issue must be settled before the interaction commences: if a peer cannot interpret the constraints, it should not be playing that role as the interaction commences. This issue must be settled through the matching techniques described in this deliverable before the interaction commences. The second issue, however, cannot always be settled before interaction. In some situations, constraints will be fully instantiated in the IM and, in such cases, a peer's ability to satisfy them will be a crucial part of the pre-interaction matching. However, if, as in the above example, these values are left uninstantiated until interaction commences, the peer can only determine how well it can interpret the ability described by the constraint, and cannot tell whether it can satisfy the particular instantiation until interaction commences: a well skilled peer may therefore fail to perform its role successfully. There is no way around this problem, but trust scores for both the IM and the peer may indicate how likely they are to be able to satisfy the necessary constraints.

## Lifecycle of Good Enough Matching

We distinguish two kinds of good enough matching: 1) global good enough (GGEA), which determines which IM a peer will choose for an interaction and which other peers will fulfil the roles of the interaction; 2) local good enough (LGEA), which is a subprocess of GGEA and determines how well each constraint on a role can be fulfilled by a given peer. Here, we give the lifecycle of GGEA and explain how LGEA fits into this.

- **STEP 1:** The initiating peer must first locate an appropriate interaction model that results in the goals it wishes to satisfy. This IM can either be already known to it can be found via the discovery service. A discovery service (described in other deliverables) returns interaction models satisfying the request, based on the requirements input by the user and the natural language or keyword descriptions of IMs.

- **STEP 2:** Once these potentially suitable interaction models have been located, semantic matching is used to generate an LGEA score for the role the intiating peer wishes to play. This will allow it to determine whether the highest ranked IM (or, failing that, any other) is appropriate for the task at hand.

- **STEP 3** The discovery service will find potentially suitable peers through matching the role description in the interaction model with the descriptions that peers give of their capabilities;

- **STEP 4** Potentially suitable peers are contacted and, if they are available and willing, will be sent a copy of the interaction model;

$a(customer, C)$ ::

$$\begin{array}{rcll} request(wine(P_1, P_2, R, C, N) & \Rightarrow & a(wine\_merchant, W) & \leftarrow \ choose\_wine(P_1, P_2, R, C, N) \\ recommendation(Pr, M) & \Leftarrow & a(wine\_merchant, W) & then \\ buy(M, N_2) & \Rightarrow & a(wine\_merchant, W) & \leftarrow \ accept(Pr, M, N_2) \ \wedge N_2 \leq N \\ owns(M, N_2) \leftarrow sold(M, N_2) & \Leftarrow & a(wine\_merchant, W) & \end{array}$$

$a(wine\_merchant, W)$ ::

$$\begin{array}{rcll} request(wine(P_1, P_2, R, C, N) & \Leftarrow & a(customer, C) & then \\ recommendation(Pr, M) & \Rightarrow & a(customer, C) & \leftarrow \ recommend(R, C, M, Pr) \ \wedge Pr > P_1, P_2 \\ & & & \wedge \ in\_stock(X, M) \ \wedge X \geq N \ \wedge \ price(M, Pr) \\ buy(M, N_2) & \Leftarrow & a(customer, C) & then \\ in\_stock(Y, M) \ \leftarrow sold(M, N_2) & \Rightarrow & a(customer, C) & \\ \wedge \ Y = X - N & & & \end{array}$$

Figure 1: Wine buying interaction model

- **STEP 5** Each peer will perform semantic matching to interpret the requirements and effects of the interaction. If they are happy with and able to fulfil the constraints, they will return a score indicating the degree to which they are able to fulfil the role, generated from their LGEA score.
- **STEP 6** The suitable peers are ranked according to an estimate of how well they will perform the role. This score is generated through a combination of their LGEA scores and trust scores associated with them[1]. The highest ranked peers are approached to play the role.

# 2 Motivating Scenario

**Example 1** *(Wine buying scenario) James is a wine merchant and regularly buys online from known distributors with whom he has an established relationship. In addition to this, he is keen to find good deals and special offers from unknown distributors. He sells his goods to customers in his shop and online and has different methods of doing this: for example, customers may want to request a particular kind of wine or may wish to provide some constraints such as price, colour, etc., and would accept recommendations on that basis. James is keen to advertise himself as a wine seller to as many potential customers as possible and to be accommodating in his interactions with them so as to encourage custom. He also wishes to actively search for suppliers to ensure that he is always able to fulfil his orders and replenish his stock and to ensure he keeps his costs to a minimum. Figure 1 shows one of the IMs that James might use.*

*It describes, for each of the two roles in the interaction (customer and wine_merchant), the messages that are passed between them during this interaction. Since there are only two roles in this interaction, the message passing described in them is symmetric. In addition, the constraints of the message passing are described.*

[1]The issues of trust are not central to this deliverable; we discuss this briefly later in this deliverable but deliverables 4.7 gives further details.

*To the left-hand side of the message are the constraints: for example, choose_wine and recommend. These can only be attached to messages to be sent as there are never constraints on receiving messages. The peer playing the customer role must first satisfy the choose_wine constraint, during which it discovers the pertinent information that is to be sent in the message. In order to satisfy the constraint, it must interpret it by mapping it to a constraint which it knows how to satisfy, and then run its constraint satisfaction program on it. It then waits to receive a recommendation from the wine_merchant, together with a specific price. It must then attempt to satisfy the accept*$(Pr, M, N_2)$ *predicate, which determines whether the response is acceptable. If the constraint is satisfied, it then waits to receive a message confirming the sale.*

*Note that in order to understand the interaction model, the peer only need interpret the constraints. The predicates in the messages are just place holders that allow the identification of the particular message; sensible names for these predicates are used to make things easier for the interaction model designer and for any users who wish to look at it but do not need to be interpreted by the peers.*

# 3 Local Good Enough Answers

In order to determine how well it can perform a role, a peer generates local good enough scores for every constraint in that role by matching them to its own abilities. A score for the role is then generated through averaging these individual scores.

Constraints are written in first-order logic and we assume that peer abilities are also in this format. The problem therefore becomes that of matching two first-order terms.

## 3.1 Contextual LCC

As described above, each peer must be able to interpret the constraints on the activities in the role it desires to play before it can either assert that it is able to play the role or commence the performance. If there is no prior agreement on what interactions will take place or on the interaction models that describe them and there is no central ontology or definitive method of knowledge representation, then there is no way of ensuring that a peer's way of representing the constraints it can satisfy will match the way in which these constraints are represented in the interaction model. Peers cannot be expected to have a uniform vocabulary, nor can they be expected to represent complex concepts in the same way.

Interaction Models are written in Ambient LCC, described in other deliverables. The syntax of Ambient LCC is described in Figure 3.1. In Ambient LCC, the matching of a constraint that a peer knows how to satisfy (or at least, knows how to determine whether it can satisfy) and a constraint which it must satisfy in a particular interaction model is done through unification; if the representation and vocabulary used is not the same, matching will fail, and thus it is only applicable in a closed domain. In an open domain, peers will find themselves unable to understand constraints even on roles that they have the ability to play.

We propose to extend Ambient LCC to facilitate semantic matching on the constraints

by adding contextual information to interaction models. It is possible to perform semantic matching on constraints even without any contextual information; however, this can make it difficult to find good semantic matches.

Another aspect missing from the Ambient LCC definition that is important in matching is type information. If type information can be attached to the arguments of constraints then it is much easier to determine how well they map to known constraints. This is especially true since, when the matching is being done on constraints, before the interaction commences, it is likely that many of the arguments of constraints will be variables and thus, without this type information, it may be impossible to unambiguously determine their meaning.

The changes to the definition that are required in order to replace resolution with semantic matching are minimal. Figure 3.1 details these definitions: the syntax of Contextual LCC is identical to the syntax of Ambient LCC with these specified changes. The differences are the way in which *Constants* and *Variables* are defined and the addition of four new objects: *Class*, *Context*, *Value* and *Holder*. Constants and Variables both become complex objects which contain, respectively, a *value* or a *holder* (whose definitions are nearly identical to the original definitions of *constant* and *variable*), and information pertinent to that value or holder: type information (which we refer to here as *class* to avoid confusion with the *type* definition that refers to role types) and contextual information. *Class* information is a value or a holder. *Context* information is also a value or holder and can consist of anything that defines the context in a term could be used can provide the context, for example, the classification of the interaction model [Giunchiglia et al., 2004], Web directories [Avesani et al., 2005] or even user preferences [periklis, 2006]. *Value* is defined identically to Ambient LCC *constant*; *Holder* is similar to *Variable* but, in addition to an upper case character sequence or number being an acceptable input, white space is also acceptable and can be used in the case that an interaction model designer does not know the pertinent information or does not wish to input it. The design of this syntax means that it is easy to add extra attributes to constant and variable definitions should we wish to do so.

Contextual information can be derived from the context in which the interaction model is used. For example, users might store information about their interaction models in a classification such as the one illustrated in Figure 4. The interaction model shown in Figure 1 would be the one stored under *Wine-Selling-Recommendation*. This contextual information can be attached to any object in the interaction model and may prove useful in its semantic interpretation.

**Example 2** *(Marking up the Interaction Model) Figure 5 shows part of the interaction model of Figure 1 with the additional mark-up included. Since the interaction model with all this information explicit is rather unwieldy, we include only a small section.*

*Both the type and the context can be extremely useful in interpreting the semantics. For example, the wine context indicates that the variable C, of type colour, should be instantiated by red or white; and that the variable M, of type make should be instantiated by some make of wine and not, for example, a make of car. These restrictions would not be apparent to a peer without context but becomes derivable once context is included.*

*Note that predicates, which, in the Ambient LCC syntax, are defined as Constants,*

$$
\begin{array}{rcl}
Model & := & Clause, ... \\
Clause & := & Role :: Def \\
Role & := & a(Type, Id) \\
Def & := & Role \mid Message \mid Def\ then\ Def \mid Def\ or\ Def \\
Message & := & M \Rightarrow Role \mid M \Rightarrow Role \leftarrow C \mid \\
 & & \quad M \Leftarrow Role \mid C \leftarrow M \Leftarrow Role \\
C & := & Constant \mid P(Term, ...) \mid \neg C \mid C \wedge C \mid C \vee C \\
Type & := & Term \\
Id & := & Constant \mid Variable \\
M & := & Term \\
Term & := & Constant \mid Variable \mid P(Term, ...) \\
P & := & Constant \\
Constant & := & \text{lower case character sequence or number} \\
Variable & := & \text{upper case character sequence or number}
\end{array}
$$

Figure 2: Ambient LCC syntax

$$
\begin{array}{rcl}
Constant & := & Context : (Value, Class) \\
Variable & := & Context : (Holder, Class) \\
Class & := & Value \mid Holder \\
Context & := & Value \mid Holder \\
Value & := & \text{lower case character sequence or number} \\
Holder & := & \text{upper case character sequence or number} \\
 & & \text{or white space}
\end{array}
$$

Figure 3: Alterations to Ambient LCC syntax required by Contextual LCC

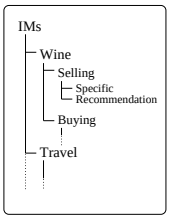



Figure 4: James's IM classification

$$
\begin{array}{l}
a(customer, C) :: \\
\quad request(wine(P_1, P_2, R, C, D, N) \quad \Rightarrow \quad a(wine\_merchant, W) \quad \leftarrow \quad (Cxt : (choose\_wine, \ ) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad Cxt : (P_1, maxium\_price), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad Cxt : (P_2, minimum\_price), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad Cxt : (R, region), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad Cxt : (C, colour), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad Cxt : (N, number\_of\_bottles)) \\
\quad Cxt \ = \ (wine\text{-}selling\text{-}recommendation)
\end{array}
$$

Figure 5: Semantic markup of interaction model

*have the same structure as any other object:* $Context : (Value, Class)$*. It may be felt that type information is inappropriate in describing predicates and, if so, this attribute will be instantiated by a white space, as occurs in the type attribute of choose_wine in Figure 5. However, if the interaction model designer or user wishes to give type information to predicates, he is at liberty to do this.*

## 3.2 Semantic Structure Matching

In this section, we discuss how we can use this contextual information as input to a structure matching process to derive values of how well two constraints (or a constraint and an ability) are matched.

### 3.2.1 Tree edit distance

In its traditional formulation, the tree edit distance problem considers three operations: (a) vertex removal, (b) vertex insertion, and (c) vertex replacement [Tai, 1979]. To each of these operations, a cost is assigned. The solution of this problem consists of determining the minimal set of operations (i.e., the one with the minimum cost) to transform one tree into another. Another equivalent (and possibly more intuitive) formulation of this problem is to discover a (proper) mapping with minimum cost between the two trees. The concept of (proper) mapping (introduced in [Tai, 1979]) is defined next.

**Definition 1** *Let* $T_x$ *be a tree and let* $T_x[i]$ *be the i-st vertex of tree* $T_x$ *in a preorder walk of the tree. A (proper) mapping between a tree* $T_1$ *of size* $n_1$ *and a tree* $T_2$ *of size* $n_2$ *is a set* $M$ *of ordered pairs* $(i, j)$*, satisfying the following conditions for all* $(i_1, j_1), (i_2, j_2) \in M$*:*

1. $i_1 = i_2$ *iff* $j_1 = j_2$;

2. $T_1[i_1]$ *is on the left of* $T_1[i_2]$ *iff* $T_2[j_1]$ *is on the left of* $T_2[j_2]$;

3. $T_1[i_1]$ *is an ancestor of* $T_1[i_2]$ *iff* $T_2[j_1]$ *is an ancestor of* $T_2[j_2]$.

In Definition 1, the first condition establishes that each vertex can appear no more than once in a mapping, the second enforces order preservation between sibling nodes and the third enforces the hierarchical relation between the nodes in the trees.

As we have already mentioned, estimating a tree edit distance is equivalent to finding the minimum cost mapping. Let $M$ be a mapping between tree $T_1$ and tree $T_2$, let $S$ be a subset of pairs $(i; j) \in M$ with distinct labels, let $D$ be the set of nodes in $T_1$ that do not occur in any $(i; j) \in M$ and let $I$ be the set of nodes in $T_2$ that do not occur in any $(i; j) \in M$. The mapping cost is given by

$$c = Sp + Iq + Dr \tag{1}$$

where $p$, $q$ and $r$ are the costs assigned to the replacement, insertion, and removal operations, respectively. It is common to associate a unit cost to all operations; however, specific applications may require the assignment of distinct costs to each type of operation.

The tree edit distance problem is a difficult one, and several algorithms, with different tradeoffs, have been recently proposed, but all formulations have complexities above quadratic [Chen, 2001]. Furthermore, it has been proved that if the trees are not ordered, the problem is NP-complete [Zhang et al., 1992]. The first algorithm for the mapping problem was presented in [Tai, 1979] and its complexity is $O(n_1 n_2 h_1 h_2)$, where $n_1$ and $n_2$ are the sizes of the trees and $h_1$ and $h_2$ are their heights. This is a dynamic programming algorithm that recursively calculates the edit distance between the strings formed by the sets of child vertices of each internal vertex in the tree. In [Wang et al., 1998], a new algorithm was presented with cost $O(d_2 n_1 n_2 min(h_1; l_1) min(h_2; l_2))$, where d is the edit distance between the trees and $l_1$ and $l_2$ are the number of leaves in each tree. Notice that this cost depends on the algorithm output. The best known upper limit for this problem is due to an algorithm presented in [Chen, 2001] with complexity $O(n_1 n_2 + l_1^2 + l_1^{2.5} l_2)$.

Despite the inherent complexity of the mapping problem in its generic formulation, there are several practical applications that can be modelled using restricted formulations of it. By imposing conditions on the basic operations corresponding to the original formulation in Definition 1 (i.e., replacement, insertion and removal which change $S$, $I$ and $D$ in Eq. 1), several restricted formulations are obtained for which more convenient and faster algorithms have been proposed [Wang and Zhang, 2001].

### 3.2.2 Categories of Abstraction

When attempting to match two objects, an important technique is to look for their abstractions and refinements. This is helpful because disagreements as to the level of detail that is necessary for expressing a concept are very common.

In [Giunchiglia and Walsh, 1992], Giunchiglia and Walsh categorise the various kinds of abstraction operation they found in a wide-ranging survey. By inverting each abstraction operation, a corresponding refinement operation can be developed. Operations from both these categorisations can be used to match constraints and relations.

Giunchiglia and Walsh's categories are as follows:

**Predicate:** Two or more predicates are merged, typically to the least general generalisation in the predicate type hierarchy, e.g.,

*(Bottle ?X) + (Cup ?X)* $\mapsto$ *(Container ?X).*

**Domain:** Two or more terms are merged, typically by moving the functions to the least general generalisation in the domain type hierarchy, e.g.,

*(Aunt Me) + (Cousin Me)* $\mapsto$ *(Relation Me).*

**Propositional:** One or more arguments are dropped, e.g.,

*(Bottle A) + (Bottle B)* $\mapsto$ *(Bottle).*

**Precondition:** The precondition of a rule is dropped, e.g.,

*[(Ticket ?X)* $\rightarrow$ *(Travel ?X)]* $\mapsto$ *(Travel ?X).*

Predicate, domain and propositional abstractions and refinements all apply to the structure of first-order terms. Precondition abstraction or refinement does not apply directly to the first-order terms but instead applies to the way in which these are ordered in a rule. This corresponds to the way in which the individual constraints form Horn clauses. However, as from the precondition abstraction definition, there is no possibility for mismatches here in the context in which we are viewing the problem.

We say that one term is an abstraction/refinement of another if, after the application of a finite number of predicate, domain and propositional refinement/abstraction operations, the terms are equivalent. We say that an abstraction/refinement relation holds between two terms $a$ and $b$ if $a$ is an abstraction/refinement of $b$.

### 3.2.3 Structure Matching

In order to perform semantic matching between the first-order constraints found in an LCC IM, we consider the first-order terms as trees and perform tree matching on them. There are two stages in the matching process:

1. Node matching: solves the semantic heterogeneity problem by considering only labels at nodes and their contextual information inside constants in IM and web service descriptors.

2. Tree matching: exploits the results of the node matching and the structure of the term to find an overall match between the terms in a web service description and in IM.

#### Node matching

Semantic matching, as from [Giunchiglia and Shvaiko, 2003] is based on the 2 key notions:

- *Concept of a label*, which is a logical formula encoding the meaning of a label;

- *Concept of a node*, which is a logical formula that encodes the meaning of a node, given that it has a certain label and is in certain position in the term tree and in the IM classification (see Figure 4 for example).

We say that two nodes $n_1$ and $n_2$ in the trees $T_1$ and $T_2$ (semantically) match iff the formula"$c@n_1$ *iff* $c@n_2$" holds given the available background knowledge, where $c@n_1$ and $c@n_2$ are the concepts at nodes of $n_1$ and $n_2$ respectively.

The semantic node matching algorithm, as introduced in [Giunchiglia et al., 2004], takes as input two term trees and computes as output a set of correspondences holding among the nodes in the trees in four macro steps:

*Step 1: for all labels L in two trees, compute concepts of labels,* $C_L$. Step 1 is concerned with automatic translation of ambiguous natural language labels taken from the term tree elements into an internal logical language with Boolean semantics (see [Giunchiglia et al., 2004] for more detail). The process involves tokenization, lemmatization, querying the Oracle (such as WordNet [Miller, 1995]) in order to determine the label senses, and, finally, the complex concept construction. The last step is concerned with interpretation of certain natural language labels as the logical connectives (for example both natural language *and* and *or* are translated into disjunction) and word sense disambiguation (see [Magnini et al., 2004] for more detail). Thus, for example, the concept of label *Number of bottles* is computed as $C_{Number\ of\ bottles} = C_{Number} \sqcap C_{bottles}$, where $C_{bottles} = \langle bottle, senses_{WN}\#4 \rangle$ is taken to be the union of four WordNet senses, and similarly for *number*.

*Step 2: for all nodes N in two trees, compute concepts at nodes,* $C_N$. During Step 2 we analyse the meaning of the positions that the labels of nodes have in a tree. Term trees are hierarchical structures where the path from the root to a node uniquely identifies that node (and also its meaning). We define the logical formula for a concept at node as a conjunction of concepts of labels located in the path from the given node to the root. For example, in Figure 6, the concept at node for the node *Champagne(C)* is computed as follows: $C_{Champagne(C)} = C_{Wine} \sqcap C_{Region} \sqcap C_{Champagne}$.

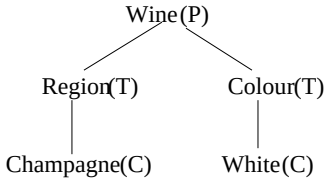
In order to constrain the set of possible concept at node interpretations, sense filtering techniques are used (see [Magnini et al., 2004, Giunchiglia et al., 2004] for detailed discussion). The main goal of sense filtering techniques is to filter out irrelevant (for the given matching task) Oracle senses from concepts of labels. For all concepts of labels we collect all their ancestors and descendants. We call them a focus set. Notice that the IM itself can be classified in a tree like structure (see Figure 4 for example). Therefore, the focus set is enriched with concept of labels of the IM and its ancestors in the IM classification. For example, as from Figures 4 and 6, for the concept at node $C_{Champagne(C)}$ the focus set contains the following concepts of labels: $C_{Wine}$, $C_{Region}$, $C_{Champagne}$ taken from the term tree along with $C_{Wine}$, $C_{Selling}$ and $C_{Recommendation}$ taken from IM classification. Then all Oracle senses of atomic concepts of labels from the focus set are compared with the senses of the atomic concepts of labels of the concept at node. If a sense of atomic concept of label is connected by an Oracle relation with the sense taken from the focus set, then all other senses of these atomic concepts of labels are discarded. Therefore, as a result of sense filtering step we have (i) the Oracle senses which are connected with any other Oracle senses in the focus set or (ii) all the Oracle senses otherwise. After this step

the meaning of concept of labels is reconciled in respect to the knowledge residing in both the term tree and IM classification structures.
*Step 3: for all pairs of labels in two trees, compute relations among $C_L$'s.* Step 3 is concerned with acquisition of "world" knowledge. Relations between concepts of labels are computed with the help of a library of element level semantic matchers (see [Giunchiglia et al., 2004] for more detail). These matchers take as input two concepts of labels and produce as output a semantic relation (e.g., equivalence, more/less general) between them. For example, from WordNet we can derive that region and area are synonyms, and therefore, $C_{region} = C_{area}$.
*Step 4: for all pairs of nodes in two trees, compute relations among $C_N$'s.* Step 4 is concerned with the computation of the relations between concepts at nodes. This is done by reducing this problem to a propositional satisfiability (SAT) problem and by exploiting state of the art SAT decider [Giunchiglia and Shvaiko, 2003, Giunchiglia et al., 2004].

It is important to notice that Step 1 and Step 2 can be done once for all, independently of the specific matching problem. Step 3 and Step 4 can only be done at run time, once the two trees which must be matched have been chosen.

**Tree matching**

In order to match first-order terms, we consider them as trees. Thus a constraint such as $wine(Cxt : (region, champagne), Cxt : (white, colour))$ would be represented as shown in Figure 6, where $Cxt$ contains the relevant contextual information; this is left implicit.




*P = predicate; T = type; C = constant*

Figure 6: Constraint $wine(champagne, white)$ expressed as a tree

In order to satisfy a set of constraints in a message in an IM, it is necessary to satisfy at least one constraint of every disjunction in the CNF. In a more complex situation, we might match a single constraint to many constraints, or match many-to-many; however, we only consider one-to-one matching in this paper.

Semantic node matching is done prior to the tree matching process, and the results of this are used to determine which nodes in the trees correspond to each other, and, when extended to deal with approximate mapping, how strong this correspondence is. Semantic tree matching is thus the combination of the results of semantic node matching with techniques that take into account the structure of the term. It must determine not only whether the objects used are the same or similar but whether they are organised in the same manner. This organisation of the terms encodes important semantic information about how they relate to one another, and the semantic tree matching techniques

determine whether these relationship are the same or similar between apparently different constraints.

We say that two trees $T_1$ and $T_2$ match iff for any node $n_{11}$ in $T_1$ there is a node $n_{21}$ in $T_2$ such that

- $n_{11}$ semantically matches $n_{21}$;
- $n_{11}$ and $n_{21}$ reside on the same depth in $T_1$ and $T_2$ respectively;
- all ancestors of $n_{11}$ are semantically matched to the ancestors of $n_{21}$;

At this stage, we assume that the problem of semantic node matching has been dealt with and can be called as a subprocess of the semantic tree matching, the details of which are discussed in Section 3.2.3.

**Approximate matching**

We say that two nodes $n_1$ and $n_2$ in the trees $T_1$ and $T_2$ approximately match iff $c@n_1$ R $c@n_2$ holds given the available background knowledge, where $c@n_1$ and $c@n_2$ are the concepts at nodes of $n_1$ and $n_2$, and where $R \in \{\equiv, \sqsubseteq, \sqsupseteq, \wedge, \bot, not\ related\}$.

We say that two trees $T_1$ and $T_2$ match iff there is at least one node $n_{11}$ in $T_1$ and a node $n_{21}$ in $T_2$ such that

- $n_{11}$ approximately matches $n_{21}$;
- all ancestors of $n_{11}$ are approximately matched to the ancestors of $n_{21}$;

The key point of approximate match is that we allow mismatches both on the node and structure level.

We distinguish between the following semantic relations holding between first-order terms ($\equiv$,$\sqsupseteq$,$\sqsubseteq$, where $\sqsupseteq$ and $\sqsubseteq$ stand for refinement and abstraction respectively). Thus, for example, we consider *vehicle(Nissan)* as an abstraction of *vehicle(Acura, yellow)* given that $Nissan$ is more general than $Acura$. In fact consequent application of propositional and domain abstractions allow us to obtain the former from the latter. However, in some cases the terms are neither abstractions nor refinements of each other. Consider *vehicle(amber)* and *car(yellow)* terms given that *car* is less general than *vehicle* and *amber* is less general than *yellow*. The similarity of the terms can be estimated by application of approximate structure matching algorithm.

The pseudo code in Figure 7 illustrates approximate structure matching algorithm. **approximateStructureMatch** takes as an input *source* and *target* term trees and some *threshold* value, which allows us to adjust the required similarity of the trees. Here and throughout the paper we assume that the source tree is derived from an IM constraint and the target tree represents the term derived from the peer capability description. **approximateTreeMatch** fills the *result* array (line 3) which stores the mappings holding between the nodes of the trees. A TreeMatchingElement *tme* is computed (line 4) by **analyzeMismatches**. If *tme* stands for equivalence or less generality relations (line 5) (i.e., the constraint in IM (*source*) can be satisfied by the peer capabilities (*target*)) and

```
Node struct of
  int nodeId;
  String label;
  String cLabel;
  String cNode;
MappingElement struct of
  int MappingElementId;
  Node source;
  Node target;
  String relation;
TreeMappingElement struct of
  Tree of Nodes source;
  Tree of Nodes target;
  String relation;
  double approximationScore;

1.MappingElement[] approximateStructureMatch(Tree of Nodes source, target,
                                                          double threshold)
2. MappingElement[] result;
3. approximateTreeMatch(source,target,result);
4. TreeMappingElement tme=analyzeMismatches(source,target,result);
5. if (getRelation(tme)=="=") or (getRelation(tme)=="<")
6.  if (getApproximationScore(tme)>threshold)
7.   return result;
8. return null;

9. void approximateTreeMatch(Tree of Nodes source,target,MappingElement[] result)
10. Node sourceRoot=getRoot(source);
11. Node targetRoot=getRoot(target);
12. String relation= nodeMatch(sourceRoot,targetRoot);
13. if (relation!="Idk")
14.  addMapping(result,sourceRoot,targetRoot,relation);
15. Node[] sourceChildren=getChildren(sourceRoot);
16. Node[] targetChildren=getChildren(targetRoot);
17. For each sourceChild in sourceChildren
18.  Tree of Nodes sourceChildSubTree=getSubTree(sourceChild);
19.  For each targetNode in target
20.   Tree of Nodes targetChildSubTree=getSubTree(targetChild);
21.   approximateTreeMatch(sourceChildSubTree,targetChildSubTree, nodesToMatch);
```

Figure 7: Pseudo Code for Approximate Structure Matching algorithm

if an *approximationScore* exceeds *threshold* (line 6) the mappings calculated by **approximateTreeMatch** are returned (line 7). **approximateTreeMatch** starts from obtaining the roots of *source* and *target* trees (lines 10-11). The semantic relation holding between them is computed by **nodeMatch** (line 12) implementing the node matching algorithm. If a semantic relation is computed, the corresponding mapping is saved to *result* array (lines 14) and the children of the root nodes are obtained (line 15-16). Finally the loops on *sourceChildren* and *targetChildren* (lines 17-21) allow to call **approximateTreeMatch** recursively for all pairs of sub trees rooted at *sourceChildren* and *targetChildren* elements. **analyzeMismatches** decides the importance of the mismatches among the nodes of the trees (if any) and calculates the aggregate score of tree match quality by exploiting a tree edit distance algorithm [Zhang et al., 1992, Chen, 2001].

### 3.2.4 Tree edit distance revisited

The contextualised tree representations of first-order terms introduced in Section 2 are complex objects composed from the nodes of different types: predicates/functions, types and variables. In this case, the tree edit distance score described in Section 3.2.1 can hardly correspond to the semantic similarity of the terms. Our proposal in this paper is to restrict the formulation of the tree edit distance problem in order to reflect the semantics of the first-order terms. In order to achieve this goal, we propose to adjust the tree edit distance operation weights to reflect both the different kinds of nodes and different semantic relations produced by the node matching algorithm. Notice that the tree edit distance problem as defined in Section 3.2.1 assumes that all the correspondences in the mapping are equivalences. However, the approximate structure matching algorithm produces correspondences which stand for equivalence, less/more generality and disjointness relations. We propose to address these challenges by an abstraction theoretic approach [Giunchiglia and Walsh, 1992]. In particular, as from Section 3.2.2, propositional abstraction/refinement can be viewed as the deleting/adding of the nodes that correspond to predicates, functions or variables in the term trees while predicate and domain abstractions/refinements can be viewed as relabelling of the nodes that stand for more/less general predicates and functions in the term trees. Therefore, in order to preserve the semantics of the terms, we allow only the edit distance operations that have their abstraction theoretic counterparts and prohibit all the other operations by assigning to them an infinite cost. Table 1 illustrates the costs of the tree edit distance operations.

Table 1: Cost of tree editing operations for the equivalence approximation score

| Operation | Cost | Comments |
|---|---|---|
| replace(a,b), $a = b$ | 0 | |
| replace(a,b), $a \sqsubseteq b$ | 1 | a and b correspond to predicates or functions |
| replace(a,b), $a \sqsupseteq b$ | 1 | a and b correspond to predicates or functions |
| insert(a) | 1 | a corresponds to predicate, function or variable |
| delete(a) | 1 | a corresponds to predicate, function or variable |
| Others | $\infty$ | |

Notice that the two terms may have a high edit distance value but one of the terms may be a refinement of the other, which is a desirable outcome for the scenario presented in Section 2. Therefore, we need to estimate not only equivalence but also abstraction/refinement approximation scores.

A term is an abstraction/refinement of the other if, after a finite number of refinement/abstraction operations, the terms are equivalent. Therefore, the tree edit distance operations corresponding to these refinement/abstraction steps should not influence the approximation score (i.e., their costs have to be equal to 0). Table 2 illustrates the costs of the tree edit distance operations for abstraction and refinement approximation scores.

This extension influences the tree edit distance computation. In particular, Eq. 2 reflects the updates to the tree edit distance score defined in Eq. 1:

$$c = \sum_{(n_1;n_2)\in M} p(n_1, n_2, R) + \sum_{n\in I} q(n, T) + \sum_{n\in D} r(n, T) \tag{2}$$

Table 2: Cost of tree editing operations for the abstraction ($Cost_a$) and refinement ($Cost_r$) approximation scores

| Operation | $Cost_a$ | $Cost_r$ | Comments |
|---|---|---|---|
| replace(a,b), $a = b$ | 0 | 0 | |
| replace(a,b), $a \sqsubseteq b$ | 1 | 0 | a and b correspond to predicates or functions |
| replace(a,b), $a \sqsupseteq b$ | 0 | 1 | a and b correspond to predicates or functions |
| insert(a), left(source) tree | 1 | 0 | a corresponds to predicate, function or variable |
| delete(a), left(source) tree | 0 | 1 | a corresponds to predicate, function or variable |
| insert(a), right(target) tree | 0 | 1 | a corresponds to predicate, function or variable |
| delete(a), right(target) tree | 1 | 0 | a corresponds to predicate, function or variable |
| Others | $\infty$ | $\infty$ | |

where $n_1$ and $n_2$ are the nodes touched by a correspondence in $M$; $T$ is the tree of the node $n$; $R$ the semantic relation holding between $n_1$ and $n_2$; $M$, $I$ and $D$ are as defined in Section 3.2.1. Therefore, contrary to Eq. 1, the tree edit distance costs in Eq. 2 depend on the particular nodes in the tree, the semantic relation holding among the nodes and the tree itself.

Notice that the mapping produced by the approximate structure matching algorithm does not in general satisfy the conditions presented in Definition 1. Therefore, in order to apply a tree edit distance algorithm, we have to ensure that it is a (proper) mapping. The first condition in Definition 1 requires a 1-to-1 mapping. Therefore we delete all the correspondences that violate this requirement. The second condition requires the order preservation among sibling nodes. Notice that sibling ordering does not influence the meaning of the term and hence the ability of the peer to interpret the constraint. Therefore, in order to satisfy the condition, the sibling nodes have to be reordered. Figure 8b illustrates an example of such reordering for the trees depicted on Figure 8a.

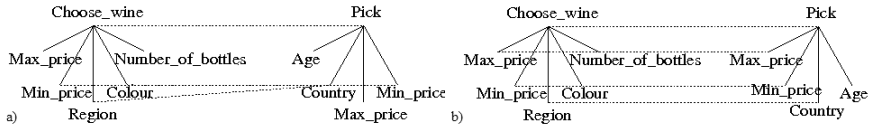



Figure 8: Approximate mappings between two constraints (a) and their reordering (b)

The third condition enforces the hierarchical relation between the nodes of the trees. In order to satisfy it, we delete from the partial mapping all the correspondences that violate it.

Since we are interested in similarity rather than in distance we exploited the following similarity score:

$$Sim = 1 - \frac{EditDistance}{max(n_1, n_2)} \tag{3}$$

where $n_1$ and $n_2$ stand for the number of nodes in the trees. Notice that for the special case of *EditDistance* equal to $\infty$ the similarity score is estimated to 0.

To ensure a quick prototyping approach we exploited a simple tree edit distance algorithm from Valiente's work [Valiente, 2002] for *EditDistance* calculation. In this algorithm, deletion and insertion operations are performed only on the leave nodes, which

improves the time complexity. The algorithm finds the least-cost transformation of an ordered tree $T_1$ and $T_2$ in $O(|n_1||n_2|)$ time using $O(|n_1||n_2|)$ additional space (see Lemma 2.20 in [Valiente, 2002]).

## 3.3 Results and Evaluation

We have implemented the algorithm described in the previous sections in Java and evaluated its matching quality on 132 pairs of first order logic terms. Half of the pairs were composed from the equivalent terms (e.g., *journal(periodical-publication)* and *magazine(periodical-publication)*) while the other half were composed from similar but not equivalent terms (e.g., *web-reference(publication-reference)* and *thesis-reference (publication-reference)*). The terms were extracted from different versions of the Standard Upper Merged Ontology (SUMO)[2] and the Advance Knowledge Transfer (AKT)[3] ontologies[4]. These are both first-order ontologies, so many of these differences mapped well to the potential differences between constraints that we are investigating.

In our evaluation we exploited the commonly accepted measures of matching quality: precision, recall, and F-measure. Precision varies in the [0,1] range; the higher the value, the smaller the set of incorrect correspondences (false positives) which have been computed. Precision is a correctness measure. Recall varies in the [0,1] range; the higher the value, the smaller the set of correct correspondences (true positives) which have not found. Recall is a completeness measure. F-measure varies in the [0,1] range. The version computed here is the harmonic mean of precision and recall. It is a global measure of the matching quality, increasing as the matching quality improves. The evaluation was performed on the Pentium 4 computer.

Figure 9 presents the matching quality measures depending on the cut-off threshold value. As from the figure, the algorithm demonstrates high matching quality on the wide

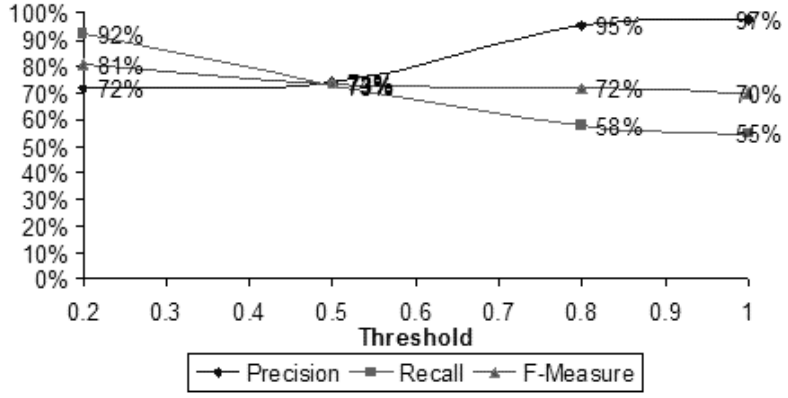



Figure 9: The matching quality measures depending on threshold value

range of threshold values. In particular, F-Measure values exceed 70% for this range. Table 3 summarises the time performance of the matching algorithm. It presents the

[2]http://ontology.teknowledge.com/

[3]http://www.aktors.org

[4]See http://dream.inf.ed.ac.uk/projects/dor/ for full versions of these ontologies and analysis of their differences

Table 3: Time performance of approximate structure matching algorithm (average on 132 term matching tasks)

| | Node matching Step 1 and 2 | Node matching Step 3 and 4 | Tree matching |
|---|---|---|---|
| Time, ms | 134.1 | 3.3 | 0.9 |

average time taken by the various steps of the algorithm on 132 term matching tasks. As from the table, Step 1 and 2 of the node matching algorithm significantly slow down the whole process. However, as discussed in section 3.2.3, these steps correspond to the linguistic preprocessing of the natural language labels that can be performed once and later reused by the matching process. Given that the term can be automatically annotated with the linguistic preprocessing results [Giunchiglia et al., 2004], the term matching task is performed in average in 4.2 ms, which corresponds roughly to 240 term matching tasks per second. This level of performance satisfies the requirements of dynamic and run time matching.

## 3.4 Estimating the Importance of Nodes

The tree edit distance problem defined in Section 3.2.1 assumes that the tree edit distance operations costs are independent on the particular node to which the given operation is applied. However, one may argue that in certain cases the given node in the tree is more (or less) important than the others. For example, the node labelled *kind* in Figure 10 plays mostly "organizational" role in the tree and has little influence on the term meaning.

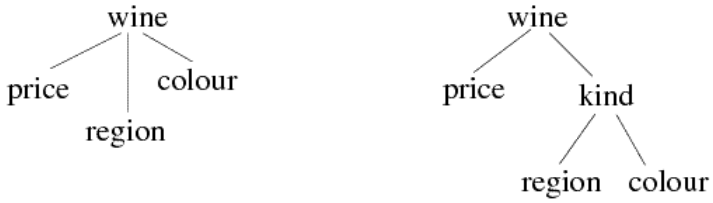



Figure 10: Two term trees

We propose to associate with every node in the trees the numerical importance score and then weight the tree edit distance operations involving the node with the score. For example, the cost of deletion of a node assigned the importance score 2 is doubled. The importance scores can be assigned both manually and (semi-)automatically. The former case corresponds to the incorporation of user preferences into the matching process. For example, in the wine buying scenario (Figure 10), a customer may have a slight preference for French wine and a strong preference for red wine. The latter case corresponds to so called weighting schemas that are applied to the nodes in the trees. One may argue that the nodes residing closer to the tree root have to be weighted higher than the nodes residing closer to the leaves. For example, the node *car* in the term *car(Nissan, yellow)* is probably more important than the node *yellow* since the user would consider a white car

as a better choice than a yellow table. On the other hand this weighting schema assigns a greater weight to the node *kind* in Figure 10 than the probably more important node *region* that resides deeper in the tree structure.

We adopt the other weighting schema based on assigning information content values to the concept of labels in the tree. The information content defines the generality or specificity of a concept in a certain topic. The information content of the given concept is calculated as follows. Firstly the frequency[5] of concept occurrences $F_C$ in the given *text corpus* is calculated. Then the frequencies of all subsuming concepts in the taxonomy (such as WordNet taxonomy [Miller, 1995]) are calculated and added to $F_C$. Thus the root concept will count the occurrences of all the concepts in its taxonomy. In the case of WordNet concepts (or synsets) the frequency counts are precomputed for wide range of large scale corpora.

The information content (IC) of a concept $c$ is defined as follows:

$$IC(c) = -\ln\left(\frac{freq(c)}{freq(root)}\right) \tag{4}$$

where $freq(c)$ and $freq(root)$ are, respectively, the frequencies of the concept $c$ and of the *root* of the taxonomy. Notice that the fraction represents the probability of occurrence of the concept in a large corpus. Notice also that quantifying information content in this way makes intuitive sense in this setting: as frequency increases, informativeness decreases, so the more abstract a concept, the lower its information content. Therefore, *kind* in Figure 10 will have a smaller information content value than *region* since its frequency in the textual corpora is significantly higher. Notice also that the exploitation of domain specific corpora for the information content values estimation may potentially improve the matching quality for the domain specific matching tasks.

# 4 Further work: Global Good Enough Answers

The focus of the work on good enough answers so far has been on developing the matching techniques for local GEA. The work on global GEA, therefore, still remains to be done. In this section, we discuss our aims and ideas concerning this work.

## 4.1 Incorporating trust

Trust is a multi-faceted concept that has received increasing attention recently [Ramchurn et al., 2005, Yolum and Singh, 2003, Sabater and Sierra, 2002a, Falcone and Castelfranchi, 2001]. In the context of negotiation, trust represents a general assessment on how 'serious' an agent is about the negotiation process, i.e. that his proposals 'make sense' and he is not 'flying a kite', and that he is committed to what he signs. A lack of trust may provoke agents to breakdown negotiations, or to demand additional guarantees to cover the risk of potential defections. Therefore, in any model of trust the central issue is how to model expectations

[5]Here and throughout the paper, following the natural language understanding community tradition, we treat frequency as count (i.e., frequency of concept occurrences is the number of times the given concept occurs in the corpora).

about the actual outcome at contract execution time. Contracts, when executed, may, and frequently do, yield a different result to what was initially signed. Goods may be delivered late, quality may be (perceived) different from the contract specification, extra costs may be claimed, etc. So the outcome is uncertain to some extent, and trust, precisely, is a measure of how uncertain the outcome of a contract is. Naturally, the higher the trust in a partner the more sure we are of his or her reliability. Trust is therefore a *measure of expected deviations of behaviour* along a given dimension, and in many cases for a given value (region) in that dimension (e.g. I might trust you on low-priced contracts but not on high-priced ones). In this sense, the higher the trust the lower the expectation that a (significant) deviation from what is signed occurs. Trust values can be used for three basic purposes:

- Trust permits us to select *what offer to send next*. As trust measures the expected deviation of behaviour of our opponent, we can use it as a 'counter-balance' by asking for even better deals.

- Trust permits us to better *select negotiation partners*. Humans (normally) prefer to live in worlds with low uncertainty about the future. The higher the trust on a partner the less probable a deception is going to happen, even if what is to be expected is not extraordinarily good. This would also explain the fact that humans tend not to explore too much once they settle down in a situation that is satisfactory enough. That is, a situation in which they have a landscape of trust that permits them to select partners in a good enough way. If I'm satisfied with my butcher I will not explore more butchers unless I am disappointed at some point.

- Trust *simplifies negotiation dialogues*. Trust is mostly built through ongoing relationships, the repetition of negotiation dialogues and contract executions means that some terms and conditions (negotiation dimensions) need not be discussed and specified again and again. A high trust in what will happen along a certain dimension enables partners to omit it from the negotiation dialogue. The higher the trust the smaller the number of dimensions of the negotiation object, and thus the shorter the negotiation dialogue.

In the OpenKnowledge context, the ability of a peer to perform a role cannot be determined solely through the qualitative scores it returns for this role. We can assume that all peers have the ability to accurately determine their ability to interpret constraints because all OpenKnowledge peers have access to the mapping module. However, we cannot assume that peers will be honest about their abilities: they may derive a very low score and then report a high score and, since noone has access to the peer's full knowledge base apart from that peer, there is no way these scores can be externally verified. Additionally, some peers may be more reliable in other ways: for example, they may have a strong connection and always locateable during interaction; they may be conscientious and not abandon the interaction half way through, and so on. Another factor that cannot be determined through the matching scores is how likely the peer will be to actually satisfy the constraints that they can interpret. For example, two wine merchant peers may be equally able to map the constraint $in\_stock(Make, No\_bottles)$ to

one of their abilities, but one of the peers may have a large number of different wines in stock and therefore be more likely to be able to satisfy the constraint during interaction than the other peer, who may have only a small number of wines.

All of these unknowns are important factors in determining whether or not to choose to interact with a peer. The way we have envisaged the choosing of peers for roles, as described in the lifecycle steps, is that their matching scores are combined with trust scores to give an overall estimate of which peer will perform a role best. We therefore need to consider how these trust scores could be generated. In this sense, we are exploring different trust and reputation models with different characteristics.

The different models found in the annexes of this deliverable propose summary measures to represent trust and reputation. These measures could be used at least in two aspects of good enough matching. First, to improve the tree edit distance in the context of negotiation by allowing for more flexibility (distance) in the acceptance of offers when they come from trustworthy agents. That is, a proposal (term) with a large distance from out current most desired proposal will be less acceptable coming from an agent we don't trust, because the final outcome could be far more distant that the proposal.[6] The *Sierra-Debenham model* in Annex A and *Repage* in Annex C propose a probabilistic approach to model trust and/or reputation while *CREDIT* in Annex D proposes a possibilistic/fuzzy model. All of them can be used to assess the expected deviation of behaviour. *ReGreT* in Annex B proposes an ad-hoc model for the same purpose. Second, to improve the process of abstraction and refinement distance calculation. The *Sierra-Debenham model* possibility of representing nested probability distributions over predicates structured as an ontology could help in assessing different distances in the abstraction process used for node matching. That is we can use the previous experience and reputation to distinguish between the distance incurred by different abstraction steps. *ReGreT*'s ontological dimension could serve the same purpose. Finally, *CREDIT*'s normative system could help in defining a declarative model of abstraction steps. These norms can be used in the term matching procedures as they expand or contract the issues in a contract under negotiation.

## 4.2 Quantitative and qualitative feedback

Our LGEA techniques provide, for any constraint and peer, a quantitative judgement as to how well that constraint maps to an ability of the peer's, and this can be used to estimate how well a peer can perform a given role in an interaction. This process is fully automatic and sensitive to user's requirements through the weighting process. However, in some situations, users may wish to have further input in this process. In such situations, the matching process can provide qualitative information about the matching process for each role back to the user. For every role in an interaction, the suitable peers will be ranked according the quantitative scores returned by the matching process. If the user has chosen not to be involved in the selection process, the system will automatically select the highest ranked available peer to perform the role. However, if the user wishes to be further involved, they can choose to view qualitative feedback for any of the potential

[6] Trust is usually understood as the expected deviation of behaviour with respect to commitments.

peers. This will provide information on which constraints contained mismatches and what these approximate matches were. This information may cause a user to reject a high ranking peer. Additionally, this information would not only concern correctness, or situations in which a peer failed to meet some of the requirements of a constraint, but also completeness, or in what ways a peer's ability exceeded the requirements of the constraint. The completeness information is not used to generate the matching scores because a peer whose abilities exceed those required is still capable of completely satisfying this lesser requirement. However, this information could be helpful to the user, who may thereby discover further aspects of constraints that may be useful. For example, consider a constraint in an IM $choose\_wine(Price, Region, Colour, No\_bottles)$. A peer with an ability to satisfy $choose\_wine(Price, Region, Colour, No\_bottles, Age)$ would be able to fulfil this constraint acceptably by dropping its final argument, $Age$. However, a user, on viewing this qualitative matching information, may realise that $Age$ was something that was important to him but which he had failed to consider before. It is not useful for the user to simply choose this peer above others and proceed with the interaction as it stands, because the constraint to be satisfied will still be the simplified one and the peer cannot use its extra ability to satisfy age. However, this information could be a trigger for the user to slightly redesign the IM so that the constraint now included this extra attribute and the lifecycle would start again. This peer would then perform better on this subsequent round of matching than other peers that could not offer this $Age$ attribute, though this would not necessarily lead it to become the highest ranked peer.

### 4.3 Moving on from one-to-one matching

We are currently only considering matching constraints one by one to specific abilities of peers. However, this is fairly simplistic approach. There may be some constraints that are covered by two abilities or many constraints that map to many abilities but not in a simple one-to-one pattern. We therefore need to extend our matching processes to take such things into consideration.

## 5 Conclusions

This deliverable describes the techniques we have developed to judge the quality of the matchings between the constraints in an interaction model, which describe what a peer performing a role needs to be able to perform, and the abilities of that peer, which describe what that peer can actually do. This matching process is currently implemented for one-to-one constraint matching and we describe the encouraging results we have got so far. In order for this matching process to be useful in the OpenKnowledge system, we need to develop the lifecycle in which this process will occur, which we refer to as global good enough matching, and to incorporate trust judgments in the process, and our aims for these are described in the further work section.

# A Information-Based Negotiation

We ground our negotiation model on information-based concepts. *Entropy*, $H$, is a measure of uncertainty [MacKay, 2003] in a probability distribution for a discrete random variable $X$: $H(X) \triangleq -\sum_i p(x_i) \log p(x_i)$ where $p(x_i) = P(X = x_i)$. Maximum entropy inference is used to derive sentence probabilities for that which is not known by constructing the "maximally noncommittal" [Jaynes, 2003] probability distribution.

Let $\mathcal{G}$ be the set of all positive ground literals that can be constructed using our language $\mathcal{L}$. A *possible world*, $v$, is a valuation function: $\mathcal{G} \rightarrow \{\top, \bot\}$. $\mathcal{V}|\mathcal{K} = \{v_i\}$ is the set of all possible worlds that are consistent with an agent's knowledge base $\mathcal{K}$ that contains statements which the agent believes are true. A *random world* for $\mathcal{K}$, $W|\mathcal{K} = \{p_i\}$ is a probability distribution over $\mathcal{V}|\mathcal{K}^a = \{v_i\}$, where $p_i$ expresses an agent's degree of belief that each of the possible worlds, $v_i$, is the actual world. The *derived sentence probability* of any $\sigma \in \mathcal{L}$, *with respect to* a random world $W|\mathcal{K}$ is:

$$(\forall \sigma \in \mathcal{L}) P_{\{W|\mathcal{K}\}}(\sigma) \triangleq \sum_n \{\, p_n \;:\; \sigma \; is \; \top \; in \; v_n \,\} \tag{5}$$

The agent's *belief set* $\mathcal{B} = \{\varphi_j\}_{j=1}^M$ contains statements to which the agent attaches a *given sentence probability* $B(\cdot)$. A random world $W|\mathcal{K}$ is *consistent* with $\mathcal{B}$ if: $(\forall \varphi \in \mathcal{B})(B(\varphi) = P_{\{W|\mathcal{K}\}}(\varphi))$. Let $\{p_i\} = \{\overline{W}|\mathcal{K}, \mathcal{B}\}$ be the "maximum entropy probability distribution over $\mathcal{V}|\mathcal{K}$ that is consistent with $\mathcal{B}$". Given an agent with $\mathcal{K}$ and $\mathcal{B}$, *maximum entropy inference* states that the *derived sentence probability* for any sentence, $\sigma \in \mathcal{L}$, is:

$$(\forall \sigma \in \mathcal{L}) P_{\{\overline{W}|\mathcal{K},\mathcal{B}\}}(\sigma) \triangleq \sum_n \{\, p_n \;:\; \sigma \; is \; \top \; in \; v_n \,\} \tag{6}$$

From Eqn. 6, each belief imposes a linear constraint on the $\{p_i\}$. The maximum entropy distribution: $\arg\max_{\underline{p}} H(\underline{p})$, $\underline{p} = (p_1, \ldots, p_N)$, subject to $M + 1$ linear constraints:

$$g_j(\underline{p}) = \sum_{i=1}^{N} c_{ji} p_i - B(\varphi_j) = 0, \quad j = 1, \ldots, M.$$

$$g_0(\underline{p}) = \sum_{i=1}^{N} p_i - 1 = 0$$

$c_{ji} = 1$ if $\varphi_j$ is $\top$ in $v_i$ and 0 otherwise, and $p_i \geq 0, i = 1, \ldots, N$, is found by introducing Lagrange multipliers, and then obtaining a numerical solution using the multivariate Newton-Raphson method. In the subsequent subsections we'll see how an agent updates the sentence probabilities depending on the type of information used in the update.

## A.1 Updating from decay and experience

An important aspect that we want to model is the fact that beliefs 'evaporate' as time goes by. If we don't keep an ongoing relationship, we somehow forget how *good* the opponent was. If I stop buying from my butcher, I'm not sure anymore that he will sell me the 'best' meat. This decay is what justifies a continuous relationship between

individuals. In our model, the conditional probabilities should tend to ignorance. If we have the set of observable contracts as $B = \{b_1, b_2, \ldots, b_n\}$ then complete ignorance of the opponent's expected behaviour means that given the opponent commits to $b$ the conditional probability for each observable contract becomes $\frac{1}{n}$ — i.e. the unconstrained maximum entropy distribution. This natural decay of belief is offset by new observations.

We define the evolution of the probability distribution that supports the previous definition of decay using an equation inspired by pheromone like models [Dorigo and Stützle, 2004]:

$$P^{t+1}(b'|b) = \kappa \cdot \left( \frac{1-\rho}{n} + \rho \cdot \left( P^t(b'|b) + \Delta^t P(b'|b) \right) \right) \tag{7}$$

where $\kappa$ is a normalisation constant to ensure that the resulting values for $P^{t+1}(b'|b)$ are a probability distribution. This equation models the passage of time for a conveniently large $\rho \in [0, 1]$ and where the term $\Delta^t P(b'|b)$ represents the increment in an instant of time according to the last experienced event as the following possibilities show.

**Similarity based.** The question is how to use the observation of a contract execution $c'$ given a signed contract $c$ in the update of the overall probability distribution over the set of all possible contracts. Here we use the idea that given a particular deviation in a region of the space, *similar* deviations should be expected in other regions. The intuition behind the update is that if my butcher has not given me the quality that I expected when I bought lamb chops, then I might expect similar deviations with respect to chicken. This idea is built upon a function $f(x, y)$ that takes into account the difference between acceptance probabilities and similarity between the perception of the execution $x$ of a contract $y$, that is a contract for which there was an Accept$(\beta, \alpha, y)$. Thus, after the observation of $c'$ the increment of probability distribution at time $t + 1$ is:

$$\Delta^t P(b'|b) = (1 - |f(c', c) - f(b', b)|) \tag{8}$$

where $f(x, y)$ is

$$f(x, y) = \begin{cases} 1 & \text{if } P^t(\text{Accept}(x)) > P^t(\text{Accept}(y)) \\ \text{Sim}(x, y) & \text{otherwise.} \end{cases}$$

and where Sim is an appropriate similarity function (reflexive and symmetric) that determines the indistinguishability between the perceived and the committed contract.

**Entropy based.** Suppose that $\alpha$ observes the event $(c'|c)$, the entropy based approach estimates $\Delta^t P(b'|b)$ by applying the principle of minimum relative entropy.[7] Let:

$$\left(P_C^t(b_j|b)\right)_{j=1}^n = \arg\min_{\underline{p}} \sum_{i=1}^n p_i \log \frac{p_i}{P^t(b_i|b)} \tag{9}$$

[7] Given a prior probability distribution $\underline{q} = (q_i)_{i=1}^n$ and a set of constraints, the *principle of minimum relative entropy* chooses the posterior probability distribution $\underline{p} = (p_i)_{i=1}^n$ that has the least relative entropy with respect to $\underline{q}$, $\arg\min_{\underline{p}} \sum_{i=1}^n p_i \log \frac{p_i}{q_i}$, and that satisfies the constraints. The principle of minimum relative entropy is a generalization of the principle of maximum entropy. If the prior distribution $\underline{q}$ is uniform, the relative entropy of $\underline{p}$ with respect to $\underline{q}$ differs from $-H(\underline{p})$ only by a constant. So the principle of maximum entropy is equivalent to the principle of minimum relative entropy with a uniform prior distribution.

satisfying the constraint $C$, and $\underline{p} = (p_j)_{j=1}^n$. Then:

$$\Delta^t P(b'|b) = P^t_C(b'|b) - P^t(b'|b) \tag{10}$$

Constraint $C$ is specified as follows in three cases: first when $c = b$, second when $c' = c \neq b$, and third when $c' \neq c \neq b$.

First, if $c = b$ then $C$ is: $P^t_C(b'|b) = P^t(b'|b) + \nu(1 - P^t(b'|b))$, for $\nu \in [0, 1]$ — the value of $\nu$ represents the strength of $\alpha$'s belief that the probability that $(b'|b)$ will occur at time $t + 1$ should increase if $(b'|b)$ occurs at time $t$.

Second, if $c' = c \neq b$ then constraint $C$ is:

$$P^t_C(b|b) = P^t(b|b) + g_1(b, c)(1 - P^t(b|b))$$

for: $g_1 \in [0, 1]$, where $g_1(b, c)$ represents the strength of $\alpha$'s belief that the probability that $(b|b)$ will occur at time $t + 1$ should increase if $(c|c)$ occurs at time $t$.

Third, if $c' \neq c \neq b$ then suppose that $c'$ is preferred to $c$ by $\alpha$ then $h(c', c) = P^t(\text{Accept}(c')) - P^t(\text{Accept}(c)) > 0$. Let $B(b)^+ = \{x \mid h(x, b) > 0\}$, ie: the set of contract executions that $\alpha$ prefers to $b$. Given a signed contract $b$, the prior probability that the contract execution will be preferred by $\alpha$ to $b$ is: $p(b)^+ = \sum_{x \in B(b)^+} P^t(x|b)$. After observing $(c'|c)$ we wish to increase the probability that a preferred execution will occur for contract $b$ to: $p(b \mid (c'|c))^+ = p(b)^+ + g_2(b, c, c')(1 - p(b)^+)$, where $g_2(b, c, c')$ represents the strength of $\alpha$'s belief that the probability that execution of contract $b$ at time $t + 1$ will be preferred to $b$ should increase if $(c'|c)$ occurs at time $t$. Constraint $C$ then is: $\sum_{x \in B(b)^+} P^t_C(x|b) = p(b \mid (c'|c))^+$. Similarly, if $c'$ is *not* preferred to $c$ by $\alpha$ then construct $B(b)^-$.

## A.2 Updating from preferences

[Debenham, 2004] describes the application of maximum entropy inference to enable $\alpha$ to estimate $P^t(\text{Accept}(\beta, \alpha, \delta))$ the probability that $\beta$ will accept deal $\delta$ from $\alpha$ in response to $\alpha$ transmitting the illocution $\text{Offer}(\alpha, \beta, \delta)$. This distribution is derived from previously observed $\text{Offer}(\beta, \alpha, \dots)$ and $\text{Reject}(\beta, \alpha, \dots)$ illocutions received from $\beta$ — the former indicating readiness to accept and the latter readiness to reject. $\alpha$ may not accept this historic readiness as being definitive now, if so then $P^t(\text{Accept}(\beta, \alpha, \delta))$ is estimated by attaching time-discounted beliefs (as sentence probabilities) to these observations, and then by calculating the maximum entropy distribution subject to those probabilities as constraints.

Suppose that $\alpha$ now receives preference information from $\beta$ in the form of an $\text{Inform}(\beta, \alpha, [info])$ illocution, and is prepared to accept this information into its belief set $\mathcal{B}$ as a belief with sentence probability $p_{info}$ — this probability may decay in time. How will this new information influence $\alpha$'s estimate of $P^t(\text{Accept}(\beta, \alpha, \delta))$? Preference information induces a partial ordering on the set of deals. If deal $\delta_1$ is preferred by $\beta$ to deal $\delta_2$ then: if $\text{Accept}(\beta, \alpha, \delta_2)$ $\alpha$ may conclude to certainty $p_{info}$ that $\text{Accept}(\beta, \alpha, \delta_1)$.

Preference illocutions generally refer to particular issues within deals — e.g. "I prefer red to yellow". In general, "I prefer deals with property $Q_1$ to those with property $Q_2$"

becomes the following constraint on the $P^t(\text{Accept}(\beta, \alpha, \delta))$ distribution:

$$p_{info} = \frac{\sum_{\delta:Q_1(\delta)} p_\delta}{\left(\sum_{\delta:Q_1(\delta)} p_\delta\right) + \left(\sum_{\delta:Q_2(\delta)} p_\delta\right)}$$

the posterior distribution for $P^t(\text{Accept}(\beta, \alpha, \delta))$ is calculated by applying the principle of minimum relative entropy[7] to it subject to this constraint.

The method of representing preference information above is quite general. Although if it is used to represent a preference ordering on an issue such as "$\beta$ prefers to pay less money to more" it generates a set of constraints. If however such a constraint is assumed with $p_{info} = 1$ — ie: if it is represented in the knowledge base $\mathcal{K}$ — then the following device is very economical. [Debenham, 2004] describes the representation of $P^t(\text{Accept}(\beta, \alpha, \delta))$ where $\beta$ is attempting to purchase something for money but with a period of warranty. There $\alpha$ assumes that $\beta$ prefers "less money to more" and "more warranty to less". These two preference orderings are dealt with neatly by estimating instead $P^t(\text{LimAccept}(\beta, \alpha, \delta))$ meaning "$\delta$ is the greatest w.r.t. money and least w.r.t warranty that $\beta$ will accept from $\alpha$".

In this way, quantitative preferences over finite domains will give a finite set of linear constraints (in particular, the device above may be used to great effect when $p_{info} = 1$), and qualitative preferences including conditional preferences also yield a finite set of linear constraints.

## A.3 Updating from social information

Social relationships between agents, and social roles or positions held by agents, introduce a bias, i.e. a constraint, on the admissible probability distributions. A social model can be then a set of constraints introduced in $\mathcal{K}$ that has to be respected by the inference mechanism.

For instance, with respect to *power*, and assuming we model power as a function from agents to real values, we could model a meek agent by adding the following constraint in $\mathcal{K}$ that establishes different degrees of acceptability for proposals according to the power of the proposer:

$$\begin{aligned}&\text{Power}(\beta) > \text{Power}(\gamma) \rightarrow \\ &\qquad P^t(\text{Accept}(\alpha, \beta, \varphi)) > P^t(\text{Accept}(\alpha, \gamma, \varphi))\end{aligned}$$

A similar case can be made for *reputation*, which refers to the institutional endorsement of observed trustworthiness[8]. Power and reputation are different instruments that help an agent to form an *a priori* assessment of the trustworthiness of an unknown opponent, or to modify the assessment of a known one. If $\alpha$ learns that her good friend $\gamma$ has a high opinion of $\beta$ then this may cause $\alpha$ to increase her trust in $\beta$ and to 'tighten up' the distribution $P^t(b', b)$. Likewise, if $\alpha$ learns that $\beta$ has a high reputation in a respected institution. So it is natural to represent reputation as $\text{Reputation}(\Phi, \beta)$ where $\Phi$ is an institution name.

[8]Electronic Institutions [Arcos et al., 2005] warrant, within specific limits, the *bona fides* of the players therein — it is in their interests to report anecdotal evidence of 'good' behaviour beyond those limits.

If $\alpha$ receives information, $\Theta$, such as Reputation$(\Phi, \beta)$ then $\Theta$ will either be a positive influence on $\alpha$'s estimate of $P^t(b', b)$ [written $\Theta^+$], a negative one [$\Theta^-$], or neutral — ie a positive influence on $P^t(b, b)$ [written $\Theta^0$]. If $\alpha$ receives $\Theta^+$ then her estimate of the probability that the execution of contract $b$ will be preferred to $b$ becomes: $p(b \mid \Theta^+)^+ = p(b)^+ + g_3(b, \Theta^+)(1 - p(b)^+)$, where $p(b)^+$ is the prior probability as in Sec. A.1, $g_3(b, \Theta^+)$ represents the strength of $\alpha$'s belief that the probability that execution of contract $b$ at time $t+1$ will be preferred to $b$ should increase given $\Theta^+$ was received at time $t$. $\alpha$ revises this estimate using the principle of minimum relative entropy (Eqn. 9 ) subject to the constraint $C$: $\sum_{x \in B(b)^+} P^t_C(x|b) = p(b \mid \Theta^+)^+$, where $B(b)^+$ is as in Sec. A.1. Similarly, if $\alpha$ receives $\Theta^-$ or $\Theta^0$.

## A.4 A trust model

### A.4.1 Trust as conditional entropy

One way of modelling $\alpha$'s trust on $\beta$ is as one minus the normalised negative entropy of distribution $P^t$. The idea is that the more trust the less dispersion of the expected observations and therefore the closer to 1. In this way we can define the Trust that an agent $\alpha$ has on agent $\beta$ with respect to the fulfilment of a contract $(a, b)$ as:

$$T(\alpha, \beta, b) = 1 + \frac{1}{B^*} \cdot \sum_{b' \in B(b)^+} P^t(b'|b) \log P^t(b'|b)$$

where $B(b)^+$ is the set of contract executions that $\alpha$ prefers to $b$ as defined in Sec. A.1, $B^* = 1$ if $|B(b)^+| = 1$ and $\log |B(b)^+|$ otherwise, and $\beta$ has agreed to execute $b$, and $\alpha$ systematically observes $b'$, for some $b'$ that $\alpha$ does not prefer to $b$, the trust value will be 0. Trust will tend to 0 when the dispersion of observations is maximal.

And, as a general measure of $\alpha$'s trust on $\beta$ we naturally use the normalised negative conditional entropy of executed contracts given signed contracts:

$$T(\alpha, \beta) = 1 + \frac{\sum_{b \in B} \left[ P^t(b) \cdot \sum_{b' \in B(b)^+} P^t(b'|b) \log P^t(b'|b) \right]}{B^* \cdot \sum_{b \in B} P^t(b)}$$

This formulation of trust is useful when any variation from the agreed contract is undesirable.

### A.4.2 Trust as relative entropy

We are usually happier, and ready to trust more, if the actual execution of a contract goes in the direction of our increasing preference. We capture this idea using as a model for trust the relative entropy[9] between the probability distribution of acceptance and the

[9]Otherwise called *cross entropy* or the *Kullback-Leibler distance* — although it is not reflexive and so it is not a metric. See also Footnote 7.

distribution of the observation of contract execution. That is:

$$T(\alpha, \beta, b) = 1 - \sum_{b' \in B(b)^+} P^t(b') \log \frac{P^t(b')}{P^t(b'|b)}$$

and, similarly

$$T(\alpha, \beta) = 1 - \sum_{b \in B} P^t(b) \left[ \sum_{b' \in B(b)^+} P^t(b') \log \frac{P^t(b')}{P^t(b'|b)} \right]$$

Finally, the trust we place in an agent is useful to determine which agent to prefer in order to accept proposals. That is, trust is useful to assess the distribution of probability $P(\text{Accept}(\alpha, \beta, \delta)|\text{Offer}(\beta, \alpha, \delta))$. What trust does to this distribution is to impose constraints on its values. As follows:

$$P^t(\text{Accept}(\alpha, \beta, \delta)|\text{Offer}(\beta, \alpha, \delta)) > P^t(\text{Accept}(\alpha, \gamma, \delta)|\text{Offer}(\gamma, \alpha, \delta)) \text{ if } T(\alpha, \beta) > T(\alpha, \gamma).$$

e.g. I prefer the same deal from my usual butcher than from someone I trust less.

# B ReGreT

ReGreT is a modular trust and reputation model oriented to complex e-commerce environments where social relations play an important role. Figure 11 shows the structure of the ReGreT system.

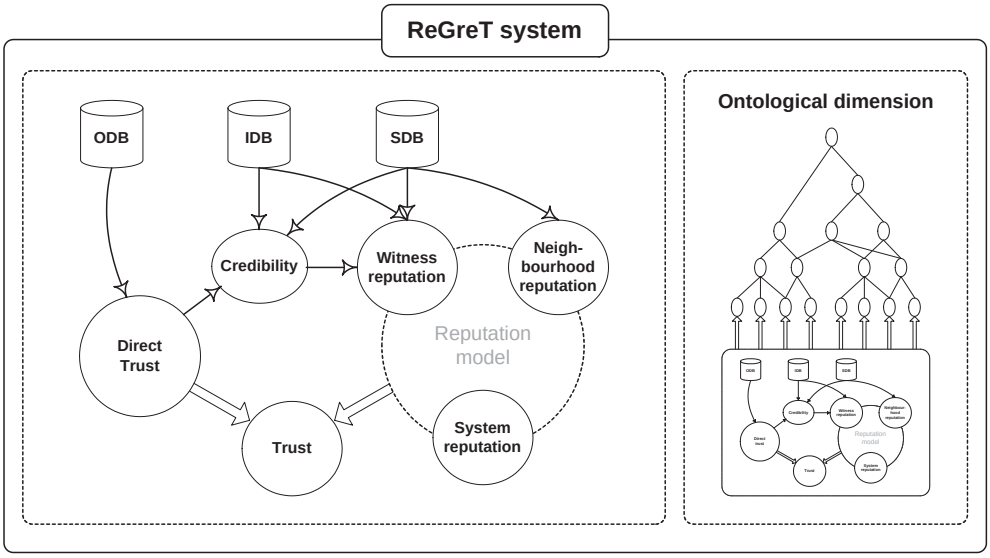



Figure 11: The ReGreT system

The system maintains three knowledge bases. The outcomes data base ($ODB$) to store previous contracts and their result; the information data base ($IDB$), that is used as a container for the information received from other partners and finally the sociograms data

base (*SDB*) to store the sociograms that define the agent social view of the world. These data bases feed the different modules of the system.

The *direct trust* module deals with direct experiences and how these experiences can contribute to the trust on third party agents. Together with the reputation model they are the basis for the trust model.

The reputation model is divided in three specialized types of reputation depending on the information source that is used to calculate them. If the reputation is calculated from the information coming from witnesses we talk about the *witness reputation*, if the reputation is calculated using the information extracted from the social relations between partners we are talking about the *neighbourhood reputation*. Finally, reputation based on roles and general properties is modelled by the *system reputation*.

The system also incorporates a credibility module that allows the agent to measure the reliability of witnesses and their information. This module is extensively used in the calculation of *witness reputation*.

All these modules work together to offer a complete trust model based on direct knowledge and reputation. However, the modular approach in the design of the system allows the agent to decide which parts it wants to use. For instance, the agent can decide not to use *neighbourhood reputation* to calculate a reputation value or rely only on *direct trust* to calculate the trust on an agent without using the reputation module.

Another advantage of this modular approach is the adaptability that the system has to different degrees of knowledge. The system is operative even when the agent is a newcomer and it has an important lack of information. As long as the agent increases its knowledge about the other members of the community and the social relations between them, the system starts using other modules to improve the accuracy of the trust and reputation values. This allows the system to be used in a wide range of scenarios, from the most simple to the most complex. If the information is available, the system will use it.

In the ReGreT system, each trust and reputation value has an associated reliability measure. This measure tells the agent how confident the system is on that value according to how it has been calculated. Thanks to this measure, the agent can decide, for example, if it is sensible or not to use the trust and reputation values as part of the decision making mechanism.

The last element in the ReGreT system is the *ontological structure*. We consider that trust and reputation are not single and abstract concepts but rather multi-facet concepts. The *ontological structure* provides the necessary information to combine reputation and trust values linked to simple aspects in order to calculate values associated to more complex attributes. For example, the reputation of being a good flying company summarizes the reputation of having good planes, the reputation of never losing luggage and the reputation of serving good food. In turn, the reputation of having good planes is a summary of the reputation of having a good maintenance service and the reputation of frequently renewing the fleet. Note that each individual can have a different *ontological structure* to combine trust and reputation values and a different way to weigh the importance of these values when they are combined.

# C Repage

Repage is a reputation model based on the cognitive theory of Conte and Paolucci [Conte and Paolucci, 2002]. The main point behind this theory is the distinction between *Image* and *Reputation*. Although both are social evaluations, image and reputation are distinct objects. Image is an evaluative belief [Miceli and Castelfranchi, 2000]; it tells that the target is "good" or "bad" with respect to a norm, a standard, or a skill. Reputation is a belief about the existence of a communicated evaluation. Consequently, to assume that a target $t$ is assigned a given reputation implies only to assume that $t$ is reputed to be "good" or "bad", i.e., that this evaluation circulates, but it does not imply to share the evaluation. Repage provides evaluations on potential partners and is fed with information from others and outcomes from direct experience.

To select good partners, agents need to form and update own social evaluations; hence, they must exchange evaluations with one another. If agents transmit only believed image, the circulation of social knowledge would be bound to stop soon. But in order to preserve their autonomy, agents need to *decide* whether to share or not others' evaluations of a given target. If agents transmit others' evaluations as if these evaluations were their own, without the possibility of choosing if they want to mix both types of evaluations or not, they would be no more autonomous. Hence, they must

- form both evaluations (image) and meta-evaluations (reputation), keeping distinct the representation of own and others' evaluations, before
- deciding whether or not to integrate reputation with their own image of a target.

Unlike current systems, given what we have said above, in Repage reputation does not coincide with image. Indeed, others can either transmit their own image of a given target, which they hold to be true, or report on what they have "heard" about the target, i.e. its reputation, whether they believe this to be true or not. Of course, in the latter case, they will neither commit to the information truth value nor feel responsible for its consequences. Consequently, agents are expected to transmit uncertain information, and a given positive or negative reputation may circulate over a population of agents even if its content is not actually shared by the majority.

The main element of the Repage architecture is the memory that is composed by a set of *predicates*. Predicates are objects containing a social evaluation, belonging to one of the main types accepted by Repage (image, reputation, shared voice, shared evaluation), or to one of the types used for their calculation (valued information, evaluation related from informers, and outcomes). These predicates have a tuple of five numbers to represent the evaluation plus a strength value that indicates the confidence the agent has on this evaluation. Predicates are conceptually organized in different levels and interconnected to reflect their dependencies. To have a pictorial summary of the kinds of predicates in the Repage's memory, and of their relative position, refer to figure 12; for a detailed description, confront [Sabater et al., 2006].

Predicates are connected by a network of dependencies, that specifies which predicates contribute to the values of other ones. Each predicate in the *Repage* memory has a set of antecedents and a set of consequents. If an antecedent is created, removed, or changes

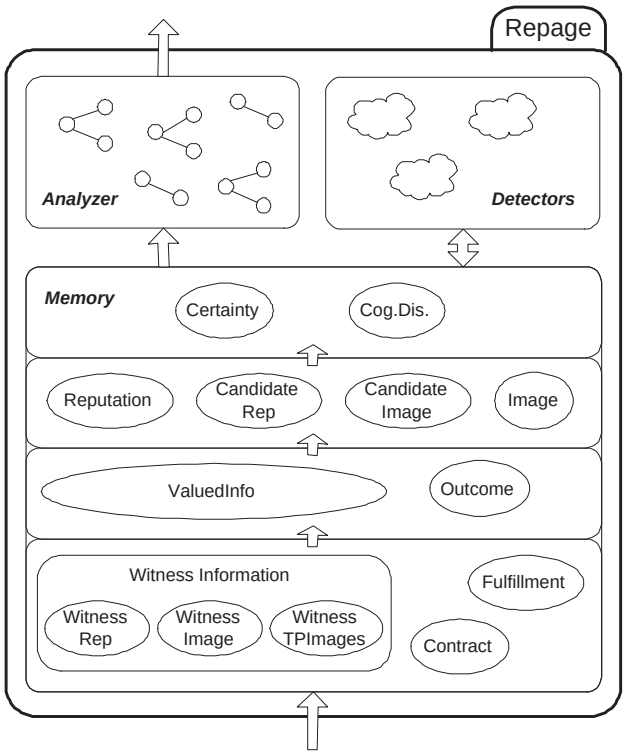



Figure 12: *Repage* architecture as a module for an intelligent agent. In the Memory, several kinds of predicates are organized in different levels. The detectors build new predicates on the basis of the existing ones. The analyzer give indications to the rest of the system

its value, the predicate is notified, recalculates its value and notifies the change to its consequents.

## C.1 On Representations and Uncertainty in RepAge

Agents must keep track of social evaluations. This means that they need the content of the evaluations - in simple systems, a position on a scale going from good to bad. Thus, we could simply associate to a predicate a scalar number.

An interesting question arise when we wonder what is the correspondent of the intuitive concept of an uncertain evaluation. For a number on a scale, the only available representation is the middle of the scale. But this representation will have multiple semantics, and the uncertainity will get mixed with actual regular middle outcomes.

An alternative can be provided by the usage of a *labeled tuple*, that is, a tuple of five numbers each of which has an associated label in a rating scale, for example from "very bad" to "very good". A labeled tuple is more expressive and allow us two dimensions of uncertainity - that is, a "spike" in the middle of the scale or a "flat" evaluation. This expressiveness allows us to distinguish a target that is constantly mediocre (the spike) from one that is actually unpredictable or uncertain (the neutral or flat evaluation).

There are several contexts where this is important. Consider for example one in which some of the classes are extremely undesirable; in the case of evaluations about the respect

of a norm, an agent could have a goal to avoid altogether the possibility of meeting with "very bad" agents - in this case, it would be important to distinguish the flat evaluation, that does not deny the possibility for a"very bad" encounter, from the spike in the middle, that excludes it. The same kind of considerations could be done for the case in which the middle value is actually important.

Thus, we argue that oversimplistic representations of social evaluations - of the kind a number on a scale - are insufficient to represent situations that are cognitively plausible. However, not even the labeled representation proposed until now is suitable as the representation of a social evaluation - the predicate content - in a model of a cognitive and social mind as Repage is.

There is another ingredient that must be added, that is, the strength of belief of the agent holding the representation in the representation itself. This last ingredient is indispensable to cover for yet another level of uncertainty. In fact, no model of reputation is worthy of its name if it does not accounts for the social transmission of evaluations.

But to consider reported information requires the ability to accept information "with reserve". To this end, we added a scalar value as the strength of belief that the holder has in the evaluation, similar to the one in the ReGreT[Sabater, 2003] model. This strength value will drive the aggregation process; an evaluation with a large strength represent a certain belief, that will preserve its shape (intended as the relation between weights) if combined with another, less strong evaluation. As an example, consider an aggregation of many evaluations, composed by a large majority that shows a consensus (very similar or identical shape) plus some outliers; all of them with the same, not very large, strength. The aggregation of the similar group will produce a very strong evaluation. When adding the outliers, the strong shape will remain more or less unchanged.

In Repage, this strength value is a function of (i) the strength of its antecedents and of (ii) some special characteristics intrinsic to that type of predicate. For instance, the strength of an *Image* is a function of the strengths of the antecedents (outcomes, information from third party agents and their image or reputation as witnesses, ...) but also of the number of these antecedents.

We maintain the content of a predicate as a tuple of five numbers (summing to one) plus a strength value. Each number has an associated label in the rating scale: very bad (vb), bad (b), neutral (n), good (g) and very good (vg). We call this representation a *weighted labeled tuple*.

In mathematical terms, we represent this tuple as $[w_{vb}, w_b, w_n, w_g, w_{vg}]$, or , for short, $[w_1, w_2, ..., w_5]$, where $w_1$ corresponds to very bad and $w_5$ to very good. The sum of the five components is fixed to 1. In addition, we have a single value indicating the strength of belief in the evaluation, a number $s \in [0, 1]$. In the following, we will express this evaluation as the tuple $\{[w_1, w_2, \ldots, w_5], s\}$. We will sometime refer to the set of the weights as the *shape* of the weighted labeled tuple.

In figure 13, we show some examples. Example a) shows an evaluation that says the agent is 0.8 sure (almost completely certain) that the behavior of the agent evaluated is usually very bad or sometimes bad. In b) the evaluation describes a behavior that is always very good or very bad (a black or white behavior with no grey) although the evaluator is quite unsure about it ($s = 0.2$). In c) the evaluator is saying it is completely certain ($s = 1.0$) that the behavior of the agent is unpredictable. Notice the difference

between c) and a situation where the strength is 0. When the evaluation has a strength of 0, the evaluator is saying it doesn't know anything about the agent that is being evaluated and that the shape of the membership is meaningless.

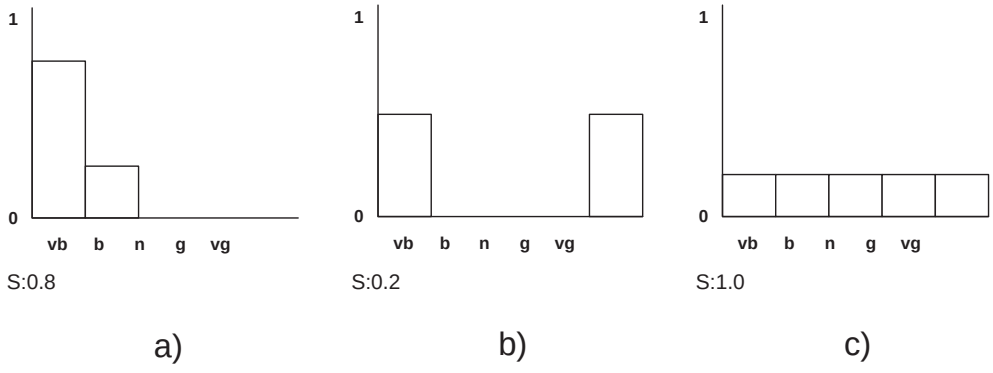



Figure 13: Examples of evaluations in Repage

The expressiveness of this representation is clear in examples like b) or c) that would be impossible to reproduce using two single real values. Intuitively, this choice for the predicate adds more levels of freedoms and allows for subtler representations. But although the examples presented seem to make perfect sense by themselves, they actually leave space for different interpretations. Are the weights probability to obtain the result corresponding to their label? Or do they express simply a possibility? There is a large body of literature about the subtelties in the representations of uncertainty; see for example [Philippe, 1998], where the author distinguishes between probability measure, possibility measure, and belief measures; our approach falls in this last area, even if it is much more practical and not based on a possible worlds logic. Indeed, we are particularly interested in the case in which the weights express the strength of belief of a decision maker in the corresponding outcome, as proposed and discussed in [Yager, 2004a], that refers to the general area of decision making under uncertainty (see also [Yager, 2004b]). Even if these points of view seem very similar, when used as a base for designing the aggregation algorithm they appear to be very different.

In the first case, the *probabilistic* approach allows the aggregation method to be simple and not demanding from a computational perspective. To the contrary, the second approach, that we will call the *strength of belief* one, will be the base for more complex considerations and a large amount of subtleties, resulting in an improved expressive power at the expenses of computational simplicity. For an extensive discussion about this we refer to [Sabater-Mir and Paolucci, 2007].

# D The CREDIT Model

Let $Ag$ be the society of agents noted as $\alpha, \beta, ... \in Ag$. A particular group of agents is noted as $G \subseteq Ag$ and each agent can only belong to one group. $\mathcal{T}$ denotes a totally ordered set of time points (sufficiently large to account for all agent interactions) noted as $t_0, t_1, \ldots$, such that $t_i > t_j$ if and only if $i > j$.

## D.1 Contracts

Contracts are agreements about issues and the values these issues should have. Let $X = \{x, y, z, \ldots\}$ be the set of potential issues to include in a contract, and the domain of values taken by an issue $x$ be noted as $D_x$. We will note that issue $x$ takes the value $v \in D_x$ as $x = v$. Thus, a particular contract, $O$, is an arbitrary set of issue-value assignments noted as $O = \{x_1 = v_1, x_2 = v_2, ..., x_n = v_n\}$ where $x_i \in X$, $v_i \in D_{x_i}$, and $O \in \mathcal{O}$ which denotes the set of potential contracts. We will also note the set of issues involved in a contract $O$ as $X(O) \subseteq X$. Given an agreed contract, two or more agents all have a subset of the contract to enact. Each subset of the contract allocated to an agent is superscripted by the respective agent identifier such that, for example, in a contract $O$ between $\alpha$ and $\beta$, $O^\alpha \cup O^\beta = O$. An agent, $\alpha$, has a utility function for contracts, noted as $U^\alpha : \mathcal{O} \rightarrow [0, 1]$, and for each issue $x \in X(O)$ in a contract noted as $U^\alpha_x : D_x \rightarrow [0, 1]$. In CREDIT, we will define the utility of a contract, for an agent, as an aggregation of the weighted utilities of the individual issues as shown below (note this assumes that issues are independent):

$$U^\alpha(O) = \sum_{x \in X(O)} \omega_x \cdot U^\alpha_x(v_x) \tag{11}$$

where $\sum \omega_x = 1$ and $v_x \in D_x$ is the value taken by the issue $x \in X(O)$.

## D.2 Confidence

We will define confidence as follows:

> $\alpha$'s confidence in an issue $x$ handled by $\beta$ is a measure of certainty (leading to trust), based on evidence from past direct interactions with $\beta$, which allows $\alpha$ to expect a given set of values to be achieved by $\beta$ for $x$.

In CREDIT, the behaviour of an agent regarding the fulfillment of an issue in a contract is perceived in terms of the variations on utility between the signed value for the issue and the enacted one. These utility variations are then sensed over multiple interactions to build up a picture of the agent's performance over time. In CREDIT we take the stance that fuzzy sets have their domains specified over 'absolute' variations on utility, rather than on relative variations. Thus, we consider that $\Delta U \in [-1, 1]$ (recall that utility values belong to the interval $[0, 1]$).

Specifically, we assume that agents share a (small) set $\mathcal{L} = \{L_1, L_2, \ldots, L_k\}$ of linguistic labels to qualify the performance of an agent on each issue. For instance, $\mathcal{L} = \{Bad, Average, Good\}$. We believe these labels provide an adequate means for an agent to express its view on the *possible* (approximate) utility deviations, gains or losses, in the executed contract with respect to the utility of the contractually signed values. For example, each agent could understand the labels 'Bad', 'Average', and 'Good' for the issue 'delivery' in different ways according to their ontology (as shown in table 4). As this shows, each agent can have a different ontology to qualify variations between the contracted values and the executed value. However, we do require that the common terms have the same agreed upon interpretation among the agents in order to permit a meaningful communication of reputation values (see section D.3). Thus, using table 4,

| Label / Agent | $\alpha$ | $\beta$ | $\gamma$ |
|---|---|---|---|
| *Bad* | Late | Very Late | Too late |
| *Average* | On time | Just in time | Right time |
| *Good* | Early | Very early | Early enough |

Table 4: Table showing the possible different meanings of the labels for 3 agents when applied to the issue 'delivery'.

agent $\alpha$ can translate a 'Very Late' rating from agent $\beta$ as *Late* (since they both equate to '*Bad*') and 'Right time' from $\gamma$ as 'On time' (since they both equate to '*Average*'). In more detail, we model the meaning of a label $L$ by a fuzzy set on the domain of utility deviations $\Delta U = [-1, 1]$, specified by its membership function $\mu_L(u) : [-1, 1] \rightarrow [0, 1]$.

Thus, agent $\alpha$'s confidence level is defined as the membership level, measured over $[0, 1]$, of the behaviour of a particular agent $\beta$ with respect to an issue $x$ to a linguistic term $L$, noted as $C(\beta, x, L)$. Therefore, the cut of the fuzzy set defined by $C(x, L)$ represents a range (on the horizontal axis) of values:

$$E\Delta U_c(x, L) = \{\delta u \in [-1, 1] \mid \mu_L(\delta u) \geq C(x, L)\} \tag{12}$$

that is understood as the range of expected utility deviations at execution time on issue $x$ by agent $\beta$. For instance, $\alpha$ may express its belief that $\beta$ is 'Good' to a confidence level 0.6 in fulfilling the contractual values on price, 'Average' to a level of 0.25, and 'Bad' to a level of 0. This would mean that $\alpha$ expects the utility deviation to lie within the range of values which support the confidence level of 0.6 for 'Good', 0.25 for 'Average', and 0 for 'Bad'.

### D.2.1 Evaluating Confidence

Given a a proposed (not yet agreed) contract $O$, for each issue $x$ in $X(O)$, we can estimate, from the history of past interactions, a probabilistic distribution $P$ of $\alpha$'s utility variation $\Delta U_x \in [-1, 1]$ (negative or positive) relative to issue $x$. Values of $\Delta U_x$ correspond to the possible differences between the utility $U_x(v)$ of the agreed value $(x = v) \in O$ and the utility $U_x(v')$ of the (unknown) final value $(x = v')$ in the executed contract $O'$ (i.e. $\Delta U_x = U_x(v) - U_x(v')$). Then we can say that the agent $\alpha$ has a certain *risk* with issue $x$ when it estimates that $1 \geq q > 0$ where $q$ is the probability that $\Delta U_x < 0$. Of course, the more positive the mean, $\overline{\Delta U}_x$, of this probability distribution (i.e. the higher the expected utility loss), the higher the risk, and the more positive this mean is, the lower the risk (i.e. the lower the expected utility loss).

Now, assume we have a probability distribution $P$ for $\Delta U_x$. In order to determine confidence levels $C(x, L)$ we initially need to determine a significantly representative interval $[\delta_1, \delta_2]$ for $\Delta U_x$ (e.g. such that the probability that $(\delta_1 \leq \overline{\Delta U}_x \leq \delta_2)$ is equal to 0.95).

Finally, to calculate confidence levels $C(x, L)$ for each label $L \in \mathcal{L}$, we want the interval $[\delta_1, \delta_2]$ to coincide as much as possible with the set of expected values $E\Delta U_c(x)$ as computed in equation 12. Since this range is defined by the confidence levels of its

limits, the procedure amounts to selecting the minimum confidence levels of the two limits for that label as shown in equation 13.

$$C(x, L) = \min(\mu_L(\delta_1), \mu_L(\delta_2)) \tag{13}$$

## D.3 Reputation

An agent's reputation is the perception of a group or groups of agents in the society about its abilities and attributes:

> $\alpha$'s estimate of $\beta$'s reputation in handling an issue $x$ is $\alpha$'s measure of certainty (leading to trust), based on the aggregation of confidence measures (for $x$) provided to it by other agents which have previously interacted with $\beta$, which allows $\alpha$ to expect a given set of values to be achieved by $\beta$ for $x$

Hence, we assume that an agent $\alpha$ possesses a function $Rep : Ag \times X \times \mathcal{L} \rightarrow [0, 1]$ where $Rep(\beta, x, L)$ represents the reputation of an agent $\beta$ in handling issue $x$ with respect to the qualifying label $L$. We also assume that the labels $L \in \mathcal{L}$ have their domain specified over the same range of utility deviations (i.e. $\Delta U \in [-1, 1]$).

## D.4 Combined Confidence and Reputation Measures

We propose to define the threshold $\kappa$ as $\kappa = \max(1, |CB_{\alpha,\beta}|/\theta_{min})$, where $|CB_{\alpha,\beta}|$ is the number of interactions of $\alpha$ with $\beta$ and $\theta_{min}$ is the minimum number of interactions (successful negotiations and completed executions above which only the direct interaction is taken into account [Sabater and Sierra, 2002b]. Thus, we capture the combination of confidence and reputation measures through the function $CR : Ag \times X \times \mathcal{L} \rightarrow [0, 1]$, which is, in the simplest case, a weighted average of both kinds of degrees (as in the previous cases we omit references to the agent whenever possible):

$$CR(x, L) = \kappa \cdot C(x, L) + (1 - \kappa) \cdot Rep(x, L), \tag{14}$$

Given $CR$ levels it is then possible to compute the expected values for an issue $x$ and label $L$ as:

$$E\Delta U_{cr}(x, L) = \{u \mid \mu_L^x(u) \geq CR(x, L)\} \tag{15}$$

and then the intersection of the expected ranges for all the labels $L \in \mathcal{L}$:

$$E\Delta U_{cr}(x) = \bigcap_{L \in \mathcal{L}} E\Delta U_{cr}(x, L) \ . \tag{16}$$

As can be seen, the above range is defined in terms of the utility deviations rather than in terms of the values that the issue could take. However, at negotiation time, for example, we might need to compute the expected values an issue could take, after execution of the contract, given an offered value $v_0$ for the issue. This requires transferring the expected utility deviations to the domain of the issue considered. This can be computed in the following way:

$$EV_{cr}(x, v_0) = \{v \in D_x \mid U_x(v) - U_x(v_0) \in E\Delta U_{cr}(x)\} \tag{17}$$

## D.5 Trust

In CREDIT we use the combined degrees $\{CR(x, L)\}_{L\in\mathcal{L}}$, as given by equation 14, to define the interval of expected values $E\Delta U_{cr}(x)$, that provides us with a maximum expected loss in utility $\Delta^{cr}_{loss} = \sup(E\Delta U_{cr}(x))$. This maximum expected utility loss represents the risk that is involved in the interaction given knowledge acquired both from direct interactions and reputation and also from the norms of the environment. While the risk describes how much we expect to lose from an interaction, trust is the opposite of this given our initial definition. Thus we define trust as:

$$T(\alpha, \beta, x) = \min(1, 1 - \Delta^{cr}_{loss}) \tag{18}$$

where $T$ serves to describe trust in $\beta$ for issue $x$ based on both confidence in $\beta$ and its reputation with respect to issue $x$.

Here, we choose to bound trust values in the range $[0, 1]$ where 0 represents a completely untrustworthy agent (and corresponds to the maximum possible utility loss) and 1 represents a completely trustworthy agent (and corresponds to zero utility loss).

In any case, we can now define the trust $T(\alpha, \beta, X(O))$ of an agent $\alpha$ in an agent $\beta$ over a particular set $X(O) = \{x_1, ..., x_k\}$ of issues appearing in the contract $O$ (or in the expanded one $O_+$) as an aggregation of the trust in each individual issue (e.g. trust in delivering on time, paying on time and the product having the quality specified in the contract). That is, we postulate:

$$T(\alpha, \beta, X(O)) = agg(T(\alpha, \beta, x_1), ..., T(\alpha, \beta, x_k)) \tag{19}$$

where $agg : [0, 1]^k \to [0, 1]$ is a *suitable* aggregation function. If some issues are considered to be more important than others, the aggregation function should take this into consideration. This can be achieved by means of different weights given for each issue $x_i \in X(O)$ (the higher the weight, the more important the issue). A typical choice would be to take the aggregation function as a weighted mean:

$$T(\alpha, \beta, X(O)) = \sum_{x_i\in X'} w_i \cdot T(\alpha, \beta, x_i) \tag{20}$$

where $\sum w_i = 1$ and $0 \leq w_i \leq 1$.

# References


[Arcos et al., 2005] Arcos, J. L., Esteva, M., Noriega, P., Rodríguez, J. A., and Sierra, C. (2005). Environment engineering for multiagent systems. *Journal on Engineering Applications of Artificial Intelligence*, 18.

[Avesani et al., 2005] Avesani, P., Giunchiglia, F., and Yatskevich, M. (2005). A large scale taxonomy mapping evaluation. In *Proceedings of International Semantic Web Conference (ISWC)*, pages 67–81.

[Chen, 2001] Chen, W. (2001). New algorithm for ordered tree-to-tree correction problem. *J. Algorithms*, 40(2):135–158.

[Conte and Paolucci, 2002] Conte, R. and Paolucci, M. (2002). *Reputation in artificial societies: Social beliefs for social order*. Kluwer Academic Publishers.

[Debenham, 2004] Debenham, J. (2004). Bargaining with information. In Jennings, N., Sierra, C., Sonenberg, L., and Tambe, M., editors, *Proceedings Third International Conference on Autonomous Agents and Multi Agent Systems AAMAS-2004*, pages 664 – 671. ACM.

[Dorigo and Stützle, 2004] Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. MIT Press, Cambridge, MA.

[Falcone and Castelfranchi, 2001] Falcone, R. and Castelfranchi, C. (2001). The socio-cognitive dynamics of trust: Does trust create trust? In *Proceedings of the workshop on Deception, Fraud, and Trust in Agent Societies*, pages 55 – 72.

[Giunchiglia and Shvaiko, 2003] Giunchiglia, F. and Shvaiko, P. (2003). Semantic matching. *The Knowledge Engineering Review*, 18(3):265–280.

[Giunchiglia et al., 2004] Giunchiglia, F., Shvaiko, P., and Yatskevich, M. (2004). S-Match: an algorithm and an implementation of semantic matching. In *Proceedings of the European Semantic Web Symposium (ESWS)*, pages 61–75.

[Giunchiglia and Walsh, 1992] Giunchiglia, F. and Walsh, T. (1992). A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–390. Also IRST-Technical Report 9001-14, IRST, Trento, Italy.

[Jaynes, 2003] Jaynes, E. (2003). *Probability Theory — The Logic of Science*. Cambridge University Press.

[MacKay, 2003] MacKay, D. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.

[Magnini et al., 2004] Magnini, B., Speranza, M., and Girardi, C. (2004). A semantic-based approach to interoperability of classification hierarchies: Evaluation of linguistic techniques. In *Proceedings of the International Conference on Computational Linguistics (COLING)*.

[Miceli and Castelfranchi, 2000] Miceli, M. and Castelfranchi, C. (2000). *Human cognition and agent technology*, chapter The Role of Evaluation in Cognition and Social Interaction. Amsterdam:Benjamins.

[Miller, 1995] Miller, G. (1995). WordNet: A lexical database for english. *Communications of the ACM*, 38(11):39–41.

[periklis, 2006] periklis (2006). H. Syed, P. Andritsos. Weigh your Preferences! Towards using hierarchies for building personal libraries. Technical Report, University of Trento.

[Philippe, 1998] Philippe, S. (1998). Probability, possibility, belief: Which and where? In Smets, P., editor, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, pages 1–24. Kluwer, Doordrecht.

[Ramchurn et al., 2005] Ramchurn, S., Sierra, C., Godo, L., and Jennings, N. (2005). Devising a trust model for multiagent interactions using confidence and reputation. *International Journal of Applied Artificial Intelligence*, 18(9–10):91–204.

[Sabater, 2003] Sabater, J. (2003). *Trust and reputation for agent societies*. PhD thesis, Universitat Autonoma de Barcelona (UAB).

[Sabater et al., 2006] Sabater, J., Paolucci, M., and Conte, R. (2006). Repage: Reputation and image among limited autonomous partners. *Journal of Artificial Societies and Social Simulation*, 9(2).

[Sabater and Sierra, 2002a] Sabater, J. and Sierra, C. (2002a). Reputation and social network analysis in multi-agent systems. In *Proceedings of the First International Conference on Autonomous Agents and Multi-Agent systems*, pages 475 – 482.

[Sabater and Sierra, 2002b] Sabater, J. and Sierra, C. (2002b). Reputation and social network analysis in multi-agent systems. In *Proceedings of the first international joint conference on autonomous agents and multiagent systems (AAMAS-02), Bologna, Italy*, pages 475—482.

[Sabater-Mir and Paolucci, 2007] Sabater-Mir, J. and Paolucci, M. (2007). On representation and aggregation of social evaluations in computational trust and reputation models. *International Journal of Approximate Reasoning*, In press.

[Tai, 1979] Tai, K. (1979). The tree-to-tree correction problem. *J. ACM*, 26(3):422–433.

[Valiente, 2002] Valiente, G. (2002). *Algorithms on Trees and Graphs*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Wang et al., 1998] Wang, J., Shapiro, B., Shasha, D., Zhang, K., and Currey, K. (1998). An algorithm for finding the largest approximately common substructures of two trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8):889–895.

[Wang and Zhang, 2001] Wang, J. T.-L. and Zhang, K. (2001). Finding similar consensus between trees: an algorithm and a distance hierarchy. *Pattern Recognition*, 34(1):127–137.

[Yager, 2004a] Yager, R. (2004a). On the determination of strength of belief for decision support under uncertainity-part i: generating strength of belief. *Fuzzy Sets and Systems*, 1:117–128.

[Yager, 2004b] Yager, R. (2004b). Uncertainity modeling and decision support. *Reliability Engineering and Systems Safety*, 85:341–354.

[Yolum and Singh, 2003] Yolum, P. and Singh, M. (2003). Achieving trust via service graphs. In *Proceedings of the Autonomous Agents and Multi-Agent Systems Workshop on Deception, Fraud and Trust in Agent Societies*. Springer-Verlag.

[Zhang et al., 1992] Zhang, K., Statman, R., and Shasha, D. (1992). On the editing distance between unordered labeled trees. *Inf. Process. Lett.*, 42(3):133–139.