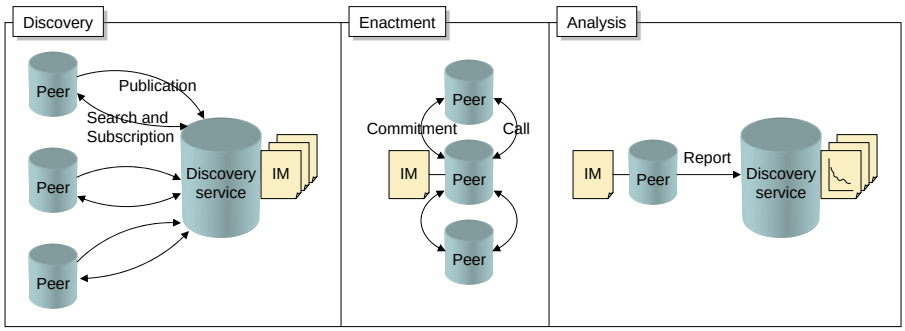

Figure 1: The discovery-enactment-analysis cycle

Webs of Interactions: Exploring Peer Ranking via Simulation in OpenKnowledge

Dave Robertson

The effort to share knowledge on a large scale has concentrated on formalisation of knowledge. Meta-data describe properties of databases. Interface specifications describe the input/output data of services. Tags describe features of documents and images. The ideal being pursued is, as far as possible, to describe knowledge independent of the tasks for which that knowledge might be deployed. Then we can devise new ways of deploying knowledge without having to re-assess how it is encoded. In practice, we seldom come close to an ideal knowledge representation in real domains of application (many would argue that we never could) but for some tasks this may not matter because, like the conventional Web, enough human involvement makes shallow knowledge representation effective. If we can develop a plausible ontology for a document collection that hitherto had been accessed only by keyword search then we are likely to improve accuracy of document recall and, since people can check each result, we have a natural human buffer against inaccuracy. This symbiotic relationship has proved more difficult to achieve, however, when our knowledge sources are complex programs and our tasks involve coordinating them.

Why is it more difficult to share knowledge for coordinated tasks involving programs? One reason is that it *is* coordinated, so the sequence in which the programs communicate influences how each will interpret knowledge communicated to it. A second reason is that programs have local state, so the program with which we interacted previously may not be quite the same when we interact with it again. A third reason is that ontologies are sensitive to task, so the form of words used in one form of collaboration may be different from that used in another, even though some of the raw knowledge being applied is the same. Given these problems, one might think that the only appropriate response is to adopt a service oriented architecture and simply assemble collections of programs using conventional software engineering methods. This, unfortunately, loses the symbiosis that we have with less brittle forms of knowledge sharing since the interaction between programs becomes buried within the programs themselves.

On the OpenKnowledge project, we invented a novel way of tackling this problem by treating specifications of interactions as knowledge to be shared. These can then be published and discovered; enacted to provide coordination; and analysed for their effectiveness, with the results of this analysis influencing human choices in discovery. In order to make as few assumptions as possible about the nature of programs being coordinated, we have adopted a peer to peer architecture in which each peer can, potentially, provide the same general functionality in terms of its ability to discover and coordinate interactions. Figure 1 illustrates this process. First, each peer must download the kernel system (a lightweight, 18 megabyte, Java downoad from `www.openk.org`). This provides it with the facilities to perform the necessary coordination functions, plus automatic connection to a default peer to peer framework (currently based on Bamboo). Through the kernel, a person can publish interaction models (with associated tags to assist discovery) and can also perform keyword searches for interactions (analogously to conventional Web page search but in this case via the peer network). This is assisted by a discovery service (playing a role analogous to a conventional Web search engine, although in this case using the peer network). If the person finds an interaction that appears useful then he or she can subscribe to an appropriate role in the interaction - for example if the interaction coordinated a negotiation of goods for sale then it might have a vendor role and a seller role so a person would choose one of those roles depending on his or her objective in the interaction. Once an interaction is fully subscribed (this can occur at any time for multiple interactions over the peer network) it proceeds to an enactment phase in which the subscribed peers commit to that instance of the interaction and coordination of that specific instance is allocated to a peer on the network (the choice of peer is irrelevant since it merely supplies the CPU cycles). The coordinating peer uses the interaction specification to control the sequence of message passing between peers and the structure of message content, via constraints delegated to appropriate peers in the interaction. Upon completion (or failure) of an interaction the coordinating peer sends a report on the interaction to the discovery service. This can be used for an analysis of the performance of peers in the context of different types of interaction, giving a means of ranking them analogously to the ranking of Web pages by a search engine.

```
a(r1, A) ::
    m1 => a(r2, B) then
    m2 <= a(r2, B).

a(r2, B) ::
    m1 <= a(r1, A) then
    m2 => a(r1, A) <-- c.
```

An interaction model in LCC is a set of clauses each of the form `R :: D`, where `R` denotes the role in the interaction and `D` is the definition of the role. Roles are of the form `a(T,P)`, where `T` gives the type of role and `P` is an identifier for the individual peer undertaking that role. The definition of performance of a role is constructed using combinations of the sequence operator (`then`) or choice operator (`or`) to connect messages and changes of role. Messages are either outgoing to another peer in a given role (`=>`) or incoming from another peer in a given role (`<=`). Message input/output or change of role can be governed by boolean constraints which may be conjunctive or disjunctive. Constraints can be satisfied via shared components registered with `www.openk.org`, so that complex (possibly interactive) solving methods can be shared along with interaction models; or they can be calls to services with private data and reasoning methods. Variables begin with an upper case character.

Figure 2: A basic interaction model in LCC, with syntax explained

1 Interactions are Shared Documents

We are accustomed, thanks to the Web, of thinking of documents and information services as sources of knowledge, while the process of confederating knowledge or coordinating services is a procedural task consigned to programs. A traditional Web browser is able to interpret the structure of a Web page (or form) expressed in a standard language (such as HTML or XML). An important engineering feature of such languages is that they capture the essential notions necessary for communicating some type of information (in the case of the Web, document information) and allow complex representations to be built from the elementary expresisons of the language. The Lightweight Communication Calculus (LCC) used in OpenKnowledge has an analogous aim but in our case the essential information is that of choreography between peers. The specification of this interaction is the interaction model, depicted as "IM" in Figure 1.

Figure 2 shows a basic interaction model in LCC. It defines an interaction between peers in two roles, `r1` and `r2`. In the interaction some peer, `A`, in role `r1`, would send the message, `m1`, to peer `B` in role `r2`; then it would wait for a reply from peer `B` with message `m2`. Conversely, peer, `B`, in role `r2`, would wait for the message, `m1`, from peer `A` in role `r1`; then it would send a reply to peer `A` with message `m2` if it can satisfy the constraint `c`.

Although it is not shown in Figure 2, role definitions in LCC can be recursive and the language supports structured terms in addition to variables and constants so that, although its syntax is simple, it can represent sophisticated interactions.

2 Peer Ranking via Interactions

Large scale networks of peers coordinated through interaction models do not yet exist but if they were to exist then we would expect very large numbers of interactions to be available on the network. This is likely to require the discovery service to operate in a manner similar to a traditional Web page search engine, offering rankings as well as keyword searches. We describe below the simplest such algorithm, based on the well known Page Rank algorithm for Web pages.

Like Page Rank, our ranking algorithm works by assigning a ranking to a peer at any given time as a function of its previous ranking modified by the rankings of each peer with which it has interacted (we refer below to such peers as supporting peers). The modification made by each of these peers is spread equally across the set of peers which it supports. In the algorithm definition below we assume that we have available minimal information about interactions - only knowing for each interaction which peers were involved in it and whether or not it completed successfully. This is less information than we actually can obtain routinely (since the coordinating peer has access to all the structure of a completed interaction) but we make this simplification so as to explore how far we can get with the bare minimum of data. Data storage and lookup requirements are similar to that of established Page Rank based services on the Web (such as Google).

Given:

- A data set of initial rankings for each peer, each of the form $cr(P, T) = R$
- A data set of records of successful or failed interactions, each of the form $im(T, I, P_I, CP)$

where:

- P is the identifier of a peer

- T is either $\oplus$ (denoting the positive ranking) or $\ominus$ (denoting the negative ranking)
- R is the numerical magnitude of the (positive or negative) ranking.
- I is the identifier of the interaction model
- P_I is the initiating peer for an interaction
- CP is the set of peers that subscribe to the interaction initiated by P_I

The algorithm for calculating the current rank of a peer is then as follows (where i is the empirically chosen number of iterations used to obtain stable ranking values, often less than 100):

- For i iterations:
  - For each peer, P:
    - Calculate $rank(P) = \langle R_p, R_n \rangle$
    - assign $cr(P, \oplus) = R_p$
    - assign $cr(P, \ominus) = R_n$

$rank(P)$ calculates the current rank for peer P, where d is an empirically chosen damping value used to tune the ranking system (a frequently used value in page ranking is 0.85).

$$rank(P) = \langle\ (1-d) + d \times r(s(\oplus, P), \oplus),\ (1-d) + d \times r(s(\ominus, P), \ominus)\ \rangle \tag{1}$$

$s(T, P)$ gives the list of peers supporting peer P. If T is $\oplus$ then this is the set of positive support or if T is $\ominus$ this is the set of negative support. Note that this is a list (which may contain duplicates) rather than a set because we are interested in the number of times each peer is supported.

$$s(T, P) = [Ps | a(P, T, Ps)] \tag{2}$$

$a(P, T, Ps)$ is true when peer P is supported by peer Ps either positively (if T is $\oplus$) or negatively (if T is $\ominus$). Note that this may (intentionally) generate the same instance of Ps more than once. This allows us to count the number of times the same peer is supported.

$$a(P, T, Ps) \ \leftarrow\ im(T, I, P_i, CP)\ \wedge \begin{pmatrix} (P = P_i\ \wedge\ Ps \in CP) \\ \vee \\ (P \in CP)\ \wedge\ Ps = P_i) \end{pmatrix} \tag{3}$$

$r(S, T)$ is the sum of the current ranks for all the peers in peer set, S, each peer's rank being divided by the number of peers it supports (so as to apportion the influence of the rank evenly across those supported peers). If T is $\oplus$ then this is the sum of ranks from positive associations or if T is $\ominus$ this is the sum of ranks from negative associations.

$$r(S, T) = \sum_{P \in S}^{P} \frac{cr(P, T)}{|(u(T, P)|} \tag{4}$$

$u(T, P)$ gives the list of peers supported by peer P. If T is $\oplus$ then this is the set of positive support or if T is $\ominus$ this is the set of negative support. Note the symmetry between this and $s(T, P)$.

$$u(T, P) = [Ps | a(Ps, T, P)] \tag{5}$$

We want peer ranking to provide a similar behaviour to that experienced on the traditional Web, where a power law effect distinguishes a few dominant pages that rank very highly on a topic from the majority that retain lower levels of popularity. This encourages strong competition for popularity, which we would like to engourage in large scale coordinations. To demonstrate this effect in operation we take the interaction model of Figure 2 and run it with populations of peers of increasing size (4, 8, 16, then 32) in which each peer has identical behaviour and each peer is compliant with the protocol described in the interaction model, so all interactions run perfectly to completion. The results are depicted in the four graphs of Figure 3. Each line on the graphs is the positive rating of a peer (measured on the y-axis) as it changes while the number of interactions increases (measured on the x-axis). Each graph shows a single peer achieving dominance with a rapid falloff to others with lower rank and a "tail" of low ranking peers. As the number of peers increases this effect becomes more pronounced and, although separation of dominant peers to the fullest extent takes more interactions with greater peer numbers, the dominant peers begin to separate early in all cases.

We also want our ranking system to be sensitive to changes in peer behaviour, for example if a peer that initially was compliant becomes less compliant in its behaviour then we want its rating (eventually) to change. Figure 4 demonstrates this behaviour. The graph shows the change in positive and negative rating for each of four peers. Instead of being always compliant, we make peers

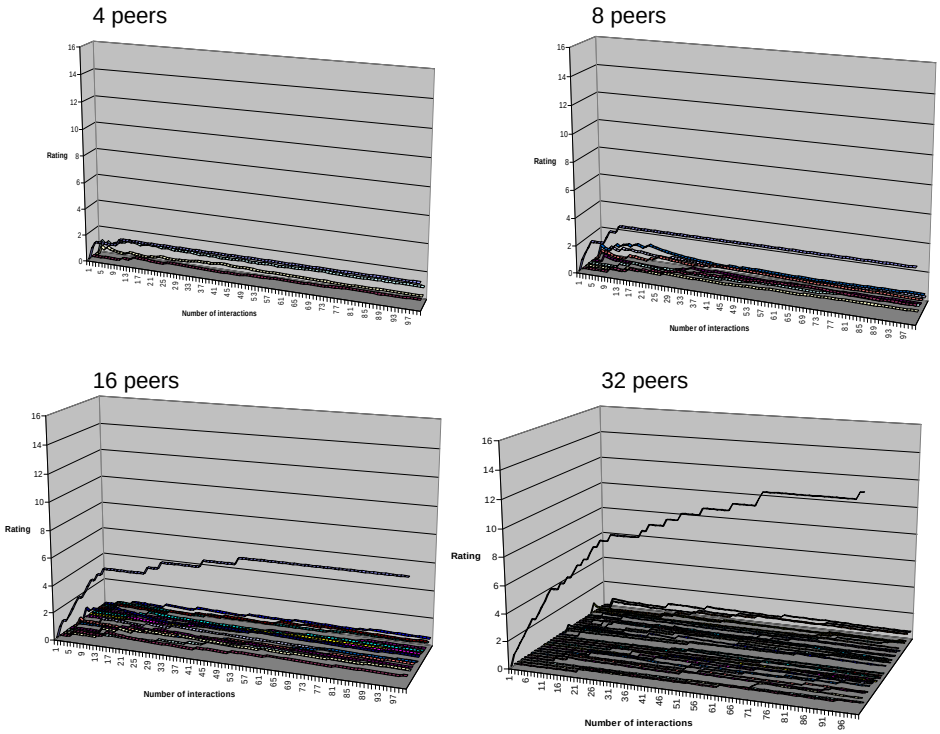

Figure 3: Rating changes with increasing peer group size in a perfectly compliant peer population.

sometimes non-compliant by making them fail to satisfy the constraint, `c` in the interaction model of Figure 2, which breaks the protocol. We make the likelyhood of failure of a peer increase as the peer has more successful interactions, and decrease as the peer has more failed interactions, so that the likelyhood of compliance of each peer alternates over time. We see the ratings in Figure 4 change in response to this, as the positive and negative rating lines for each peer tend to twine across each other. As a peer is successful, its positive rating exceeds its negative rating but this success makes it more likely to fail to be compliant (because we made each peer decide to behave less compliantly when it has been successful) so its negative rating then climbs and can exceed its positive rating, making it less likely to be sleected by other peers. Overlaid on this twining effect is the basic power law effect that we demonstrated in Figure 3, so peer 4 has a dominant rating (both positive and negative, since its popularity attracts many interactions including more that fail as well as more that succeed), followed by peer 3 and then (some way behind) by peers 1 and 2.

We have focused thus far on ranking peers but, since interaction models anchor coordination we must also be concerned about the quality of those models. To see the difference on peer ratings depending on the quality of the interaction models they share, we introduce the more complex interaction given in Figure 5. In this interaction a meeting organiser chooses at random a subset of three peers and a date from a set of five possible dates; then it checks with each selected peer whether the chosen date is acceptable. If any of the peers is not able to confirm the date is acceptable then the interaction fails.

Let's assume that each peer has only three dates (out of the five possible) that are aceptable to it, and that these dates differ between peers. Figure 6 shows the ratings for 20 peers running this interaction 200 times. Dark coloured lines represent positive ratings while light coloured lines represent negative ratings. The pattern overall is confused, with no dominant peer and negative ratings tending to exceed positive ones. The reason we do not see the ordered emergence of dominant peers, as in earlier examples, is that the `organiser` peer's early choices of the set of peers to coordinate (`S` in Figure 5) and of date value (`V` in Figure 5) are made in ignorance of the constraints on the peers it is trying to coordinate, so it is highly likely that `participant` peers will fail to satisfy the acceptability constraint and the interaction will then fail.

We can address the problems of the interaction in Figure 5 by extending it as shown in Figure 7. This interaction operates in exactly the same as for Figure 5 when fixing a date with the set of three selected peers but, before that, it employs a more sophisticated algorithm for selecting the peers set and date by polling peers to find out the dates at which they believe they are available and choosing a date that every peers in the selected subset claimed is available for it. This increases the likelyhood that peers will be able to cooperate when asked, although it does not guarantee success since a peer is not obliged to agree to a date it previously said was free.

Figure 8 shows how the chaotic behavior we saw in Figure 6 is corrected with the new interaction model of Figure 7. Three of the peers now dominate with positive ratings consistently exceeding their negative ratings, and the other peers form an orderly "tail" of much lower ranked but stable peer group members. As our example shows, a stable peer rank is dependent on choosing high quality interactions appropriate to the task. This is a beneficial feature because it encourages peers in pursuit of high rankings to be selective in their choice of interaction models, which we would expect to force out low-grade models. Poorly performing interaction models

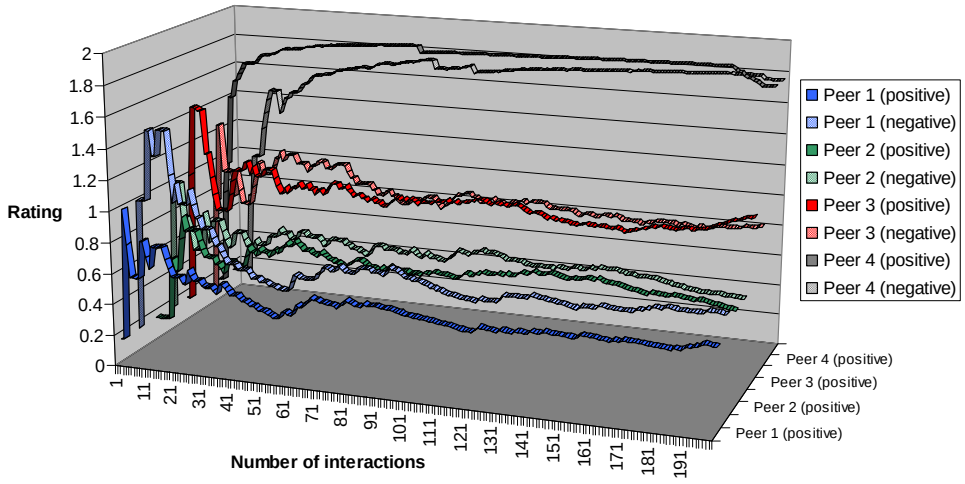

Figure 4: Rating differences with time-variant compliance.

```
a(organiser(P), X) ::
    a(organiser(S,V), X) <--
            random_subset(3, P, S) and
            choose(V, [1,2,3,4,5]).

a(organiser(S,V), X) ::
    null <-- S = []
    or
    (  check(V) => a(participant, Y) <-- S = [Y|Sr] then
       confirm(V) <= a(participant, Y) then
       a(organiser(Sr,V), X) ).

a(participant, Y) ::
    check(V) <= a(organiser(_,_), X) then
    confirm(V) => a(organiser(_,_), X) <-- acceptable(V).
```

In the interaction model above, `P` is a set of identifiers for peers; `random_subset(N, P, S)` selects a random subset, `S` of cardinality, `N` from set `P`; `choose(V, S)` selects an element, `V`, from set `S`; the expression `null <-- S = []` means "do nothing if set `S` is empty"; the expression `S = [Y|Sr]` selects the first element, `Y`, from set `S` leaving the remaining elements in set `Sr`; and `acceptable(V)` succeeds if the date, `V`, is aceptable to the `participant` peer as a meeting time.

Figure 5: A poorly crafted interaction model for arranging a meeting time.

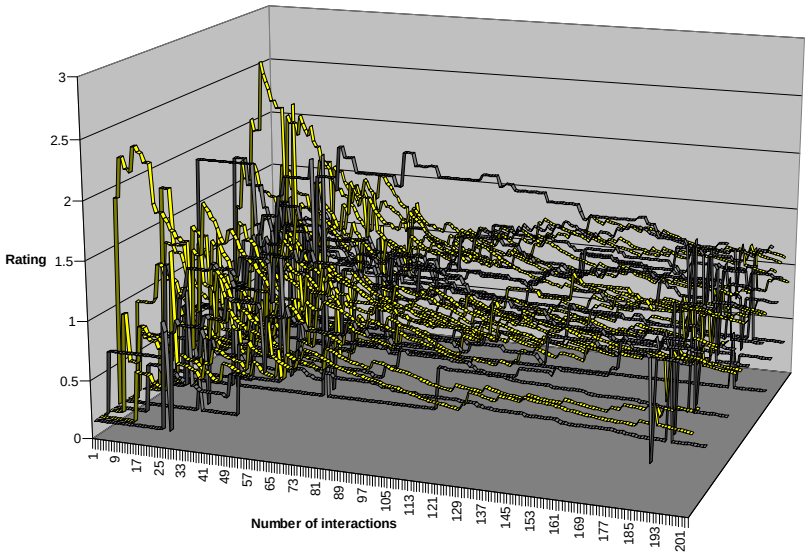

Figure 6: Ratings for a poorly crafted interaction model.

```
a(organiser(P), X) ::
    a(organiser(P,[],S,[],Vs), X) then
    a(organiser(S,V), X) <-- member(V, Vs).

a(organiser(P,Ss,Sf,Vs,Vf), X) ::
    ( query => a(candidate, Y) <-- select_candidate(P, Ss, Y, Pr) then
      response(R) <= a(candidate, Y) then
      ( a(organiser(Pr,[Y|Ss],Sf,Vn,Vf), X) <-- merge(R, Vs, Vn)
        or
        a(organiser(Pr,Ss,Sf,Vs,Vf), X) <-- not(merge(R, Vs, _)) ) )
    or
    ( null <-- length(Ss, N) and N = 3 and Sf = Ss and Vf = Vs ).

a(candidate, Y) ::
    query <= a(organiser(_,_,_,_,_), X) then
    response(R) => a(organiser(_,_,_,_,_), X) <-- response_data(R).
```

In the interaction model above, `select_candidate(P, Ss, Y, Pr)` selects an element, `Y`, from the set of peer identifiers, `P`, that is not already in the set of selected candidates, `Ss`, leaving the remaining peers, `Pr`, and ensuring that `Ss` is no larger then three peers. `merge(R, Vs, Vn)` makes `Vn` the intersection of `R` (the set of dates said by a peer to be acceptable) and `Vs` (the set of dates believed to be acceptable to all peers contacted so far). `length(Ss, N)` is true when the cardinality of set `Ss` is `N`. The roles of `organiser(S,V)` and `participant` are the same as in Figure 5.

Figure 7: Extending the interaction model of Figure 5.

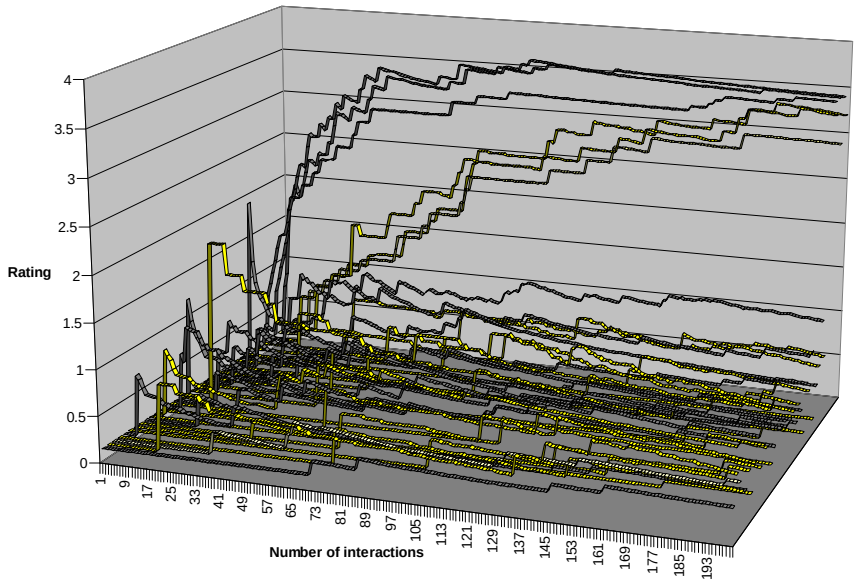

Figure 8: Ratings for the extended interaction model of Figure 7.

are easy to detect (they fail frequently) so a discovery service can easily supply this information to peers searching for appropriate interactions.

3 Ontology Matching to Stabilise Peer Rankings

We have seen how careful crafting of interactions influences peer rank but even well crafted interactions can fail. One cause of this is ontology mismatch: a peer may have a constraint to satisfy in an interaction that uses terminology with which it is unfamiliar, preventing it from complying when otherwise it might. A traditional response to this problem is to write ontology mappings between the terminologies used by peers, but this is a limited solution because it is difficult to predict in advance of interaction where mappings will be needed and the alternative (to write mappings for all the terms every combination of peers coudl possibly share) is impractical. Sharing interactions can help here because, if it is possible to share information about the ontology mappings used by other peers, a peer that does not recognise a constraint may be able to obtain mapping possibilities from other peers. To demonstrate this we use the interaction model given in Figure 9. This interaction involves messages sent between three peers, with the content of the messages being determined by four constraints (two on the initiating peer and one on each of the other peers).

Suppose that we have four peers: two of them are capable of satisfying only the constraints `a(1)`, `a(1,2)`, `a(2,3)` and `a(3,4)`, while the other two are capable of satisfying only the constraints `b(1)`, `b(1,2)`, `b(2,3)` and `b(3,4)`. If we try to coordinate these peers with the interaction model of Figure 9 it will always fail because none of the peers can match the constraints it can satisfy to those of the interaction model. If, however, we give the peers the capacity to guess that it might map a a constraint to another with the same arity (*e.g.* `c1(X)` to `a(X)`) then each peer can attempt a simple ontology mapping. These mappings could lead to failed interactions, since they are guesses, but if some are successful and if it is possible to convey this mapping information to other peers then we make it more likely that each peer will discover successful mappings, leading to successful interactions.

Figure 10 shows this effect. Initially, the interactions fail, giving high negative ratings to all peers but after 85 interactions a peer discovers a mapping that leads to a successful interaction and this information is propagated to the other peers which then also make successful mappings. All peers then raise their positive rating, with the usual dominance effect from one of them.

The example of Figure 10 is based on a primitive form of mapping prediction but more sophisticated forms of matching are possible in which the structure of interaction models are used to estimate the conditional probability of word usage based on the current interaction state. The most important feature of these sorts of algorithms is that they require no pre-defined ontologies to have been engineered (since guesses about mappings are based only on peer behaviour during interaction). If ontologies are available, however, the structure of interaction models can also be used to narrow the focus of conventional ontology mappers, making them more efficient.

4 Peer Ranking and Provenance

In many data sharing applications we are concerned to use information about the source of the data (its provenance) in order to influence which data we use. Traditionally, the way in which provenance is incorporated into data-sharing interactions is by devising

```
a(r1, A) ::
    M1 => a(r2, B) <-- c1(M1) then
    M2 <= a(r2, B) then
    M3 => a(r3, C) <-- c2(M2, M3) then
    M4 <= a(r3, C).

a(r2, B) ::
    M1 <= a(R, A) then
    M2 => a(R, A) <-- c3(M1, M2).

a(r3, B) ::
    M3 <= a(R, A) then
    M4 => a(R, A) <-- c4(M3, M4).
```

In the interaction model above, `c1(M1)` generates message `M1`; `c3(M1,M2)` generates message `M2` dependent on `M1`; `c2(M2,M3)` generates message `M3` dependent on `M2`; and `c4(M3,M4)` generates message `M4` dependent on `M3`.

Figure 9: An interaction model requiring four constraints to be satisfied.

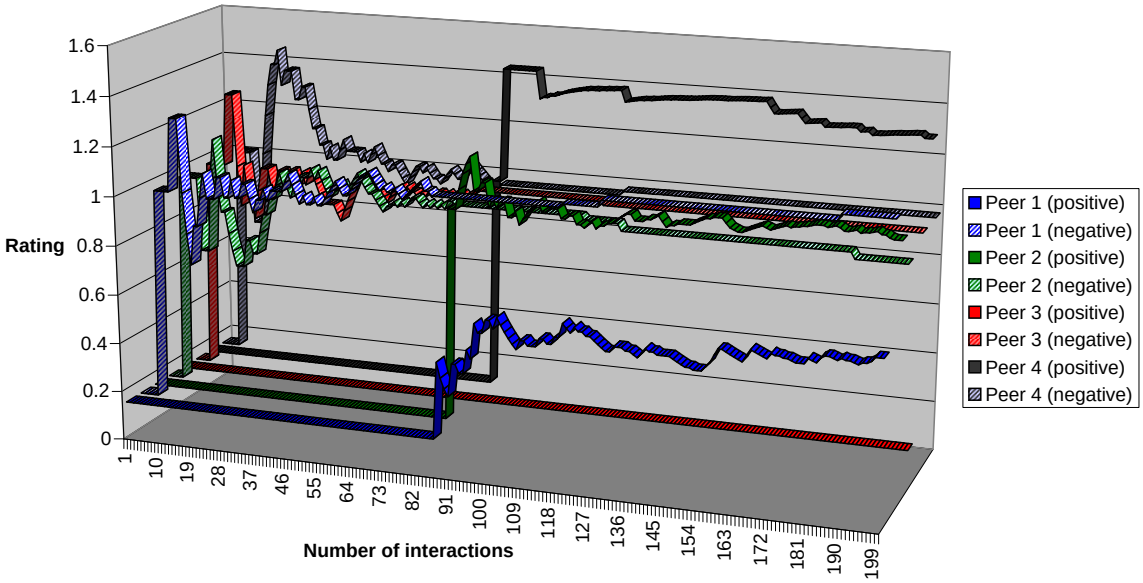

Figure 10: Ratings incorporating simple ontology matching.

```
a(data_source, X) ::
    request(Q) <= a(data_seeker, Y) then
    offer(D) => a(data_seeker, Y) <-- have_data(Q, D).

a(data_seeker, Y) ::
    request(Q) => a(data_source, X) <-- need_data(Q) then
    cache_data(Q,D) and acceptable(T, D) <-- offer(D) <= a(data_source, X).
```

In the interaction model above, `X` is the identifier for the data source; `Y` is the identifier for the data seeker; `have_data(Q, D)` generates the best available set of data, `D`, known to `X` for query, `Q`; `need_data(Q)` generates a data query, `Q` for `Y`; `cache_data(Q,D)` merges the data set, `D` with the data cached by `Y` for query, `Q`; and `acceptable(T, D)` succeeds when the data set, `D`, has a mean quality level that exceeds the threshold `T` (note that `T` is set to a specific value).

Figure 11: An interaction model for competitive data sharing.

ontologies for provenance and supplying task/domain-specific methods for propagating provenance information between peers. We could do this in the OpenKnowledge system by writing interaction models in LCC that specify provenance propagation along with the message passing needed for "raw" data transmission. This takes enginering effort so it is useful if peer ranking could, alone, take some of the strain of assigning reputation to peers in a data sharing context.

Figure 11 gives a basic interaction model for data sharing. It describes two roles: a data source which offers a data set to a data seeker if it receives a request for data from the seeker, and a data seeker which requests data for a query and caches the data received (testing also that the data set was of acceptable overall quality). This interaction model has no representation of the provenance of the data but there is a point in the interaction, when the data seeker sends its request to a data source, when an appropriate data source needs to be chosen. At this point the data seeker can, if it so chooses, use peer rank information to select the highest ranking peer. The rank of a data source in this interaction will, in turn, depend on how frequently the data seekers with which it interacts find the quality of the data it supplies to be acceptable (otherwise the `acceptable` constraint in the data seeker role will fail and consequently the interaction overall will fail). This gives a feedback loop from supply to ranking via the peers that request data.

We can use the interaction model of Figure 11 to simulate the flow of data from a central database through a peer group. To make our simulation as simple as possible we represent data quality as a number ranging from 0 (lowest quality) to 1 (top quality). Instead of storing actual data elements, the peers in our simulation store these quality values (representing the quality of a data item). A single peer in the simulation represents the central database and (to keep it simple) it generates a set of 10 data elements with quality chosen randomly to be between 0 and 1, so the data elements that are obtained from it by peers will tend to have a normal distribution with mean of 0.5. Let us assume that the quality that is acceptable to peers is higher than this mean, setting it at 0.8 for all peers. This means that peers seeking data from the database will obtain a mixture of acceptable (mean quality greater than 0.8) and unacceptable results, so the database will build up a negative ranking as well as a positive one. Initially, however, the other peers will have no data so if any of them are asked for data the interaction definitely will fail, increasing their negative ranking while their positive ranking remains unchanged. At first, this makes the central database the most popular peer but as peers gain data from the database through interaction then those peers lucky enough to get high quality data can then offer it to other peers, raising the positive ranking of the supplier and giving the recipient high quality data with which it can increase its rating.

Figure 12 shows the behaviour of this system for nine peers interacting with a database. The positive and negative ratings for the top rated peer (indicated on the diagram) cross over after about 160 interactions (note that the units on the X axis are in tens of interactions) because the peer has acquired a substantial amount of high quality data from the database and is using that to dramatically increase its positive rating. Its high rating makes it attractive to other peers which place demands on it for more data than it possesses so its negative rating climbs too, but not enough to overtake its positive rating.

We recorded the change in data quality obtained at each peer by taking the sum (for the 10 different queries) of the mean quality of data obtained for that query. This gives a possible minimum from 0 (no data obtained) to 10 (all data for all queries at a quality level of 1). The results are shown on the right-hand graph of Figure 13. After 1000 interactions the sum of mean data quality at each peer rises to approximately 6, which is consistent with the drop in popularity of the central database that we saw in Figure 12. Because peers retain high quality data they are able, once they eventually have obtained high quality data from the central database or from another peer, to boost their ratings by contributing to more successful interactions than the central database (which delivers data a more variable quality). The overall effect is to "lift" the data provision from the central database into the peer group.

To show how close our peer to peer method of data sharing comes to the ideal situation, where all the peers can directly poll the database as frequently as they wish, we set up a simulation in which this takes place. In this simulation there is no need for ranking of peers because every peer interacts only with the database and the database is assumed to have infinite resources so that it can answer every query. The graph on the left in Figure 13 shows the change in mean quality of data at each peer under these ideal conditions. As expected, the quality rises quickly to nearly the maximum possible because each peer in this simulation simply polls the database as many times as it takes to get the highest quality data. This is ideal but it is impractical when the peer group gets large because the database has to deal with all of the interactions and would quickly become overloaded with queries. By contrast,

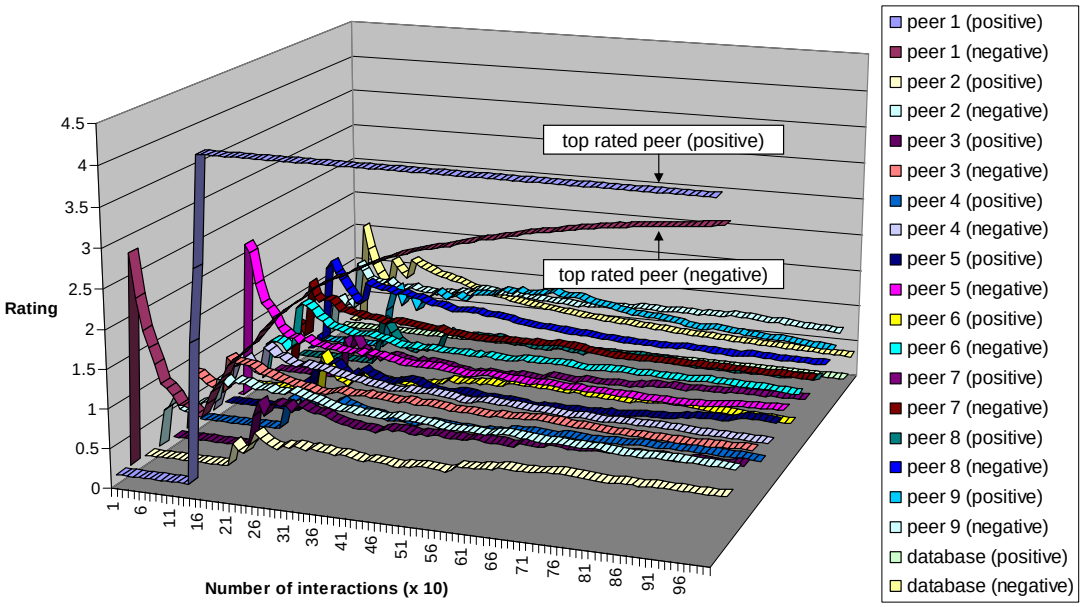

Figure 12: Ratings for data sharing example.

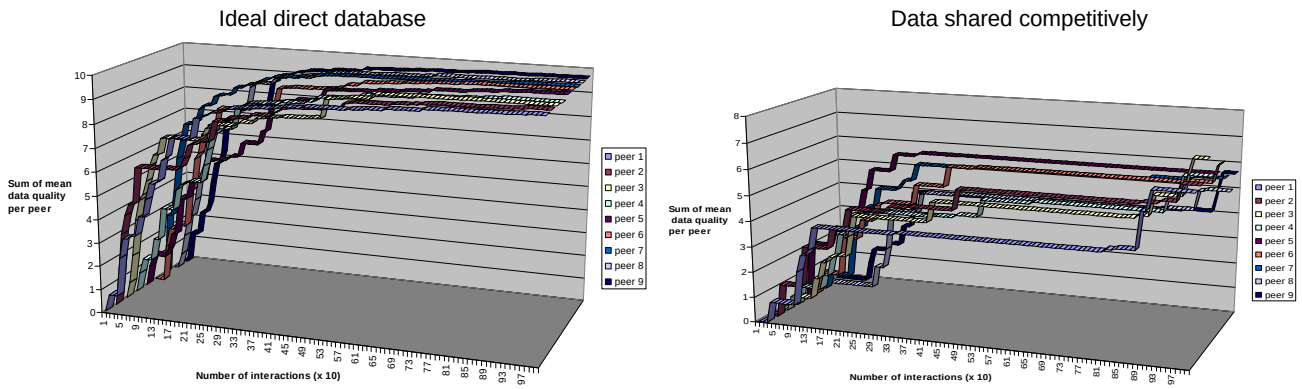

Figure 13: Comparison of ideal versus competitive data sharing.

the results of peer to peer sharing on the right of Figure 13) show slower improvement in quality but (from Figure 12) we know that this improvement was done by querying across the peer group rather than risking overload on the central database.