Open Knowledge
Coordinating Knowledge Sharing through Peer–to–Peer Interaction

Dave Robertson[1], Fausto Giunchiglia[2], Frank van Harmelen[3], Maurizio Marchese[2], Marta Sabou[4], Marco Schorlemmer[5], Nigel Shadbolt[6], Ronnie Siebes[3], Carles Sierra[5], Chris Walton[1], Srinandan Dasmahapatra[6], Dave Dupplaw[6], Paul Lewis[6], Mikalai Yatskevich[2], Spyros Kotoulas[3], Adrian Perreau de Pinninck[5], and Antonis Loizou[6]

[1] Informatics, University of Edinburgh, UK
[2] Information and Communication Technology, University of Trento, Italy
[3] AI Department, Vrije Universiteit Amsterdam, Netherlands
[4] Knowledge Media Institute, Open University, UK
[5] Institut d'Investigació en Intel·ligència Artificial, Barcelona, Spain
[6] Electronics and Computer Science, University of Southampton, UK

Abstract. The drive to extend the Web by taking advantage of automated symbolic reasoning (the so-called Semantic Web) has been dominated by a traditional model of knowledge sharing, in which the focus is on task-independent standardisation of knowledge. It appears to be difficult, in practice, to standardise in this way because the way in which we represent knowledge is strongly influenced by the ways in which we expect to use it. We present a form of knowledge sharing that is based not on direct sharing of "true" statements about the world but, instead, is based on sharing descriptions of interactions. By making interaction specifications the currency of knowledge sharing we gain a context to interpreting knowledge that can be transmitted between peers, in a manner analogous to the use of electronic institutions in multi-agent systems. The narrower notion of semantic commitment we thus obtain requires peers only to commit to meanings of terms for the purposes and duration of the interactions in which they appear. This lightweight semantics allows networks of interaction to be formed between peers using comparatively simple means of tackling the perennial issues of query routing, service composition and ontology matching. A basic version of the system described in this paper has been built (via the OpenKnowledge project); all its components use established methods; many of these have been deployed in substantial applications; and we summarise a simple means of integration using the interaction specification language itself.

1 Introduction

To coordinate the sharing of knowledge in an automated way on a large scale is an aim shared by many areas of computing. In some areas the challenge of scale is especially difficult because the environment for knowledge sharing is open, in the sense that we cannot prescribe which programs will choose to interact in sharing knowledge. One solution to this problem would be to develop sophisticated agent architectures to make individual programs more adaptive and resilient. In practice, however, the architectures of most internet-based systems are not specifically agent-oriented. A second approach, and the one taken in this paper, is to take the approach adopted in electronic institutions and use a formal model of the intended interaction in different social contexts in order to provide the necessary constraints on interaction and synchronisation of message passing. We motivate our approach by comparison to the aspirations of semantic web systems but, in Section 6, we discuss the relevance of our approach across a broader range of large scale coordination systems.

Semantic Web efforts encourage designers to specify pages and programs in a knowledge representation language. What fundamentally distinguishes the Semantic Web from traditional knowledge representation is not the formal representations of pages and programs but the way these representations are used in automated processing. The use we have in mind is to improve the accuracy of Web searches and to allow components (especially those that are programs operating as Web services) to be connected automatically rather than simply be discovered individually. It simplifies our explanation, without loss of generality, if we think of all components (pages and programs) as peers[1] capable of supplying information in some formal language: a page supplies information about that page; a program generates information, perhaps requiring input information in order to do so (we return to the data-centric versus process-centric view in Section 6.1). This way of attempting to share knowledge between peers encounters elementary problems, familiar from traditional software and knowledge engineering:

– When two different components supply exactly the same formal expression this does not imply that they mean the same thing. Conversely, when they supply different formal expressions they may mean the same thing. In traditional knowledge (and software) engineering we avoid this problem by reaching a consensus amongst component designers on an ontology. To obtain an ontology one needs an oracle to decide which formal terms to use. There are only two sources of oracle: human or mechanical. Neither source scales well. Human oracles can only agree in small groups for narrow tasks or domains. Mechanical oracles need to acquire domain knowledge via automated learning methods, none of which has scaled beyond narrow tasks or domains.
– Only some combinations of components work well together, even when the meanings of their appropriate inputs and outputs correspond, because of assumptions made deep within the design of components. In traditional knowledge (and software) engineering this problem is detected through integration

[1] We use the word "peer" in this paper simply to underline that no component is given special authority over others, other than via the manner in which components interact with humans and with each other. We use "peer" rather than "agent" in order to emphasise that the programs we coordinate need not be built to an agent architecture.

testing: by running assemblies of components comprising the system and checking for defects such as range errors (where the type of an input is correct but its value is outside the range anticipated by the component's designers). It is infeasible to test all ranges for all variables for all components so integration testing only covers some small subset of the possible interactions.

– Components may fail without warning, either because they are removed or as a consequence of their environment. Traditionally this problem is solved by ensuring that the components are in a shared, controlled environment. No such controls are available in an open environment, in which we may not always know who, what or where are our peers.
– Even without the problems above, theoretical results (*e.g.* [8]) show that in general it is impossible to guarantee consistently shared common knowledge in asynchronous distributed systems. The engineering solution to this problem is to provide mechanisms for establishing synchronisation between appropriate components, but for this we must have a frame of reference to bound the synchronisation - otherwise synchronisation itself falls foul of the general consistency problem.

As an example, suppose we want to find the addresses of expert researchers on wave energy in Scotland. A conventional Web search using Google doesn't find this information easily because the information about who is an expert is hard to infer from conventional Web pages and the large number of pages discussing wave energy tend to swamp out the few home pages of expert researchers and their groups. A semantic web approach might do better if appropriate information services could be combined. Suppose that an expert finding service does exist and it offers to supply as output a set, S, of names of specialists if given as input a request for experts in some country, X, and discipline, D. A simple specification of this service (made much simpler then, say, an OWL-S, specification since we need only a simple example here) might be this:

$$\boxed{\begin{array}{rl} \text{service}: & expert\_finder \\ \text{input}: & request\_experts(country(X), discipline(D)) \\ \text{output}: & specialists(X, D, S) \\ \text{operation}: & get\_experts(X, D, S) \end{array}}$$

Suppose also that two address finding services exist in Scotland, allowing one to send a request for the address of a person with a given name, P, and receive an address, A, for him or her. One of these services is run by a UK address company (*uk_address*); the other by the University of Edinburgh (*univ_ed*). Both, by coincidence, have identical specifications:

$$\boxed{\begin{array}{rl} \text{service}: & address\_finder \\ \text{input}: & request\_address(person(P)) \\ \text{output}: & address(P, A) \\ \text{operation}: & get\_address(P, A) \end{array}}$$

A semantic web search engine could not obtain the set of addresses we require using only these service specifications (assuming that the specifications connect to actual services via some grounding mechanism not discussed here) because it is missing some vital functionality:

Functionality 1. Interaction specification: *Wherever there is distributed computation there is an issue of control over message passing between services. The $expert\_finder$ service supplies a set, S, of names of experts but the $address\_finder$ service deals with individual names. Something must infer that we should take each element of S and pass it to the $address\_finder$, collecting the results. We cannot always infer this by examining only the types and values of variables - just like normal programming, someone or something must supply the necessary control structure.*

Functionality 2. Interaction coordination: *We need to know that some form of interaction involving $expert\_finder$ and $address\_finder$ would be useful for the task. Notice that our task is not, and could not be, specified with the services because the engineers of those services could not be assumed to predict all the possible tasks of which they might be a part. To provide any automation we must separately obtain a formal description of the interaction we might need. And this description will most likely be provided using language terms that are different from those used by the services which want to interact with a given service. Some mechanism must make sure that the services are fed appropriate information. The variable, D, in the $expert\_finder$ service will need to be bound to the name of some scientific discipline that the service recognises and it is unlikely to recognise all conceivable discipline descriptions. Will it accept "wave energy"? We don't know simply by looking at the type of the variable.*

We have described major problems associated with the aspiration of semantic web efforts. These are traditional engineering problems but their traditional solutions do not apply in open environments. The key to solving these, we shall argue, is in making specifications of interactions an integral part of knowledge exchange between peers. In Section 2 we summarise a compact but expressive formal language for specifying and computing interactions. We then describe the basic mechanisms needed to support this computation in an open environment: ontology alignment (Section 3.1), discovery (Section 3.2) and visualisation (Section 3.3). Section 4 then combines these elements within a minimal model of interaction, shareable between peers. Finally, since we view this as an evolutionary manifesto, in Section 6 we compare our approach to the major existing paradigms.

2 Interaction Specification

The functional requirement we must satisfy here is: given that a peer has been given a model of interaction, use this computationally to control its own behaviour and communicate to other relevant peers the behaviours expected of them for this interaction.

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

Fig. 1. LCC syntax

From previous implementations by the authors using the system described below, there are several different ways in which a peer might exploit an interaction model with a precise computational behaviour:

– It might simply run this model locally, sending messages to other peers but not allowing them to be privy to the broader picture of interaction contained in the model. This is a traditional server-based style of coordination.
– It might distribute appropriate components of the model to those peers with which it interacts, on the assumption that each of them will separately run that component - producing a distributed computation.
– It might use the model as a form of script which it passes entire to the next peer with which it needs to interact - producing complex interactions amongst many peers via pairwise interactions between peers.

None of these computational models is optimal. Server or script based computation keeps the interaction model in one piece, giving advantages in imposing constraints on interaction that apply across groups of peers. On the other hand, a server based computation means that peers other than the one acting as server have little control over the course of the interaction; while a script based computation only works for interactions that can be deconstructed into a series of pairwise interchanges. Since no winning execution strategy is known, it makes sense to use an interaction modelling language that has a semantics independent of any one of these strategies but that is capable of being executed by any of them.

The need to supply this sort of information has been recognised by many in the semantic web community. The most direct solution is to specify the process of service combination, and the roles undertaken by the services in that process, in an executable language. We give in Figure 2 a specification for our running example in one such language: the Lightweight Coordination Calculus (LCC) which is the core language used in the OpenKnowledge project. Figure 1 defines the syntax of LCC. An interaction model in LCC is a set of clauses, each of which

$$
\begin{array}{l}
a(\mathit{expert\_locator}(X,D,L),C) :: \\
\quad \mathit{request\_experts}(X,D) \ \Rightarrow \ a(\mathit{expert\_finder},E) \ \mathit{then} \\
\quad \mathit{specialists}(X,D,S) \ \Leftarrow \ a(\mathit{expert\_finder},E) \ \mathit{then} \\
\quad a(\mathit{address\_collector}(S,L),C) \\
\\
a(\mathit{address\_collector}(S,L),C) :: \\
\quad \left(\begin{array}{l}
\mathit{request\_address}(P) \ \Rightarrow \ a(\mathit{address\_finder},F) \leftarrow S = [P|Sr] \wedge L = [A|Lr] \ \mathit{then} \\
\mathit{address}(P,A) \ \Leftarrow \ a(\mathit{address\_finder},F) \ \mathit{then} \\
a(\mathit{address\_collector}(Sr,Lr),C)
\end{array}\right) \ \mathit{or} \\
\quad \mathit{null} \leftarrow S = [] \wedge L = [] \\
\\
a(\mathit{expert\_finder},E) :: \\
\quad \mathit{request\_experts}(X,D) \ \Leftarrow \ a(\mathit{expert\_locator}(X,D,L),C) \ \mathit{then} \\
\quad \mathit{specialists}(X,D,S) \ \Rightarrow \ a(\mathit{expert\_locator}(X,D,L),C) \leftarrow \mathit{get\_experts}(X,D,S) \\
\\
a(\mathit{address\_finder},F) :: \\
\quad \mathit{request\_address}(P) \ \Leftarrow \ a(\mathit{address\_collector}(S),C) \ \mathit{then} \\
\quad \mathit{address}(P,A) \ \Rightarrow \ a(\mathit{address\_collector}(S),C) \leftarrow \mathit{get\_address}(P,A) \ \mathit{then} \\
\quad a(\mathit{address\_finder},F)
\end{array}
$$

Fig. 2. Example interaction model

defines how a role in the interaction must be performed. Roles are described by the type of role and 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 (‘$\Rightarrow$’) or incoming from another peer in a given role (‘$\Leftarrow$’). Message input/output or change of role can be governed by a constraint defined using the normal logical operators for conjunction, disjunction and negation. Notice that there is no commitment to the system of logic through which constraints are solved - so different peers might operate different constraint solvers (including human intervention).

Returning to the example of Figure 2, each of the four clauses defines the message passing behaviour of a role in the interaction. The first clause defines the role we wish some peer (identified by name C) to perform: that of a locator for experts which, given a country, X and a discipline, D, identifies a list of addresses, L. To undertake this role, C must send a message to an expert finder, E, requesting names of experts and then receive a set of names, S from E before taking the role of an address collector. The second clause defines the address collector role which involves recursing through the set of names, requesting, and then receiving, an address for each from an address finder, F. The third and fourth clauses define our earlier $\mathit{expert\_finder}$ and $\mathit{address\_finder}$ services in terms of the message passing required by each of them. Note that these make specific commitments to the temporal behaviours of the services in our interaction (for instance we prescribe that an $\mathit{expert\_finder}$ is contacted only once in this interaction while an $\mathit{address\_finder}$ may be contacted many times).

It is not the task of this paper to explain LCC in detail or to justify its use rather than some other process specification language (for that argument see [16]). For our current purposes, the salient features of LCC are:

- It is an executable specification language, providing the features common to such languages (such as variables, data structures and recursion). All of these were needed to deal with even our simple running example.
- Despite its specificity in terms of data and control structure, so we know precisely how we want our services to interact, there remain obstacles to achieving that interaction:
  - Which specific terms will work when communicating between services? This is discussed in Section 3.1.
  - How do we know which actual services to use? In our LCC specification the variables E and F are unbound but messages must be sent to specific services. This is discussed in Section 3.2.
  - What happens if our peers are human operated? For example, the operation $get\_experts(X, D, S)$ in the third clause of Figure 2 might involve asking a human who the experts are. At that point the human needs to know enough of the interaction's context to give an informative reply (see Section 3.3).

Principle 1. *Interaction models are declarative, executable specifications that may be understood independently of any particular peer or execution model. The conditions under which they are run, however, requires more run-time support than a traditional executable specification language.*

3 Interaction Coordination

The functional requirement we must satisfy here is: given a peer with no knowledge of how to interact with others to perform some task, obtain for it a description of an appropriate interaction.

One way to do this might be through synthesis, either fully automated (*e.g.* [4]) or interactive (*e.g.* [12]), so that peers could compose appropriate service clusters for whatever task arises. Although an important means of initiating some kinds of interaction specification, synthesis is unlikely to be the main source of this functionality for three reasons. First, as in Section 1, the specifications we need are non-trivial to synthesise automatically and would require specialist expertise to synthesise interactively. Second, synthesis can be time consuming and peers in a semantic web are likely to need fast responses. Third, and perhaps most importantly, sharing information about useful interactions is a way to propagate experience about effective knowledge sharing - so obtaining a model of interaction that has been widely used and is popular with one's peers may in many cases be better (and much easier) than building one from scratch. We now consider two key enablers for this from of sharing: dynamic ontology matching and peer-to-peer interaction model sharing.

3.1 Dynamic Ontology Matching

In our introduction we argued that traditional methods of ontology matching do not scale to open knowledge sharing systems. Our focus on interaction has not, however, eradicated the issue. Instead, we have a different form of ontology matching problem to consider: matching terms that appear dynamically in the course of an interaction. Being autonomously and independently defined inside each peer, most of these terms will be semantically heterogeneous. Thus, while one peer could have $expert\_finder$ as its service role, the others could have $person\_finder$, $expertICT\_finder$, $expert\_broker$, and so on. Notice that while the first and the fourth role denote services which are essentially equivalent, the second is more general than $expert\_finder$, while the third is less general. It is a fact that these terms are always used in the context of some local, a priori defined, often left implicit, ontological description of the world. And this influences not only the specific equivalent terms used to describe a concept but also the level of generality of the concept itself.

The solution we propose is to construct semantic mappings (*e.g.* more or less general, equivalent to, disjoint from) existing between the terms used during interaction. One such example is the mapping $\langle expert\_finder, person\_finder, LG\rangle$, stating that $expert\_finder$ is less general (LG) than $person\_finder$. These mappings are those defined and used in C-OWL [14]. Their main advantage over "syntactic mappings", namely mappings which return an affinity measure in the interval [0,1] (see, e.g., the mappings constructed by the state of the art system COMA [9]), is that the information carried by the semantic relation can then be exploited in many ways, for instance when fixing mismatches (see "Term matching" below).

We discover semantic mappings, using the method implemented in the S-Match system [7]. This is applied in at least three different phases:

– *Role matching*: Aligns the different ways in which roles are described when initiating or joining an interaction. An example is the mapping $\langle expert\_finder, person\_finder, LG\rangle$.
– *Term matching*: Aligns (structured) terms within the clause defining a role in order to undertake an interaction. An example is matching $get\_address$, which in one peer could take two arguments (*e.g.* name of a person and his/her address) and in another three arguments, where the third argument (*e.g.* $Type\_of\_Comm$) could discriminate whether we need an address for personal or work communication.
– *Query / Answer matching*: Takes place when running an interaction model and deals with the semantic heterogeneity arising from the statement of a query and in the values returned in its answers. For example, an interaction model specifying that the $address\_finder$ needs to look up the address for Stephen Salter by invoking the $get\_address('Stephen\ Salter', A)$ operation is not guaranteed (as we are with the ontology of messages) to match perfectly to the operation the peer actually can perform. Perhaps the operation used by the peer is $find\_address$ and the surname is expected first (as in $'Salter, Stephen'$).

The kind of (semantic) matching that we need here differs from the previous approaches in that it is not done once and for all, at design time on statically defined ontologies, but, rather, it is performed at run-time. This moves the problem of *ontology* (and data) *integration*, widely studied in the literature, to the problem of *ontology coordination*. As discussed in [6], the problem of data and ontology coordination is characterised by various new difficulties (beyond the obvious requirement of very fast response times). Coordination is dynamic so exactly what needs to be coordinated depends on what is interacting. Peers have partial knowledge of their network so cannot always predict which other peers will interact with them. Ontology matches made in this way without prior consensus are therefore intended to support query answer sets that are *good enough* for the interaction in hand but not necessarily complete or always correct.

This form of dynamic semantic matching might at first sight seem a harder problem than conventional static matching. We can, however, utilise our coordination framework to turn it into a more limited, easier problem. From Principle 1, interaction models are shareable resources. So too are mappings used with them. This cuts down the number of new mappings that are needed as models are reused. Furthermore, the context in which mappings are constructed is much more limited than with traditional ontology mapping. Since the purpose of mappings is to ensure that a specific interaction model functions correctly then the only issue is whether the meaning associated with the particular use of an operation required of a peer corresponds to an operation it can do. This is a much less demanding task than the general task of mapping entire ontologies between peers, for two reasons. First, we need consider only the fragment of the ontology that applies to a specific interaction. Second, the commitments we make in a mapping can be weaker and more easily judged: for instance, mapping "visit to Italy" to "holiday trip" may make perfect sense for an interaction with a holiday service even though the mapping does not hold in general.

Principle 2. *Models of interactions are developed locally to peers. However, they must be shared in order to achieve interaction coordination. This is achieved by dynamically matching, at run time, terms in theinteraction models. This happen both when synthesising them and when running them to answer specific queries. Dynamic semantic matching does this by considering the terms in the context (defined as a local ontology) of the involved peers.*

3.2 Interaction Model Sharing and Discovery

The obstacle to a peer wishing to acquire an interaction model in an open system is knowing who to ask and how to ask. We cannot expect a peer to know all other peers or to know the details of an interaction. We can expect, however, that each peer knows about some distributed discovery service(s) and that each peer knows some features of the interaction in which it is willing to participate. These features need not be complex - keyword matching may suffice. In our running example, a peer with no knowledge of how to find addresses of experts might ask the discovery service for interaction models described by the keywords

$\{expert, address\}$. The task of the semantic discovery system is then to locate interaction models with these features and peers that want to participate in these interactions across the peer network.

Principle 2 separates interaction models from services, with the advantage that interactions can be shared. To make them work, however, they must connect to specific services. This requires choice. In our running example we must choose, when binding F (the $address\_finder$) whether we use $uk\_address$ or $univ\_ed$? That may depend on the names suggested as wave energy experts by the $expert\_finder$ service. If 'Stephen Salter' were a name suggested then $univ\_ed$ might be the best choice (since Salter is a professor at Edinburgh). If not, then $uk\_address$ might be a better bet because it is more general. Hence the context set by earlier interactions conditions subsequent choices of services.

There are three (non–exclusive) ways to tackle this issue:

Sharing of experience between peers: The most direct way to overcome problems like those above is (like traditional Web search engines) to refer to records of past experience. If we know, for example, that someone else had used the LCC specification in our example successfully with the variable bindings $E = e\_find$, $F = univ\_ed$, $X = 'Scotland'$ and $D = 'wave\ energy'$ then if we were to ask a similar query we might follow a similar binding pattern. A detailed description of the method currently used for this appears in [2].

Using the semantic discovery service: To facilitate ranking of peers or interaction models, the semantic discovery service can calculate and maintain statistical information about keywords and contexts from all interaction models and all peers in the system. A description of the semantic discovery service appears in [19].

Collaborative recommendation of interaction models and services: By identifying emergent groups amongst peers, based on sharing of interaction models between peers, groups of users that are likely to use services under similar contexts can be inferred. In addition we can require that the local ontologies of peers in the same cluster (or at least those segments relevant to the candidate interaction protocols) can be automatically mapped to each other. In this setting the problem of choosing the best candidate component is reduced to collaborative filtering. The appropriateness of interaction models retrieved by the system and specific services can be assessed based on the frequency of their use within the community, while the added mapping requirement can ensure that the input parameters will be provided in the format expected by the service.

Principle 3. *The choice of interaction models and peers to occupy roles in them is determined by a distributed discovery service. Evidence of role performance in interactions may be routed to this service. Interaction models in need of role performers and role performers in need of related interaction models consult this service.*

In recent years, a wide variety of resource discovery systems have been proposed and developed. Most of them, like UDDI and Napster, support attribute-based retrieval. Their major disadvantage is that they are centralised approaches. To alleviate these problems, many peer-to-peer indexing systems have been proposed, and basic methods such as distributed hash tables, latent semantic indexing and semantic overlays have been developed.

We assume that data or services are described through sets of terms which we call *descriptions* and that the system contains a large number of such descriptions. Our routing methods extend the work described in [18] to provide statistical information about terms. Initially, peers use a distributed hash table to muster data and group descriptions by term. Since we make no assumptions of shared ontologies, different terms may be used for similar concepts. In [17] a method for automatic word sense discrimination is proposed. Words (or in our case terms) and their senses are represented in a real-valued high-dimensional space where closeness in the space corresponds to semantic similarity. As input for this method we use the descriptions for each term. From the representations of terms in the high-dimensional space, we can then extract information about term generality, term popularity, related terms and homonyms.

3.3 Visualisation for User Interaction and Interaction Monitoring

There are two constraints in the interaction model for our running example of Figure 2: $get\_experts(X, D, S)$ in the third clause and $get\_address(P, A)$ in the fourth clause. We have assumed that each of these would be satisfied automatically via a service call. Sometimes, however, constraints may need to be satisfied by human interaction if all automated means of constraint satisfaction fail. A variant of our example might involve the satisfaction of $get\_experts(X, D, S)$ by a human expert who expresses an opinion on the set, S, of experts in a given country, X, in domain of expertise, D. When these interactive constraints must be satisfied then it is necessary to link interaction model specifications to visualisation specifications (which then can be interpreted via a Web browser or similar mechanism). There are several (non-exclusive) ways to do this:

– By carrying the visualisation specification with the interaction model. This allows interaction model designers to customise visualisations to interactions.
– By providing alternative visualisation methods on peers. This allows limited local customisation to account for style choices.
– By building customised visualisers for very heavily used interaction models. This is appropriate for tasks where the visualisation is the inspiration for the interaction model - for example if we wished to have a complex geospatial visualisation on peers but maintain consistency across peers of information viewed within that visualisation framework.
– By generating visualisation directly from the structure of an interaction model. This is appropriate for tasks such as monitoring the state of an interaction or investigating a failure - a facility not essential to all users but essential for some.

Principle 4. *Interaction models must permit versatility in visualisation: providing default visualisations for common structures but also allowing customisation of visualisation by both peers and interaction model designers. This can be* *achieved by adopting a standard, declarative markup language for visualisation that each peer may interpret without needing to understand the deeper semantics of the constraints themselves.*

4 A Minimal, Most General Interaction Model

We have used the interaction model of Figure 2 as an illustrative example of the various reasoning components needed for peer-to-peer discovery, sharing and collaboration. This example is domain specific (as are most interactions) but we can also specify more general forms of interaction. The most general of these describes how a peer manages other interaction models.

Figure 3 gives a minimal specification of peer interaction (reduced to its essentials to save space). It describes the key roles of a peer: discovery and sharing of interaction models; and collaboration driven from interaction models communicated between peers. In the definition: $\mathcal{P}$ is an interaction model; $\mathcal{S}$ is the state of the interaction in which the peer currently is engaged; $\mathcal{O}$ is the set of onology mapping applying to $\mathcal{S}$; M is a message; X, is the unique identifier of a peer; $locate(K, \mathcal{P})$ means the peer can find a $\mathcal{P}$, described by keyword list, K; $add\_to\_interaction\_cache(\mathcal{P})$ adds $\mathcal{P}$, to the cache known to the peer; $in\_interaction\_cache(\mathcal{P})$ select $\mathcal{P}$, from the cache known to the peer; $match(M, \mathcal{P}, \mathcal{S}, \mathcal{O}, M', \mathcal{P}', \mathcal{S}', \mathcal{O}')$ extends $\mathcal{O}$, adapting $(\mathcal{P}, \mathcal{S})$, to peer giving $(\mathcal{P}', \mathcal{S}', \mathcal{O}')$; $expand(M, \mathcal{P}, \mathcal{S}, \mathcal{S}', M', Z)$ expands $\mathcal{S}$ given M, yielding M' sent to per Z; and $completed(M, \mathcal{P}', \mathcal{S}')$ means that$M$ completes this interaction for this peer.

$$
\begin{array}{l}
a(peer, X) :: \\
\quad (a(discoverer, X)\ or\ a(collaborator, X)\ or\ a(sharer, X))\ then \\
\quad a(peer, X) \\
\\
a(discoverer, X) :: \\
\quad \begin{pmatrix} descriptors(K)\ \Leftarrow\ a(discoverer, Y)\ then \\ discovered(\mathcal{P})\ \Rightarrow\ a(discoverer, Y) \leftarrow locate(K, \mathcal{P}) \end{pmatrix}\ or \\
\quad \begin{pmatrix} descriptors(K)\ \Rightarrow\ a(discoverer, Y)\ then \\ add\_to\_interaction\_cache(\mathcal{P}) \leftarrow discovered(\mathcal{P})\ \Leftarrow\ a(discoverer, Y) \end{pmatrix} \\
\\
a(collaborator, X) :: \\
\quad m(M, \mathcal{P}, \mathcal{S}, \mathcal{O})\ \Leftarrow\ a(collaborator, Y)\ then \\
\quad \begin{pmatrix} m(M', \mathcal{P}'', \mathcal{S}'', \mathcal{O}')\ \Rightarrow\ a(collaborator, Z) \leftarrow \begin{pmatrix} match(M, \mathcal{P}, \mathcal{S}, \mathcal{O}, M', \mathcal{P}', \mathcal{S}', \mathcal{O}') \wedge \\ expand(M', \mathcal{P}', \mathcal{S}', \mathcal{P}'', \mathcal{S}'', Z) \end{pmatrix}\ or \\ m(M, \mathcal{P}, \mathcal{S}, \mathcal{O})\ \Rightarrow\ a(collaborator, Z) \leftarrow routing(M, \mathcal{P}, \mathcal{S}, \mathcal{O}, Z)\ or \\ null \leftarrow completed(M', \mathcal{P}', \mathcal{S}') \end{pmatrix} \\
\\
a(sharer, X) :: \\
\quad share(\mathcal{P})\ \Rightarrow\ a(sharer, Y) \leftarrow in\_interaction\_cache(\mathcal{P})\ or \\
\quad add\_to\_interaction\_cache(\mathcal{P}) \leftarrow share(\mathcal{P})\ \Leftarrow\ a(sharer, Y)
\end{array}
$$

Fig. 3. Interaction model for a basic peer

Figure 3, of course, is not complete because it does not define important aspects of routing, ontology matching, *etc.* that we discussed in earlier sections. For these the reader is referred to the papers cited in appropriate earlier sections. Our point here is that even the basic knowledge sharing operations of a peer can be understood in terms of shareable interaction models.

5 The OpenKnowledge Kernel

The previous sections are language oriented - they discuss the issues tackled in the OpenKnowledge project with respect to interaction models expressed in LCC. Many different systems could address these issues but the first (to our knowledge) actually to do is the OpenKnowledge kernel system, currently available to download as open source Java code from `www.openk.org`. In this section we briefly sketchthe main functional elements of the kernel system from the point of view of the subscribe-bootstrap-run cycle through which interactions are deployed. Readers with an interest in more detailed description of the kernel are referred to the tutorial and manual sections of `www.openk.org`.

Interactions in OpenKnowldge take place via a cycle of subscription (when peers say they want to take part in interactions); bootstrapping (to initiate a fully subscribed interaction) and running (to perform the bootstrapped interaction):

Subscription to an interaction model: When a peer needs to perform a task it asks the discovery service for a list of interaction models matching the description of the task. Then, for each received interaction model, the peer compares the (Java) methods in its local component library with the constraints in the entry role in which it is interested. If the peer finds an interaction model whose constraints (in the role the peer needs to perform) are covered by these methods, then the peer can subscribe (via subscription negotiator) to that interaction model in the discovery service. The subscription, through a subscription adaptor, binds the Interaction Model to a set of methods in the peer. A subscription can endure for only a single interaction run or for many, possibly unlimited, interaction runs (for example, a buyer will likely subscribe to run a purchase interaction once, while a vendor may want to keep selling its products or services).

Bootstrapping an interaction: When all the roles in the interaction model have subscriptions, the discovery service selects a random peer as a coordinator. The coordinator then bootstraps and runs the interaction. The bootstrap involves first asking the peers who they want to interact with, among all the peers that have subscribed to the various roles, then creating a team of mutually compatible peers and finally - if possible - asking the selected group of peers to commit to the interaction.

Running an Interaction: This part of the cycle is handled by the randomly chosen coordinator peer. The coordinator peer runs the interaction locally with messages exchanged between local proxies of the peers. However, when the coordinator encounters a constraint in a role clause, it sends a message, containing the constraint to be solved, to the peer performing the role, The subscription adaptor on the peer calls the corresponding method - found during the comparison at subscription time (see above). The peer then sends back a message to the coordinator with the updated values of variables and the boolean result obtained from satisfying the constraint. The kernel's matcher allows the components on the peer and the interaction models to be decoupled. The peer compares the constraints in the roles in which it is interested with the methods in its local components and creates a set of adaptors that maps the constraint in the roles to similar methods.

The OpenKnowledge kernel is intended as the main vehicle for deploying LCC and as the point of reference for programmers (particularly Java programmers) who wish to extend on our framework. Although the current paper focuses on issues connected to deployment of coordination in a peer to peer setting, an equally important aspect of our use of interaction models is at the level of specification and analysis. Here we have found that by viewing interactions as shareable specifications we can re-apply traditional formal methods in novel ways, for example in model checking [13], matchmaking [10], dynamic ontology mapping [3] and workflow [11]. In this activity it is crucial to have a compact, declarative language in which we can specify interactions independent of the infrastructure used to deploy them.

6 Comparison to Current Paradigms

In the main part of this paper we have motivated, through example, the use of shared, explicit models of interaction to provide context for knowledge sharing. We used the aspirations of the semantic web community as a focus for our arguments but our approach has relevance more broadly across communities involved in the coordination of knowledge sharing. We consider some of these below.

6.1 The Data-Centric and Process-Centric Semantic Web

One view of the Semantic Web is data-centric, where nodes are data sources with which we associate formal specifications of content. The onus is on curators of data (or engineers of services supplying data) to author their specifications appropriately so that generic systems can discover and use data, guided by this additional information. The intention in doing this is to describe key aspects of the semantics of content - so called, semantic annotations. The difficulty in practise with using only semantic annotations is that to gain consensus on what the annotations should be it is necessary for them to be used for practical purposes by the peer group to which they are relevant. From this it follows that the data-centric paradigm needs to be supported by a way of sharing patterns of usage and knitting them into semantic annotations. The interaction models described in this paper are a means of expressing such patterns. Peer-to-peer routing makes it possible to share these on a large scale. Various forms of ontological alignment (including manual alignment) can then be applied to allow peers to select (and collectively reinforce) specific patterns of usage that work when combining data.

The need to represent potential interactions has long been recognised, hence the process specification elements of OWL-S. In OWL-S, however, an interaction process is associated with a service (and with its data) rather than being separately defined. By separating interaction specifications from data annotations we obtain three crucial advantages:

- We no longer have to define generic processes for services. Instead we expect to have many, perhaps very domain specific, interaction models that we then align more narrowly with services.
- We no longer have to aim for broad ontological consensus across services because data is used via narrow alignments.
- By losing the above two restrictions we are able to knit together services with less sophisticated formal languages.

6.2 Web Service Architecture

Our approach is intended to complement and extend a traditional Web service architecture by addressing a number of restrictions. The key extensions that we are proposing, and the restrictions that they address, are summarised below:

- The Web Service Architecture, while distributed, is not inherently peer-to-peer. In particular, there is no support for efficient routing of messages between services, service discovery is performed in a centralised manner using registries, and there is the assumption that services will always be available at a fixed location. In our Peer-based architecture we relax these restrictions. We provide efficient query routing between components to prevent bottlenecks, we support component discovery using distributed search techniques, and we can cope with components that are not always available through dynamic substitution.
- The lightweight interaction models that we define avoid problems associated with dynamic service composition. Our models define precisely how interaction should be performed with individual components, and also how composition of components should be performed. We do not rely upon complex planning operations or require the construction of detailed workflows by users, although our methods do not exclude these methods where appropriate.
- The basic Web services architecture does not contain any support for assessing trust across services when conducting interactions. Because our methods maintain explicit models of interaction to coordinate services we can apply a repertoire of trust assessment methods to these: from evidence based or provenance-based methods through to methods based on statistical analysis (on groups of interactions) or social analysis (on groups of peers with shared interactions). Importantly, we can associate different measures of trust with appropriate interactions, since one measure will not fit all situations.

6.3 Grids

In [1] three generations of grids are identified: first generation, where proprietary solutions are aimed mainly at sharing computational resources of large supercomputing centres for high-performance applications; second generation, where (through middleware and standardisation) grids have become more ubiquitous; and third generation, which shifts to a service-oriented architecture, and the use of meta-data to describe these services. In this third generation, services are given well-defined, machine-processable meaning so as to enable autonomic configuration of the grid and assembly of services - the so-called "semantic grid" elaborated in [5].

Our approach is consistent with the semantic grid vision but without our methods being predicated on sophisticated underlying Grid infrastructure. Traditional grid systems connect together specific services (or types of service) in stable networks - the aim being to do as much as possible to make these networks robust, secure and resistant to failure. We concentrate on specifying interactions, which may take place with different combinations of services in an open, peer-to-peer environment where the only essential infrastructure requirement is the ability to pass messages between peers. In this sense, our aim is an "everyman's grid" in the sense that we aim to maintain integrity of interaction (a key grid objective) without requiring specialist (centralised) infrastructure or computing resources to do so and at a very low entry cost.

7 Conclusions

The need for coordinated interactions between software components is growing quickly with the increase in numbers and diversity of programs capable of supplying data on the Internet. Although multi-agent, semantic web and grid communities have traditionally taken different approaches to tackling this problem, we have argued in this paper that a substantial area within these communities is (from a coordination point of view) a shared problem that may be tackled by developing shareable, formal models of interaction. In Section 2 we described a simple language (LCC) for this purpose. In Section 3 we described the demands placed on this language for automated inference during knowledge sharing. To conclude the language description, we use LCC to describe the bare essentials of the peer interaction process; then in Section 5 we briefly describe the implemented OpenKnowledge kernel system currently available from `www.openk.org`. Finally, in Section 6, we compared this approach across semantic web, web service and grid approaches to coordination. Our aim throughout has been to demonstrate that a basic, common core of interaction specification is appropriate across these areas.

The methods described in this paper have already been applied to a variety of domains. For example, [20] describes how to use interaction models for experiment specification in astrophysics and [15] describes a novel result in protein structure prediction using our methods. Despite these early successes we still have a long way to go before achieving these sorts of peer to peer coordination routinely on a large scale. What we now know is that the basic infrastructure can be built. What remains to be seen is whether this infrastructure has resonance with the social settings in which people wish to share knowledge.

Acknowledgements

This research is funded by the OpenKnowledge project (`www.openk.org`).

References

1. Berman, F., Fox, G., Hey, A. (eds.): Grid Computing: Making the Global Infrastructure a Reality. John Wiley & Sons, New York (2003)
2. Besana, P., Robertson, D.: How service choreography statistics reduce the ontology mapping problem. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ISWC 2007. LNCS, vol. 4825, pp. 44–57. Springer, Heidelberg (2007)
3. Besana, P., Robertson, D.: How service choreography statistics reduce the ontology mapping problem. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ISWC 2007. LNCS, vol. 4825, pp. 44–57. Springer, Heidelberg (2007)
4. Blyth, J., Deelman, E., Gil, Y.: Automatically composed workflows for grid environments. IEEE Intelligent Systems (July/August 2004)
5. De Roure, D., Jennings, N.R., Shadbolt, N.R.: The semantic grid: A future e-science infrastructure. In: Berman, F., Fox, G., Hey, A.J.G. (eds.) Grid Computing - Making the Global Infrastructure a Reality, pp. 437–470. John Wiley and Sons Ltd., Chichester (2003)
6. Giunchiglia, F., Zaihrayeu, I.: Making peer databases interact: A vision for an architecture supporting data coordination. In: Klusch, M., Ossowski, S., Shehory, O. (eds.) CIA 2002. LNCS (LNAI), vol. 2446. Springer, Heidelberg (2002)
7. Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-match: an algorithm and an implementation of semantic matching. In: Bussler, C., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, pp. 61–75. Springer, Heidelberg (2004)
8. Halpen, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. Journal of the ACM 37(3), 549–587 (1990)
9. Do, H.H., Rahm, E.: COMA - a system for flexible combination of schema matching approaches. In: Proceedings of Very Large Data Bases Conference (VLDB), pp. 610–621 (2001)
10. Lambert, D., Robertson, D.: Matchmaking and brokering multi-party interactions using historical performance data. In: Fourth International Joint Conference on Autonomous Agents and Multi-agent Systems (2005)
11. Li, G., Chen-Burger, J., Robertson, D.: Mapping a business process model to a semantic web services model. In: IEEE International Conference on Web Services (2007)
12. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17), 3045–3054 (2004)

13. Osman, N., Robertson, D.: Dynamic verification of trust in distributed open systems. In: Twentieth International Joint Conference on Artificial Intelligence (2007)
14. Bouquet, P., Giunchiglia, F., Van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-OWL: contextualizing ontologies. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 164–179. Springer, Heidelberg (2003)
15. Quang, X., Walton, C., Gerloff, D., Sharman, J., Robertson, D.: Peer-to-peer experimentation in protein structure prediction: an architecture, experiment and initial results. In: International Workshop on Distributed, High-Performance and Grid Computing in Computational Biology (2007)
16. Robertson, D.: Multi-agent coordination as distributed logic programming. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 416–430. Springer, Heidelberg (2004)
17. Schutze, H.: Automatic word sense discrimination. Computational Linguistics 24(1), 97–123 (1998)
18. Siebes, R.: pnear - combining content clustering and distributed hash tables. In: Proceedings of the IEEE 2005 Workshop on Peer-to-Peer Knowledge Management, San Diego, CA, USA (July 2005)
19. Siebes, R., Dupplaw, D., Kotoulas, S., Perreau de Pinninck, A., van Harmelen, F., Robertson, D.: The openknowledge system: an interaction-centered approach to knowledge sharing. In: Proceedings of the 5th International Conference on Cooperative information Systems, Portugal (November 2007)
20. Walton, C., Barker, A.: An Agent-based e-Science Experiment Builder. In: Proceedings of the 1st International Workshop on Semantic Intelligent Middleware for the Web and the Grid, Valencia, Spain (August 2004)