Selecting Web Services Statistically

David Lambert and David Robertson

School of Informatics
University of Edinburgh
d.j.lambert@sms.ed.ac.uk dr@inf.ed.ac.uk

Abstract. Service oriented computing offers a new approach to programming. To be useful for large and diverse sets of problems, effective service selection and composition is crucial. While current frameworks offer tools and methods for selecting services based on various user-defined criteria, little attention has been paid to how such services act and interact. Similarly, the patterns of interaction might be important at a level other than that of the user-programmer. Semantic agreement between services, and the patterns of interaction between them, will be an important factor in the usability and success of service composition. We argue that this cannot be guaranteed by logic-based description of individual services. We have developed a simple but apparently effective technique for selecting agents and interactions based on evidence of their prior performance.

1 Introduction

Service oriented computing is already a key part of e-Science, business, and government computing. Web services in particular offer a compelling vehicle for distributing software functionality, offering a common platform for traditional remote procedure call, developing Grid services, and nascent agent technologies. They enable access to distributed resources such as databases, compute servers, and physical objects like telescopes; obviate the difficulties of distributing, installing, and ensuring currency of software that must otherwise be deployed on users' machines; and allow contracts and virtual organisations to be constructed between organisationally distinct domains.

Tools like Taverna [1] make it straightforward for users to construct workflows and select services to perform them. However, the available services are generally either hard-coded into the tools, or manually acquired from web pages or UDDI registries. The problem of discovering suitable services for a task is known as the connection problem, and one that the multi-agent community long ago automated by introducing middle-agents [2], which provide a meeting point for service providers and clients. Doing this for systems as open as web services, however, is a challenge that is somewhat greater than that faced by those working with typically closed, laboratory-bound multi-agent systems.

Web, Grid, and agent services are currently treated as fungible black boxes, when there is justification to believe they are not. In this paper, we present a technique which allows matchmakers to construct an effective interaction pattern and populate it with an optimal set of services, in a manner that makes minimal demands on the user.

2 The Madness of Crowds

Middle-agents [2] connect clients, as service requesters, with service providers, by providing some mechanism for matching providers' capability advertisements with clients' requests for service. One of the most common types of such middle-agents is the matchmaker, on which we focus. A matchmaker is privy to both advertisements and requests, as opposed to say a yellow-pages like directory where clients can inspect adverts in privacy. Almost all matchmaking research to date has focused on the mechanisms for describing services and requirements, such as capability description languages [3]. This is also the favoured approach for the semantic web, which uses OWL-S [4] descriptions of the service. Using taxonomies of services, descriptions of inputs and outputs, and planning-like descriptions of pre-conditions and effects, the idea is that a matchmaker can reason about the relative merits of advertising services and select those that best suit the client.

Despite the obvious utility of logic based approaches, it is far from clear that they can in practice capture all the pertinent system features in a complex world. Where many individuals, companies, and organisations offer ostensibly similar services, it is unlikely that many services will fully match their specification, or perform their task equally well [5]. In open systems, one cannot rely on the intelligence and familiarity of the expert, the insight of the designers and implementers, nor goodwill between investigators. Some particular reasons for the insufficiency of logic-based descriptions are:

- *The capability description language lacks expressiveness.* This does not imply a criticism of the language: it is unreasonable to expect any general purpose capability description language to allow the communication of arbitrarily complex capabilities and restrictions in every imaginable domain. However, it would frequently be possible for matchmakers, especially domain-specific ones, to discover such constraints.
- *User ignorance of ability of the language to express a characteristic, or of the effect of declaring it.* As expressible features or limitations become more complex, and services more common, it becomes increasingly likely that a user would be unaware of her ability to aid the matchmaker.
- *User expectation that the information will not be used by clients or matchmakers.* In an negative example of 'early-adopter syndrome', it is not unreasonable to expect service providers will refrain from supplying this kind of data until they observe a significant portion of the agent ecosystem using it.
- *Not wanting to express particular information.* In some instances, there is an incentive for service providers to keep the description of their services as general as possible, though not to the extent of attracting clients they has

no possibility of pleasing. Alternatively, the provider may not wish to be terribly honest or open about her service's foibles.
- *Comprehensive descriptions too expensive to generate or use.* Even if none of the above hold, it would often simply not be worthwhile for the service provider to analyse and encode the information. Further, in the case of web, semantic web and Grid services, it is reasonable to expect that users are discouraged by standards flux from investing much time in this endeavour.

We claim that even with the best will, it will often simply be too expensive in time, money, computation, or human brainpower, to fully describe services. Even then, as is shown in [6], it is not unreasonable to expect the reasoners (and hence any matchmakers using them) to differ in their interpretation in some cases.

Compounding this difficulty of correctly describing individual services is the one of finding agents that work well together. Our aim is to enable interactions between two or more agents, where one or more services must be found to satisfy the interaction's initiator. Why might this happen, and what can we do to overcome this obstacle to widespread use of multiple-service interactions in real-world operation?

- *'Social' reasons.* For instance, different social communities, or communities of practice, may each cluster around particular service providers for no particular reason, yet this would result in improved performance on some tasks if agents were selected from the same social pool.
- *Strategic (or otherwise) inter-business partnerships.* For example, an airline may have a special deal with other airlines or car-hire companies that would lead to a more satisfied customer.
- *Components designed by same group.* Organisations that seem to have nothing in common may well be using software created by a single group. Such software would be more likely to inter-operate well than software from others.
- *Different groups of engineers held differing views of a problem, even though the specification is the same.* Thus, the implementations are subtly incompatible, or at least do not function together seamlessly.
- *Particular resources or constraints shared between providers.* For example, in a Grid environment, a computation server and a file store might share a very high bandwidth connection, leading to improved service.
- *The inter-relationship is not known to the service provider.* Some of the dependencies may be extremely subtle, or simply not obvious.
- *Ontology mapping*
  Sometimes ontology mapping will work perfectly, in other cases, it would be better to select those services that share a native ontology.
- *Gatewaying issues* It is quite likely that many services will be provided via gateways, and that these will make semantic interaction possible and affordable, but more error prone than systems designed explicitly to interact.
- *Malice* It is hardly unknown for software vendors to ensure lock in by making their software deliberately fail to interact correctly with that of other vendors.

And, of course, we have the ever-present problem of bugs, which will often manifest themselves in such a way that some collaborating services will exercise them, and others will not.

The interaction is normally seen as secondary. If, instead, we treat it as a first-class object, neither emergent from agent behaviour, nor fixed by the client or server's interaction model, we can begin to examine some of its properties. Some such techniques include model checking [7], and ontology matching [8].

If, however, we made the interaction pattern known, and used a matchmaker to select our services, as well as, we could share empirical data about performance. This is our approach. It is made possible by our choice of interaction language, LCC. While conventional agent matchmaking is done by reference to the client's service request and the advertised capabilities of the providers, we make the protocol that drives the interaction the centrepiece. This implies:

– The purpose of the interaction is captured.
– Multi-agent dialogues, far from being impossible, are the norm.
– All agents can reason about the dialogue they are in, not just the initiator or broker.

What if, instead of simply choosing an agent and using its interaction model, we chose the interaction model, too? We can then apply our agent selection technique [9] to the question of how to construct the interaction.

In addressing all these problems we can use the evidence provided by clients on the effectiveness of services, discovering the actual performance of agents in the roles they claim to perform. Gathering enough data on any given service or interaction is hard work, and likely to be beyond the ability of any single agent. It is, however, a task for which a middle agent is ideally suited.

3 Framework

3.1 Lightweight Coördination Calculus

To describe the interactions, we use a language called the Lightweight Coördination Calculus (LCC) [10]. LCC is based on the Calculus of Communicating Systems (CCS) [11], and provides a simple language featuring message passing (denoted $\Rightarrow$ for sending, and $\Leftarrow$ for receiving) with the operators *then* (sequence), *or* (choice), and $\leftarrow$ (if). An LCC protocol is interpreted in a logic-programming style, using unification of variables which are gradually instantiated as the conversation progresses. The rules governing execution of a protocol are in figure 2.

An LCC protocol consists of dialogue framework, expanded clauses, and common knowledge. The framework defines the roles necessary to conduct an interaction, along with the allowable messages and the conditions under which they can be sent. For our astronomy workflow (figure 3), the roles include *astronomer*, *astronomy-database*, and *black-hole-finder*. The expanded clauses note where each service has reached in the dialogue. The common knowledge records conversation-specific state agreed between the services.

Fig. 1. Grammar for the LCC dialogue framework

$$
\begin{aligned}
Framework &::= Clause^{*} \\
Clause &::= Agent :: Def \\
Agent &::= a(Role, Id) \\
Def &::= Agent | Message | Def \textit{ then } Def | Def \textit{ or } Def | Def \textit{ par } Def \\
Message &::= M \Rightarrow Agent | M \Rightarrow Agent \leftarrow C | M \Leftarrow Agent | C \leftarrow M \Leftarrow Agent \\
C &::= Term | C \wedge C | C \vee C \\
Id, M, Type &::= Term \\
Term &::= Variable | Atom | Number | Atom(Term^{+}) \\
Atom &::= \textit{lowercase-char alphanumeric}^{*} \\
Variable &::= \textit{uppercase-char alphanumeric}^{*}
\end{aligned}
$$

Fig. 2. Rewrite rules governing matchmaking for an LCC protocol
These rewrite rules constitute an extension to those described in [10]. A rewrite rule

$$\alpha \xrightarrow{M_i, M_o, \mathcal{P}, O, \mathcal{C}, \mathcal{C}'} \beta$$

holds if α can be rewritten to β where: M_i are the available messages before rewriting; M_o are the messages available after the rewrite; $\mathcal{P}$ is the protocol; O is the message produced by the rewrite (if any); $\mathcal{C}$ is set of collaborators before the rewrite; and $\mathcal{C}'$ (if present) is the—possibly extended—set of collaborators after the rewrite. $\mathcal{C}$ is a set of pairs of role and service name, e.g. *col*(*black-hole-finder*, *ucsd-sdsc*)}. The same rewrite rules hold regardless of the implementation of the matchmaking function *recruit*. This enables us to apply other LCC tools, such as model-checkers and the interpreter itself, without alteration while allowing us to change *recruit*, and means clients can use their own choice of matchmaker and matchmaking scheme.

$$
\begin{array}{ll}
A :: B \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} A :: E & \textit{if } B \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E \\
A_1 \textit{ or } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E & \textit{if } \neg closed(A_2) \wedge A_1 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E \\
A_1 \textit{ or } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E & \textit{if } \neg closed(A_1) \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E \\
A_1 \textit{ then } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E \textit{ then } A_2 & \textit{if } A_1 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E \\
A_1 \textit{ then } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} A_1 \textit{ then } E & \textit{if } closed(A_1) \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}', O} E \\
 & \quad \wedge collaborators(A_1) = \mathcal{C}' \\
C \leftarrow M \Leftarrow A \xrightarrow{M_i, M_i \setminus \{M \Leftarrow A\}, \mathcal{P}, \mathcal{C}, \emptyset} c(M \Leftarrow A, \mathcal{C}) & \textit{if } (M \Leftarrow A) \in M_i \wedge satisfied(C) \\
M \Rightarrow A \leftarrow C \xrightarrow{M_i, M_i, \mathcal{P}, \mathcal{C}, \mathcal{C}', \{M \Rightarrow A\}} c(M \Rightarrow A, \mathcal{C}') & \textit{if } satisfied(C) \wedge \\
 & \quad \mathcal{C}' = recruit(\mathcal{P}, \mathcal{C}, role(A)) \\
null \leftarrow C \xrightarrow{M_i, M_i, \mathcal{P}, \mathcal{C}, \emptyset} c(null, \mathcal{C}) & \textit{if } satisfied(C) \\
a(R, I) \leftarrow C \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, \emptyset} a(R, I) :: B & \textit{if } clause(\mathcal{P}, C, a(R, I) :: B) \\
 & \quad \wedge satisfied(C)
\end{array}
$$

$$
\begin{aligned}
collaborators(c(Term, \mathcal{C})) &= \mathcal{C} \\
collaborators(A_1 \textit{ then } A_2) &= collaborators(A_1) \cup collaborators(A_2) \\
collaborators(A :: B) &= collaborators(A) \cup collaborators(B)
\end{aligned}
$$

While we use LCC as our framework, this paper's contribution regarding service selection does not require it. For our purpose, LCC's key provisions are support for multi-party dialogues, and for enabling the matchmaker and client to identify an ongoing interaction and determine the agents engaged in it. These requirements could be met by many other coördination approaches or simple extensions thereof, whether in Grid or web services domains.

3.2 Incidence Calculus

We use the incidence calculus [12] for our probabilistic calculations, It is a truth-functional probabilistic calculus in which the probabilities of composite formulae are computed from intersections and unions of the sets of worlds for which the atomic formulae hold true, rather than from the numerical values of the probabilities of their components. The probabilities are then derived from these incidences. Crucially, in general $P(\phi \wedge \psi) \neq P(\phi) \cdot P(\psi)$. This fidelity is not possible in normal probabilistic logics, where probabilities of composite formulae are derived only from the probabilities of their component formulae. In the incidence calculus, we return to the underlying sets of incidences, giving us more accurate values for compound probabilities.

$$\begin{array}{llll}
i(\top) & = worlds & i(\bot) & = \{\} \\
i(\alpha \wedge \beta) & = i(\alpha) \cap i(\beta) & i(\alpha \vee \beta) & = i(\alpha) \cup i(\beta) \\
i(\neg\alpha) & = i(\top) \backslash i(\alpha) & i(\alpha \rightarrow \beta) & = i(\neg\alpha \vee \beta) = (worlds \backslash i(\alpha)) \cup i(\beta) \\
P(\phi) & = \frac{|i(\phi)|}{|i(\top)|} & P(\phi|\psi) & = \frac{|i(\phi\wedge\psi)|}{|i(\psi)|}
\end{array}$$

The incidence calculus is not frequently applied, since one requires exact incident records to use it. Fortunately, that's exactly what the matchmaker has on hand.

4 The Matchmaker

First, we will explain the overall process of executing an LCC protocol, and the matchmaker's place in it. A client has a task or goal it wishes to achieve: using either a pre-agreed look-up mechanism, or by reasoning about the protocols available, the client will select a protocol, with possibly more than one being suitable. This done, it can begin interpreting the protocol, dispatching messages to other agents as the protocol directs. When the protocol requires a message to be sent to an agent that is not yet identified, the sender queries a matchmaker to discover services capable of filling the role. These new agents we term 'collaborators'. The matchmaker selects the service that maximises the probability of a successful outcome given the current protocol type and role instantiations. The protocol is then updated to reflect the agent's selection, and the term $col(Role, Agent)$, instantiated to the requested role and newly chosen agent, is stored in the protocol's common knowledge where it is visible to the participants and the matchmaker.

The success of a protocol and the particular team of collaborators is decided by the client: on completion or failure of a protocol, the client informs the matchmaker whether the outcome was satisfactory to the client. Each completed brokering session is recorded as an incident, represented by an integer. Our propositions are ground predicate calculus expressions. Each proposition has an associated list of worlds (incidents) for which it is true. Initially, the incident database is empty, and the broker selects services at random. As more data is collected, a threshold is reached, at which point the matchmaker begins to use the probabilities.

4.1 Selecting Agents

The traditional issue is selecting agents, so we will address this first. In our scenario, an astronomer is using the Grid to examine a black hole. Having obtained the LCC protocol in figure 3, she instantiates the *File* variable to the file she wants to work with, and runs the protocol. The protocol is sent first to a *black-hole-finder* service. This service, in turn, requires an *astronomy-database* to provide the file. If a black hole is found the *black-hole-finder* service will pass the data to a visualisation service. Finally, the client will receive a visualisation or notification of failure. Where is the inter-service variation? Consider that the astronomical data file is very large, and thus network bandwidth between sites will be a crucial factor in determining user satisfaction. Thus, some pairs of database and computation centre will outperform other pairs, even though the individuals in each pairing might be equally capable. Indeed, the 'best' database and compute centre may have a dreadful combined score because their network interconnection is weak.

If we imagine how the matchmaker's incidence database would look after several executions of this workflow (most likely by different clients), we might see something like this:

$$
\begin{array}{l}
i(\mathit{protocol}(\textsc{black-hole-search}), [1, 2, \ldots, 25]) \\
i(\mathit{outcome}(\mathit{good}), [1, 2, 3, 4, 6, 10, 11, 12, 16, 22, 23, 24]) \\
i(\mathit{col}(\mathit{astronomy\text{-}database}, \mathit{greenwich}), [18, 19, 20, 21, 22, 23, 24, 25]) \\
i(\mathit{col}(\mathit{astronomy\text{-}database}, \mathit{herschel}), [10, 11, 12, 13, 14, 15, 16, 17]) \\
i(\mathit{col}(\mathit{astronomy\text{-}database}, \mathit{keck}), [1, 2, 3, 4, 5, 6, 7, 8, 9]) \\
i(\mathit{col}(\mathit{black\text{-}hole\text{-}finder}, \mathit{barcelona\text{-}sc}), [8, 9, 16, 17, 24, 25]) \\
i(\mathit{col}(\mathit{black\text{-}hole\text{-}finder}, \mathit{ucsd\text{-}sdsc}), [1, 2, 3, 4, 10, 11, 12, 13, 18, 19, 20]) \\
i(\mathit{col}(\mathit{black\text{-}hole\text{-}finder}, \mathit{uk\text{-}hpcx}), [5, 6, 7, 14, 15, 21, 22, 23]) \\
i(\mathit{col}(\mathit{visualiser}, \mathit{ncsa}), [1, 2, \ldots, 25])
\end{array}
$$

Each $i(\mathit{proposition}, \mathit{incidents})$ records the incidents (that is, protocol interactions or executions) in which the proposition is true. We can see that the BLACK-HOLE-SEARCH protocol has been invoked 25 times, and that it has been successful in those incidents where *outcome*(*good*) is true. Further, by intersecting various incidences, we can compute the success of different teams of agents, and obtain predictions for future behaviour. Let us examine the performance of the Barcelona supercomputer:

$$
\begin{array}{rcl}
i(\mathit{col}(\mathit{black\text{-}hole\text{-}finder}, \mathit{barcelona\text{-}sc}) \wedge \mathit{outcome}(\mathit{good})) & = & \{16\} \\
P(\mathit{outcome}(\mathit{good}) | \mathit{col}(\mathit{black\text{-}hole\text{-}finder}, \mathit{barcelona\text{-}sc})) & = & \frac{|\{16\}|}{|\{8,9,16,17,24,25\}|}
\end{array}
$$

This performance is substantially worse than that of the other supercomputers on this task not because *barcelona-sc* is a worse supercomputer than *ucsd-sdsc* or *uk-hpcx*, but because its network connections to the databases required for this task present a bottleneck, reducing client satisfaction.

From this database, the matchmaker can then determine, for a requester, which services are most likely to lead to a successful outcome, given the current protocol and services already selected. That is, the matchmaker tries to optimise

$$argmax_s P(outcome(good)|\mathcal{P}, col(r,s) \cup \mathcal{C})$$

Where $\mathcal{P}$ is the protocol, $\mathcal{C}$ is the current set of collaborators, r is the role requiring a new service selection, and s is the service we are to select.

Fig. 3. LCC dialogue framework for astronomy workflow scenario

$$\begin{array}{l}
a(\mathit{astronomer}(File), Astronomer) :: \\
\quad \mathit{search}(File) \Rightarrow a(\mathit{black\text{-}hole\text{-}finder}, BHF)\ \mathit{then} \\
\quad \left(\begin{array}{l} \mathit{success} \Leftarrow a(\mathit{black\text{-}hole\text{-}finder}, BHF)\ \mathit{then} \\ \mathit{receive\text{-}visualisation}(Thing, V) \leftarrow \mathit{visualising}(Thing) \Leftarrow a(\mathit{visualiser}, V) \end{array} \right)\ \mathit{or} \\
\quad \mathit{failed} \Leftarrow a(\mathit{black\text{-}hole\text{-}finder}, BHF)
\end{array}$$

$$\begin{array}{l}
a(\mathit{black\text{-}hole\text{-}finder}, BHF) :: \\
\quad \mathit{search}(File) \Leftarrow a(\mathit{astronomer}(File), Astronomer)\ \mathit{then} \\
\quad \mathit{grid\text{-}ftp\text{-}get}(File) \Rightarrow a(\mathit{astronomy\text{-}database}, AD)\ \mathit{then} \\
\quad \left(\begin{array}{l} \mathit{grid\text{-}ftp\text{-}sent}(File) \Leftarrow a(\mathit{astronomy\text{-}database}, AD)\ \mathit{then} \\ \mathit{success} \Rightarrow a(\mathit{astronomer}, Astronomer) \\ \qquad\qquad \leftarrow \mathit{black\text{-}hole\text{-}present}(File, Black\text{-}hole)\ \mathit{then} \\ \mathit{visualise}(Black\text{-}hole, Astronomer) \Rightarrow a(\mathit{visualiser}, V) \end{array} \right)\ \mathit{or} \\
\quad \mathit{failed} \Rightarrow a(\mathit{astronomer}(File), Astronomer)
\end{array}$$

$$\begin{array}{l}
a(\mathit{astronomy\text{-}database}, AD) :: \\
\quad \mathit{grid\text{-}ftp\text{-}get}(File) \Leftarrow a(\mathit{black\text{-}hole\text{-}finder}, BHF) \\
\quad \mathit{grid\text{-}ftp\text{-}sent}(File) \Rightarrow a(\mathit{black\text{-}hole\text{-}finder}, BHF) \leftarrow \mathit{grid\text{-}ftp\text{-}completed}(File, AD)
\end{array}$$

$$\begin{array}{l}
a(\mathit{visualiser}, V) :: \\
\quad \mathit{visualise}(Thing, Client) \Leftarrow a(\_, Requester)\ \mathit{then} \\
\quad \mathit{visualising}(Thing) \Rightarrow a(\_, Client) \leftarrow \mathit{serve\text{-}visualisation}(Thing, Client)
\end{array}$$

Note that LCC is being used only to coördinate the interaction: where appropriate, individual agents may use domain-specific protocols, such as Grid FTP, to perform the heavy lifting or invoke specific services outside of the LCC formalism and communication channel.

We have developed two algorithms for choosing services, although others are possible. The first, called RECRUIT-JOINT, fills all the vacancies in a protocol at the outset. It works by computing the joint distribution for all possible permutations of services in their respective roles, selecting the grouping with the largest probability of a good outcome.

The second approach, RECRUIT-INCREMENTAL, is to select only one service at a time, as required by the executing protocol. The various services already engaged in the protocol, on needing to send a message to an as-yet-unidentified service, will ask the matchmaker to find an service to fulfil the role at hand. RECRUIT-INCREMENTAL computes the probability of a successful outcome for each service available for role R given $\mathcal{C}$ ($\mathcal{C}$ being the collaborators chosen so far), and selects the most successful service. To illustrate RECRUIT-INCREMENTAL, imagine the workflow scenario. At first, the astronomer must ask the matchmaker to fill the *black-hole-finder* role. The *BHF* service's first action is to request the data file from an astronomy database. It therefore returns the protocol to the matchmaker, which selects the *astronomy-database* most likely to produce success, given that the *black-hole-finder* is already instantiated to *BHF*.

Both algorithms support the pre-selection of services for particular roles. An example of this might be a client booking a holiday: if it were accumulating frequent flier miles with a particular airline, it could specify that airline be used, and the matchmaker would work around this choice, selecting the best agents given that the airline is fixed. This mechanism also allows us to direct the matchmaker's search: selecting a particular service can suggest that the client wants similar services, from the same social pool, for the other roles, e.g. in a peer-to-peer search, by selecting an service you suspect will be helpful in a particular enquiry, the broker can find further services that are closely 'socially' related to that first one.

We can see from figure 5(a) that using this technique can substantially improve performance over random selection of agents which can individual meet the requirements. Which algorithm should one choose? In protocols where most roles are eventually filled, RECRUIT-JOINT will outperform RECRUIT-INCREMENTAL, since it is not limited by the possibly suboptimal decisions made earlier. RECRUIT-JOINT is also preferable when one wishes to avoid multiple calls to the matchmaker, either because of privacy concerns, or for reasons of communication efficiency. However, in protocols which rarely have all their roles instantiated, RECRUIT-JOINT can end up unfairly penalising those services which have not actually participated in the protocols they are allocated to. RECRUIT-INCREMENTAL is therefore more suitable in protocols where many roles go unfilled: total work on the broker would be reduced, and the results would probably be at least as good as for brokering all services.

4.2 Selecting Roles

So far, we have considered the case where the protocol is defined, and we simply need to select agents to fill the roles. What if the roles themselves are undefined, if the protocol is incompletely specified? What if we allowed agents to begin

Fig. 4. Algorithms

```
RECRUIT-JOINT(protocol, database)
1 roles ← ROLES-REQUIRED(protocol)
2 collaborations ← ALL-COLLABORATIONS(protocol, database, roles)
3 for $c \in$ collaborations
4         do quality[c] ← PROBABILITY-GOOD-OUTCOME(protocol, database, c)
5 return ARGMAX(collaborations, quality)
```

```
RECRUIT-INCREMENTAL(protocol, database, role)
1 for $r \in$ ACTIVE-ROLES(protocol)
2         do collaborators[r] ← COLLABORATOR-FOR-ROLE(protocol, r)
3 candidates ← CAPABLE-AGENTS(database, role)
4 for $c \in$ candidates
5         do collaborators[role] ← c
6             quality[a] ← PROBABILITY-GOOD-OUTCOME(database, collaborators)
7 return ARGMAX(candidates, quality)
```

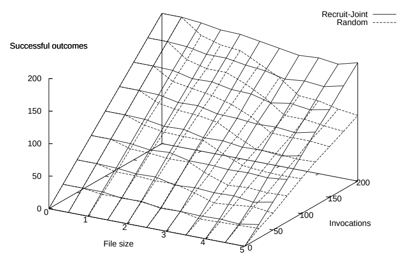
```
EMBELLISH-INCREMENTAL(protocol, database, role)
1 for $r \in$ ROLE-DEFINITIONS(protocol)
2         do role-definition[r] ← DEFINITION-FOR-ROLE(protocol, r)
3 candidates ← AVAILABLE-ROLE-DEFINITIONS(protocol,database,role) role)
4 for $c \in$ candidates
5         do role-definition[role] ← c
6             quality[c] ← PROBABILITY-GOOD-OUTCOME(database, role-definitions)
7 return ARGMAX(role-definitions, quality)
```

ARGMAX as used here does not always select the highest value. To improve the exploration of options, those entries that have low numbers of data points (i.e. have not often been selected previously) are preferentially chosen, and in other cases a random selection is sometimes made.

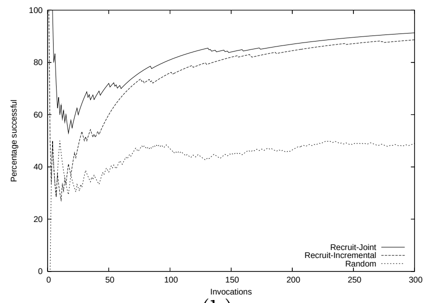
executing incomplete protocols? If the matchmaker could elaborate protocols at run time, selecting the elaboration based on prior experience? We will show one way to do this using the incidence calculus, in a very similar fashion to how we selected agents.

Roles consist of an ordering of messages, together with constraints, and moves to other roles. It might be the case that just changing the ordering might make a large difference. For instance, if one is arranging to travel to a concert, it is preferable to obtain event tickets first, then organise transport. In our example, we take to problem of booking a trip involving a flight and hotel room. The LCC protocol is shown in figure 6. If we suppose that it is a preferable course of action to book the flight then the hotel room, since hotel room costs are more

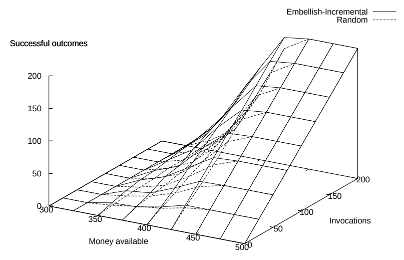
Fig. 5. Simulation results



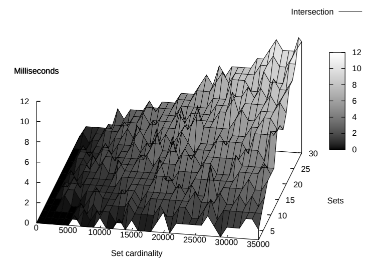
(a)



(b)



(c)



(d)

In (a), we see the improvement in task achievement using agent selection obtained using RECRUIT-JOINT versus random selection. Using the same scenario, but fixing the file size at 5 Gigabytes, (b) shows the relative performance of RECRUIT-JOINT, RECRUIT-INCREMENTAL, and random selection. We can see similar gains for selection of roles in (c), using the travel agent scenario. Performance of the underlying set calculus intersection operation is shown in (d).

flexible that flight ones, we can expect a better outcome using *flight-then-hotel* rather than *hotel-then-flight*. Figure 5(c) shows the improvement in a simulation. The algorithm used is EMBELLISH-INCREMENTAL, shown in figure 4. EMBELLISH-INCREMENTAL works similarly to RECRUIT-INCREMENTAL, adding role definitions to the protocol as those roles are required at run-time. We have not provided equivalent to RECRUIT-JOINT, since this can inflate protocols with many roles that will remain unused.

5 Discussion

Performance seems quite reasonable for very large sets. The core operation of this technique is set intersection, since for every collaboration or set of role definitions,

Fig. 6. Booking a holiday with LCC

$$
\begin{array}{l}
a(\textit{traveller}, \textit{Traveller}) :: \\
\quad \textit{book-holiday}(\textit{Src}, \textit{Dst}, \textit{Start}, \textit{End}, \textit{Money}) \Rightarrow a(\textit{travel-agent}, \textit{Agent}) \\
\qquad \leftarrow \textit{travel-details}(\textit{Src}, \textit{Dst}, \textit{Start}, \textit{End}, \textit{Money})\ \textit{then} \\
\quad \left(\begin{array}{l}
\textit{booking}(\textit{Start}, \textit{End}, \textit{Cost}) \Leftarrow a(\textit{travel-agent}, \textit{Agent})\ \textit{then} \\
\textit{matchmaking}(\textit{good}) \Rightarrow a(\textit{matchmaker}, \textit{matchmaker})
\end{array}\right)\ \textit{or} \\
\quad \left(\begin{array}{l}
\textit{failure} \Leftarrow a(\textit{travel-agent}, \textit{Agent})\ \textit{then} \\
\textit{matchmaking}(\textit{bad}) \Rightarrow a(\textit{matchmaker}, \textit{matchmaker})
\end{array}\right)
\end{array}
$$

Note that the *travel-agent* role is not specified in the client's protocol! We leave it to the matchmaker to find one. The matchmaker, let us say, has the following role definitions available to it:

$$
\begin{array}{l}
\textit{role}(\textit{flight-then-hotel}) \equiv a(\textit{travel-agent}, \textit{Agent}) :: \\
\quad \textit{book-holiday}(\textit{Src}, \textit{Dst}, \textit{Start}, \textit{End}, \textit{Money}) \Leftarrow a(\textit{client}, \textit{Client})\ \textit{then} \\
\quad \textit{book-flight}(\textit{Src}, \textit{Dst}, \textit{Start}, \textit{End}, \textit{Money}) \Rightarrow a(\textit{airline}, \textit{Airline})\ \textit{then} \\
\quad \left(\begin{array}{l}
\textit{no-flights} \Leftarrow a(\textit{airline}, \textit{Airline})\ \textit{then} \\
\textit{failure} \Rightarrow a(\textit{client}, \textit{Client})
\end{array}\right)\ \textit{or} \\
\quad \left(\begin{array}{l}
\textit{flight-booking}(\textit{Flight-Cost}) \Leftarrow a(\textit{airline}, \textit{Airline})\ \textit{then} \\
\textit{flight-available}(\textit{Src}, \textit{Dst}, \textit{Start}, \textit{End}, \textit{Money}) \Leftarrow a(\textit{airline}, \textit{Airline})\ \textit{then} \\
\textit{book-hotel}(\textit{Location}, \textit{Start}, \textit{End}, \textit{Money}) \Rightarrow a(\textit{hotel}, \textit{Hotel}) \\
\qquad \leftarrow \textit{is}(\textit{Money-Left}, \textit{Money} - \textit{Flight-Cost})\ \textit{then} \\
\left(\begin{array}{l}
\textit{hotel-booking}(\textit{Hotel-Cost}) \Leftarrow a(\textit{hotel}, \textit{Hotel})\ \textit{then} \\
\textit{booking}(\textit{Total-Cost}) \Rightarrow a(\textit{client}, \textit{Client}) \\
\qquad \leftarrow \textit{is}(\textit{Total-Cost}, \textit{Flight-Cost} + \textit{Hotel-Cost})
\end{array}\right)\ \textit{or} \\
\left(\begin{array}{l}
\textit{no-vacancy} \Leftarrow a(\textit{hotel}, \textit{Hotel})\ \textit{then} \\
\textit{failure} \Rightarrow a(\textit{client}, \textit{Client})
\end{array}\right)
\end{array}\right)
\end{array}
$$

$$
\begin{array}{l}
\textit{role}(\textit{flight-then-hotel}) \equiv a(\textit{travel-agent}, \textit{Agent}) :: \\
\quad \textit{book-holiday}(\textit{Src}, \textit{Dst}, \textit{Start}, \textit{End}, \textit{Money}) \Leftarrow a(\textit{client}, \textit{Client})\ \textit{then} \\
\quad \textit{book-flight}(\textit{Src}, \textit{Dst}, \textit{Start}, \textit{End}, \textit{Money}) \Rightarrow a(\textit{airline}, \textit{Airline})\ \textit{then} \\
\quad \left(\begin{array}{l}
\textit{no-flights} \Leftarrow a(\textit{airline}, \textit{Airline})\ \textit{then} \\
\textit{failure} \Rightarrow a(\textit{client}, \textit{Client})
\end{array}\right)\ \textit{or} \\
\quad \left(\begin{array}{l}
\textit{flight-booking}(\textit{Flight-Cost}) \Leftarrow a(\textit{airline}, \textit{Airline})\ \textit{then} \\
\textit{flight-available}(\textit{Src}, \textit{Dst}, \textit{Start}, \textit{End}, \textit{Money}) \Leftarrow a(\textit{airline}, \textit{Airline})\ \textit{then} \\
\textit{book-hotel}(\textit{Location}, \textit{Start}, \textit{End}, \textit{Money}) \Rightarrow a(\textit{hotel}, \textit{Hotel}) \\
\qquad \leftarrow \textit{is}(\textit{Money-Left}, \textit{Money} - \textit{Flight-Cost})\ \textit{then} \\
\left(\begin{array}{l}
\textit{hotel-booking}(\textit{Hotel-Cost}) \Leftarrow a(\textit{hotel}, \textit{Hotel})\ \textit{then} \\
\textit{booking}(\textit{Total-Cost}) \Rightarrow a(\textit{client}, \textit{Client}) \\
\qquad \leftarrow \textit{is}(\textit{Total-Cost}, \textit{Flight-Cost} + \textit{Hotel-Cost})
\end{array}\right)\ \textit{or} \\
\left(\begin{array}{l}
\textit{no-vacancy} \Leftarrow a(\textit{hotel}, \textit{Hotel})\ \textit{then} \\
\textit{failure} \Rightarrow a(\textit{client}, \textit{Client})
\end{array}\right)
\end{array}\right) \\
a(\textit{hotel}, \textit{Hotel}) :: \\
\quad \textit{book-hotel}(\textit{Location}, \textit{Start}, \textit{End}, \textit{Money}) \Leftarrow a(\textit{Role}, \textit{Agent})\ \textit{then} \\
\quad \textit{room-available}(\textit{Location}, \textit{Start}, \textit{End}, \textit{Money}, \textit{Cost}) \Rightarrow a(\textit{Role}, \textit{Agent}) \\
\qquad \leftarrow \textit{room-available}(\textit{Location}, \textit{Start}, \textit{End}, \textit{Money}, \textit{Cost})\ \textit{or} \\
\quad \textit{no-vacancy} \Rightarrow a(\textit{Role}, \textit{Agent})
\end{array}
$$

$$
\begin{array}{l}
a(\textit{airline}, \textit{Airline}) :: \\
\quad \textit{book-flight}(\textit{Src}, \textit{Dst}, \textit{Start}, \textit{End}, \textit{Money}) \Leftarrow a(\textit{Role}, \textit{Agent})\ \textit{then} \\
\quad \textit{flight-available}(\textit{Src}, \textit{Dst}, \textit{Start}, \textit{End}, \textit{Money}) \Rightarrow a(\textit{Role}, \textit{Agent}) \\
\qquad \leftarrow \textit{flight-available}(\textit{Src}, \textit{Dst}, \textit{Start}, \textit{End}, \textit{Money})\ \textit{or} \\
\quad \textit{no-flights} \Rightarrow a(\textit{Role}, \textit{Agent})
\end{array}
$$

$$
\begin{array}{l}
a(\textit{matchmaker}, \textit{matchmaker}) :: \\
\quad \textit{record-matchmaking-outcome}(\textit{Outcome}) \\
\qquad \leftarrow \textit{matchmaking}(\textit{Outcome}) \Leftarrow a(\textit{Role}, \textit{Agent})
\end{array}
$$

the intersection of their incidences must be computed. By using a heap-sort like intersection algorithm, this can be done in order $O(n \log n)$. Figure 5(d) shows that we can quickly calculate intersections over large sets for reasonable numbers of sizable sets.

We note here two significant problems that seem to be inescapable issues intrinsic to the problem: trusting clients to evaluate protocol performance honestly and in a conventional manner; and the problems of locating mutually coöperative services in a large agent ecology. Since individual client services are responsible for the assigning of success metrics to matchmakings, there is scope for services with unusual criteria or malicious intent to corrupt the database. The second question, largely unasked, is about the likely demographics of service provision. For some types of service, like search, we have already seen that a very small number of providers. For other tasks, a few hundred exist: think of airlines. For some, though, we may millions of service providers. Further, we must ask how many service types will be provided. Again, in each domain, we might have a simple, monolithic interface, or an interface with such fine granularity that few engineers ever fully understand or exploit it. The answers to these will impact the nature of our matchmaking infrastructure.

While our technique handles large numbers of incidences, it does not scale for very large numbers of services or roles. For any protocol with a set of roles R, and with each role having $|providers(r_i)|$ providers, the number of ways of choosing a team is $\prod_{r_i \in R} |providers(r_i)|$, or $O(m^n)$. No matchmaking system could possibly hope to discover all the various permutations of services in a rich environment, although machine learning techniques might be helpful in directing the search for groupings of services. How much of an issue this actually becomes in any particular domain will be heavily influenced by the outcomes to the issues discussed above.

6 Related Work

The connection problem arises in agent systems, semantic web, and grid environments. It is discussed in [2, 13, 14]. We consciously ignored methods like those found in [15], though they would be crucial in any real-world deployment: we believe our technique would usefully augment such systems. Similarly, several groups have attacked the workflow synthesis issue using automated planning [16]: we again suggest our method as an adjunct to other techniques, not a replacement.

Two studies have investigated the issue of using previous performance records. In [5] we first see the use of records to improve selection. In [17] this technique is combined with description logic concepts to improve matching of OWL-S requests.

Most work has been restricted to the case of selecting a single agent for a single role. Although current interactions are primarily client-server, we can imagine a future where match-made agent interactions are more distributed, involve many agents, and operate in a more peer-to-peer manner. It can be expected that these newer forms of dialogue will make even greater use of, and demands upon, matchmaking services than do current modes of employment. Our problem conception—matchmaking multiple roles for the same dialogue—is anticipated by the SELF-SERV system [18], though we believe our approach is novel in detecting emergent properties that are not known to the operator, and is more transparent, requiring less intervention (that is, specification of service parameters) from the client.

7 Conclusion

Distributed computing is becoming commonplace, and automated discovery of these systems will become crucial, too. The current, dominant model of service provision can be characterised by noting that: workflow execution happens on one machine dedicated to the purpose, whether the client's, a workflow server, or a middle agent, and other services are used as remote procedure calls; that the workflow is never exported beyond the machine; that services are selected based only on their capability as advertised through logical descriptions; and that information recorded about success or otherwise is, at best, held only by the machine which discovered it.

In this paper we made four claims: that services may not be totally described by their service advertisements; that services may interact in odd ways; that interaction patterns may be as important as the interacting objects; and that a simple, statistical matchmaker can be of help in solving all three problems.

We have shown that the successful completion of a task may depend not only on the advertised abilities of services but on their collective suitability and inter-operability. We also showed that the structure of the interaction can be important. We presented a simple, but effective, technique for detecting successful groupings of services, and choosing those interaction patterns that suit them best. We highlighted the intractability of the problem in environments with large numbers of available provider services and/or roles. We can sum up the traditional model of matchmaking as *static and action-oriented*. Our approach is *dynamic and interaction-oriented*, allowing us to respond to actual performance, and better support agent selection and protocol synthesis.

Further work remains. Practical issues of managing a large database of incidences must be resolved: can the task be distributed, either across clusters, like current Internet search engines, or in a peer-to-peer way? What scope is there for applying machine learning? Is our current description of services sufficient, and if not, how do we integrate more sophisticated notions of service description?

References

1. Oinn, T.M., Addis, M., Ferris, J., Marvin, D., Greenwood, R.M., Carver, T., Pocock, M.R., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics **20**(17) (2004) 3045–3054
2. Decker, K., Sycara, K., Williamson, M.: Middle-Agents for the Internet. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence, Nagoya, Japan (1997)

3. Wickler, G., Tate, A.: Capability Representations for Brokering: A Survey (1999)
4. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: Owl-s: Semantic markup for web services (2004)
5. Zhang, Z., Zhang, C.: An improvement to matchmaking algorithms for middle agents. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, ACM Press (2002) 1340–1347
6. Pan, Z.: Benchmarking DL Reasoners Using Realistic Ontologies. In: OWL: Directions and Experiences, Galway, Ireland (2005)
7. Osman, N., Robertson, D., Walton, C.: Run-time model checking of interaction and deontic models for multi-agent systems. In: Proceedings of the European Multi-Agent Systems Workshop, 2005. (2005)
8. Besana, P., Robertson, D., Rovatsos, M.: Exploiting Interaction Contexts in P2P ontologoy mapping. In: Proceedings of the second international workshop on peer-to-peer knowledge management. (2005)
9. Lambert, D., Robertson, D.: Matchmaking multi-party interactions using historical performance data. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05). (2005) 611–617
10. Robertson, D.: A lightweight method for coordination of agent oriented web services. In: Proceedings of the 2004 AAAI Spring Symposium on Semantic Web Services, California, USA (2004)
11. Milner, R.: Communication and concurrency. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1989)
12. Bundy, A.: Incidence calculus: A mechanism for probabilistic reasoning. Journal of Automated Reasoning **1**(3) (1985) 263–284
13. Wong, H., Sycara, K.: A Taxonomy of Middle-agents for the Internet. In: 4th International Conference on Multi-Agent Systems (ICMAS 2000). (2000)
14. Klusch, M., Sycara, K.: Brokering and matchmaking for coordination of agent societies: a survey. In: Coordination of Internet agents: models, technologies, and applications. Springer-Verlag (2001) 197–224
15. Paulucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic Matching of Web Services Capabilities. In: The Semantic Web — ISWC 2002: Proceedings. (2002)
16. Blythe, J., Deelman, E., Gil, Y.: Planning for workflow construction and maintenance on the grid (2003)
17. Xiaocheng Luan: Adaptive Middle Agent for Service Matching in the Semantic Web: A Quantitative Approach. PhD thesis, University of Maryland, Baltimore County (2004)
18. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: WWW '03: Proceedings of the twelfth international conference on World Wide Web, New York, NY, USA, ACM Press (2003) 411–421