Matchmaking and Brokering Multi-Party Interactions Using Historical Performance Data

Registration number #601*

Abstract

Matchmaking and brokering will be a crucial component of future agent and agent-like systems, such as the semantic web. Most research on matchmaking has been directed toward sophisticated matching of client requirements with provider capabilities based on capability descriptions. This is a vital mechanism for conducting matchmaking, but ignores the likelihood that in practice, and for various reasons, capability descriptions will not fully characterise the interaction behaviour of agents.

This problem is further compounded in systems with many interacting agents, all of which have idiosyncrasies. As in everyday life, some groupings of agents will be more effective than others, regardless of their individual competencies or suitability to the task. The quality of the interaction between agents is a crucial factor.

Using the incidence calculus and the lightweight coördination calculus, we show that we can easily implement matchmaking agents that will learn from experience how to select those groups known to inter-operate well for particular purposes.

1. Introduction

The issue of matchmaking is key to the deployment of many types of multi-agent system [1]. It first appeared as a research problem at the dawn of agent systems [2], and has resurfaced as a fundamental problem in newer areas like the semantic web and Grid computing.

Most of the previous research into agents has addressed interactions with only two-parties, service -requester and -provider. This bias continues on the semantic web. OWL-S [3], for instance, imagines that any interaction will be principally two-party, despite the presence of the `hasParticipants` slot, which is under-specified and unused. One can imagine interactions that are inherently multi-agent and which would thus require any matchmaker to find an appropriate *set* of agents. Indeed, recent developments in web services choreography [4] reveal the growing realisation that many real-world processes require multiple participants.

The technique presented here developed from work on the lightweight coördination calculus (LCC) [5], a language for describing multi-party dialogues, where we found it necessary to provide a middle-agent to supply collaborators for the fulfilment of protocols.

We do not commit to any particular deployment environment. While we have implemented it for LCC, we believe the approach would have value any setting where service capabilities are specified in a flexible, semantic encoding. Consequently, we see applications not only in MAS, but on the Grid [6] and semantic web, and in work-flow and peer-to-peer systems.

The remainder of this paper is organised thus: We first examine the problem in greater detail in section 2, before introducing the two key ideas we use to tackle it — the lightweight coördination calculus and the incidence calculus — in section 3. Section 4 shows how we use the incidence calculus to select an appropriate team of agents for a task. Section 5 surveys related work, and section 6 concludes.

2. Motivation *or* Three is not a crowd

The motivation for matchmaking and brokering is clear: in any environment with large numbers of agents, the only feasible mechanism for connecting those requesting a service with those willing to provide it is via middle agents [1, 7]. However, most previous work has examined only the case of selecting a single agent (or in some cases, a range of agents, leaving the final selection of the agent to the client) for a single role. To our knowledge, [8] is the only exception.

While current interactions are primarily client-server, we can imagine a future where matchmade agent interactions are more distributed, involve many agents, and operate in a more peer-to-peer manner (i.e. have a less hierarchical structure). It can be expected that these newer forms of dialogue will make even greater use of, and demands upon,

∗ The primary author is a student.

matchmaking services than do current modes of employment.

2.1. Capability description is not enough

In [8] the assumption is that agent implementations would vary in their *intrinsic ability* to complete a task, even if the task were completely specified and accurately advertised. We adopt this view, and extend it: we think that, as well as the agent's inherent talent at a task, there is also reason to believe that issues of ontological mismatch and other 'social' effects will reduce the efficacy of purely logical agent capability descriptions. Regardless of how carefully service providers specify the behaviour of their systems, there will always remain some level of 'semantic slack' such that matchmaking based on purely semantic service description can be improved by statistical or other learning techniques. In these circumstances, an approach like ours would be beneficial. Below, we examine several reasons for this encoding gap. Note that only the first is a technical problem: the rest cover scenarios where additional semantic information could be embedded in profiles but is not for non-technical reasons.

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 constraints 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 constraint, or of the effect of declaring the constraint. As constraints 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.

The inter-relationship is not known to the service provider. Some of the dependencies may be extremely subtle, or simply not obvious.

Comprehensive constraints 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.

But what underlying causes are there for these problems that are so difficult to encode? Below are some reasons why such issues might arise:

'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. This particular case underlies our example scenario, detailed in figure 3.

3. Technical preliminaries

Our matchmaker relies on two techniques developed previously: the lightweight coördination calculus, and the incidence calculus. We briefly explain these here.

3.1. Lightweight coördination calculus

The lightweight coördination calculus (LCC) [5] is a method for specifying agent interaction protocols. A generalisation of the Electronic Institutions [9] model, it is based on process calculi, specifically CCS [10]. It provides a simple message passing framework (denoted $\Rightarrow$ for sending, and $\Leftarrow$ for receiving) with the operators $then$ (sequence), or (choice), par (parallel execution), and $\leftarrow$ (if). The grammar is shown in figure 1. An LCC protocol framework

Figure 1: Grammar for the LCC dialogue framework

$$\begin{array}{rcl}
Framework & ::= & Clause, Clause+ \\
Clause & ::= & Agent :: Def \\
Agent & ::= & a(Role, Id) \\
Def & ::= & Agent \mid Message \mid Def\ then\ Def \mid \\
 & & Def\ or\ Def \mid Def\ par\ Def \\
Message & ::= & M \Rightarrow Agent \mid M \Rightarrow Agent \leftarrow C \mid \\
 & & M \Leftarrow Agent \mid C \leftarrow M \Leftarrow Agent \\
C & ::= & Term \mid C \wedge C \mid C \vee C \\
Role & ::= & Term \\
Id & ::= & Term \\
M & ::= & Term
\end{array}$$

is interpreted in a logic-programming style, using unification of variables which are gradually instantiated as the conversation progresses. Along with the dialogue framework, which specifies the various messages that can be transmitted and when, a protocol carries 'common knowledge', which is simply data, specific to a conversation, that every participant in the dialogue can access, and modify.

This style of protocol definition is flexible, and allows us to easily capture, and perform matchmaking for, multi-agent interactions, the primary contribution of our this paper. Conveniently, LCC allows us to use the same language to express various degrees of decentralisation. For example, we can convert a protocol from one in which the middle-agent functions as a broker (routing all communication through itself, delivering only the final result to the client), to a matchmaker (once the middle-agent has identified agents for the required roles, it informs the client of the decisions and plays no further part in the protocol's execution).

The protocols can be created dynamically, but for this paper we choose to situate the matchmaker system in an environment where a library of standard protocols exists. Each protocol functions as a pre-defined plan. An agent, wishing to accomplish some task (such as creating an auction, or calling a meeting, finding participants, and arranging a mutually suitable time) will select a protocol from a library, and ask its matchmaker to suggest service-provider agents for each role. In some sense, each role is equivalent to an atomic service capability, although a role carries also the responsibility of participating in a specific pattern of on-going conversation.

3.2. Incidence calculus

The incidence calculus [11] 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. The rules are given in figure 3.2. In general, $p(\phi \wedge \psi) \neq p(\phi) \cdot p(\psi)$. This fidelity is not possible in 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. The conditional probabilities of incidences drive our matchmaking, as we see in the next section.

Figure 2: Incidence calculus rules

$$\begin{array}{llll}
i(\top) & = allworlds & i(\bot) & = \{\} \\
i(\neg\alpha) & = i(\top) \backslash i(\alpha) & & \\
i(\alpha \wedge \beta) & = i(\alpha) \cap i(\beta) & i(\alpha \vee \beta) & = i(\alpha) \cup i(\beta) \\
i(\alpha \rightarrow \beta) & = i(\neg\alpha \vee \beta) = (i(\top) \backslash i(\alpha)) \cup i(\beta) & &
\end{array}$$

Probabilities are derived from incidences thus:

$$p(\phi) = \frac{|i(\phi)|}{|i(\top)|} \qquad p(\phi|\psi) = \frac{|i(\phi \wedge \psi)|}{|i(\psi)|}$$

As an example, consider the following set of incidences describing the weather in a given week:

$$\begin{array}{llll}
i(\top) & = \{mon, tue, wed, thu, fri, sat, sun\} & & \\
i(rain) & = \{mon, wed, thu, fri, sat, sun\} & & \\
i(wind) & = \{mon, thu, fri, sun\} & & \\
i(sun) & = \{tue, wed, sat, sun\} & & \\
i(snow) & = \{sat\} & & \\
i(rain \wedge sun) & = i(rain) \cap i(sun) & = & \{wed, sat, sun\} \\
i(\neg rain \vee snow) & = i(\neg rain) \cup i(snow) & = & \{tue, sat\}
\end{array}$$

To illustrate the computation of probabilities of compound formulae from the incidences, note that the probabilities of $wind$ and sun are both $\frac{4}{7}$, but their conjunctions with $rain$ (probability $\frac{6}{7}$) are different, at $\frac{4}{7}$ and $\frac{2}{7}$ respectively.

The incidence calculus is not frequently applied, since one requires exact incident records to use it. For the application at hand, however, we have detailed information about each matchmaker invocation, and the calculus provides a simple, intuitive way of dealing with the problem. More powerful mechanisms exist in the calculus for dealing with situations where knowledge is incomplete, though we do not exploit them in this paper.

4. The LCC matchmaker

In using LCC for matchmaking, we must ask how we arrive at a protocol. A client agent has a task or goal it wishes to achieve. Using either a pre-agreed lookup mechanism, or by reasoning about the protocols available, the agent will select a protocol: more than one might be suitable. This done, it must recruit a matchmaker to propose other agents to fill the various roles in the protocol. These other agents we term 'collaborators' and denote $col(Role, AgentId)$.

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 as an integer. Our propositions are ground predicate calculus expressions, e.g. $outcome(good)$, $col(bank, citibank)$. Each proposition has an associated list of worlds (incidents) for which it is true. For our example scenario (see figure 3), the database might look like this:

$i(allworlds, [1, 2, \ldots, 25])$
$i(protocol(\text{BLACK\_HOLE\_SEARCH}), [1, 2, \ldots, 25])$
$i(outcome(good), [1, 2, 3, 4, 6, 10, 11, 12, 16, 22, 23, 24])$
$i(col(astronomy\_database, keck), [1, 2, 3, 4, 5, 6, 7, 8, 9])$
$i(col(astronomy\_database, herschel), [10, 11, 12, 13, 14, 15, 16, 17])$
$i(col(astronomy\_database, greenwich), [18, 19, 20, 21, 22, 23, 24, 25])$
$i(col(black\_hole\_finder, ucsd\_sdsc), [1, 2, 3, 4, 10, 11, 12, 13, 18, 19, 20])$
$i(col(black\_hole\_finder, uk\_hpcx), [5, 6, 7, 14, 15, 21, 22, 23])$
$i(col(black\_hole\_finder, barcelona\_sc), [8, 9, 16, 17, 24])$
$i(col(visualizer, ncsa), [1, 2, \ldots, 25])$

For instance, the Barcelona supercomputer is rarely successful:

$$i(col(black\_hole\_finder, barcelona\_sc) \wedge outcome(good)) = \{16\}$$

not because it is a worse supercomputer than UCSD-SDSC or UK-HPCX, but because its connection to the available databases is limited.

Initially, the database is empty, and the broker selects agents at random. As more data is collected, a threshold is reached, at which point the matchmaker begins to use the probabilities.

4.1. Algorithms

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

Figure 5: MATCHMAKE-JOINT
Extract the roles required in the protocol $\mathcal{P}$. Compute the joint distribution for all agent permutations for these roles. Select the agent set with greatest likelihood of success.

In logic terms, MATCHMAKE-JOINT for our Grid example looks like this:

$$\begin{array}{l} extendcollaborators(\mathcal{P}, \mathcal{C}, R) = \\ \quad \{col(astronomy\_database, AD) \wedge \\ \qquad col(black\_hole\_finder, BHS) \wedge \\ \qquad col(visualizer, V)\} \\ if \\ P = p(col(astronomy\_database, AD) \wedge \\ \quad col(black\_hole\_finder, BHS) \wedge \\ \quad col(visualizer, V) \wedge outcome(good) \mid \\ \quad protocol(\text{BLACK\_HOLE\_SEARCH})) \\ and\ P\ is\ maximized \end{array}$$

The second approach, MATCHMAKE-INCREMENTAL, is to select only one agent at a time, as required by the executing protocol. This is done by the matchmaker on demand. The various agents already engaged in the protocol, on needing to send a message to an as-yet-identified agent, will ask the broker to find an agent to fulfil the role at hand.

Figure 6: MATCHMAKE-INCREMENTAL
Compute probability of successful outcome for each agent available for role R given $\mathcal{C}$, the collaborators chosen so far. Select most successful agent.

To illustrate MATCHMAKE-INCREMENTAL, imagine the workflow scenario. At first, Astrid must ask the matchmaker to fill the *black_hole_finder* role. The BHS agent'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.

$$\begin{array}{l} P = p(col(astronomy\_database, AD) \wedge outcome(good) \mid \\ \quad protocol(\text{BLACK\_HOLE\_SEARCH}) \wedge col(black\_hole\_finder, BHF)) \end{array}$$

The final method, MATCHMAKE-TREE is a mix of the first two. Like MATCHMAKE-JOINT, it runs only once, before the protocol executes. Like MATCHMAKE-INCREMENTAL, it selects only one agent at a time (that is, when a message is sent). This seeming paradox is resolved by considering that MATCHMAKE-TREE walks through the protocol, exploring each possible branch, and selecting an agent when necessary in the same manner as MATCHMAKE-INCREMENTAL. This tree of possible choices can be stored with the protocol that is sent to the client, and consulted as required.

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

4.2. Discussion

Having described the three algorithms, we must decided which to use, and when. MATCHMAKE-JOINT is preferable when one wishes to avoid multiple calls to the matchmaker, either because of privacy concerns, or for reasons of communication efficiency.

Figure 3: Astronomy workflow scenario with LCC dialogue framework

We take a hypothetical Grid workflow for our example scenario. We name this protocol BLACK_HOLE_SEARCH. Astrid, our $astronomer$, is attempting to find and visualise a suspected black hole in a region of space around Cygnus-X1. The voluminous data about this segment of space is kept in the very large file *cygnus_x1*, which is stored at numerous repositories, all of which can fill the role *astronomy_database*. She uses a computational intensive service called *black_hole_finder* to actually determine if there is a black hole present. The *black_hole_finder*, if successful, will send the data (now refined and significantly smaller) to a visualisation service, which will pass the final image to Astrid. The variables AD, BHF, and V represent the systems providing the services. Each of these will be selected by the matchmaker when the protocol is executed.

The conceit on which this example hangs is that network bandwidth between various pairs of *black_hole_finder* and *astronomy_database* will be different, largely unknown to the persons providing the individual services, and hence *not declared to the matchmaker*. Since the file *cygnus_x1* is particularly large, this network bandwidth is likely to be a strong determiner of the satisfaction of Astrid.

$$
\begin{array}{ll}
a(astronomer(File), Astronomer) :: & search(File) \Rightarrow a(black\_hole\_finder, BHF)\ then \\
& \left(\begin{array}{l} success \Leftarrow a(black\_hole\_finder, BHF)\ then \\ receive\_visualisation(Thing, V) \leftarrow visualising(Thing) \Leftarrow a(visualizer, V) \end{array}\right) \\
& or \\
& failed \Leftarrow a(black\_hole\_finder, BHF) \\
\\
a(black\_hole\_finder, BHF) :: & search(File) \Leftarrow a(astronomer(File), Astronomer)\ then \\
& grid\_ftp\_get(File) \Rightarrow a(astronomy\_database, AD)\ then \\
& \left(\begin{array}{l} grid\_ftp\_sent(File) \Leftarrow a(astronomy\_database, AD)\ then \\ success \Rightarrow a(astronomer, Astronomer) \\ \qquad\qquad \leftarrow black\_\ hole\_present(File, Black\_\ hole)\ then \\ visualize(Black\_hole, Astronomer) \Rightarrow a(visualizer, V) \end{array}\right) \\
& or \\
& failed \Rightarrow a(astronomer(File), Astronomer) \\
\\
a(astronomy\_database, AD) :: & grid\_ftp\_get(File) \Leftarrow a(black\_hole\_finder, BHF) \\
& grid\_ftp\_sent(File) \Rightarrow a(black\_hole\_finder, BHF) \leftarrow grid\_\ ftp\_\ completed(File, AD) \\
\\
a(visualizer, V) :: & visualize(Thing, Client) \Leftarrow a(?, Requester)\ then \\
& visualising(Thing) \Rightarrow a(?, Client) \leftarrow serve\_visualisation(Thing, Client)
\end{array}
$$

Note that LCC is being used only to coördinate the interaction: when domain-specific protocols, such as Grid FTP, are available and more appropriate, they are used to perform the heavy lifting.

MATCHMAKE-INCREMENTAL and MATCHMAKE-TREE would probably be 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 agents. Such a protocol, in which many roles are never used, could be viewed as a superclass of a set of more specific protocols: the matchmaker would then be determining the particular type at run-time, and select the optimal set of agents for that subtype.

One cannot determine in general which of the two solutions would provide the optimal selection of agents. MATCHMAKE-JOINT appears to provide the 'optimal' solution, but there are some issues with it. The most immediate is that, unlike MATCHMAKE-INCREMENTAL and MATCHMAKE-TREE, agents can be unfairly black-balled for 'under-performing' in unsuccessful protocols in which they never actively participated. Secondly, we hope to add backtracking to LCC, such that we might undo certain agent selections: this is not possible if all roles are filled at the outset.

4.3. Implementation

The broker is currently implemented in Prolog. Performance, even in a naïve implementation, is adequate for tens of thousands of records, and well within the time frame that would be expected on the Internet.

4.4. Inherent difficulties in the problem

We note here two significant problems that seem to be inescapable issues intrinsic to the problem: trusting clients to report honestly and in a socially 'normal' manner the outcome of protocol executions; and the problems of locating mutually co-operative agents in a large society.

Since individual client agents are responsible for the assigning of success metrics to matchmakings, there is scope for agents with unusual criteria, or downright malicious intentions, to soil the database.

Matchmaking is a social activity: clients wish to communicate with service providers that, by definition, they are unaware of. It is unclear how this aspect of agency will develop, and it will depend in many respects on companies' economic decisions, and the behaviour of individuals as to

Figure 4: Rewrite rules governing matchmaking for an LCC protocol

These rewrite rules constitute an extension to those described in [5]. 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 agent name, e.g. $\{col(astronomy\_database, greenwich), col(black\_hole\_finder, ucsd\_sdsc)\}$)

$$\begin{array}{lll}
A :: B \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} A :: E & \textit{if} & B \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E \\
A_1 \ \textit{or} \ A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E & \textit{if} & \neg closed(A_2) \wedge A_1 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E \\
A_1 \ \textit{or} \ A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E & \textit{if} & \neg closed(A_1) \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E \\
A_1 \ \textit{then} \ A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E \ \textit{then} \ A_2 & \textit{if} & A_1 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E \\
A_1 \ \textit{then} \ A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} A_1 \ \textit{then} \ E & \textit{if} & closed(A_1) \wedge collaborators(A_1) = \mathcal{C}' \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}', O} E \\
A_1 \ \textit{par} \ A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O_1 \cup O_2} E_1 \ \textit{par} \ E_2 & \textit{if} & A_1 \xrightarrow{M_i, M_n, \mathcal{P}, \mathcal{C}, O_1} E_1 \wedge A_2 \xrightarrow{M_n, M_o, \mathcal{P}, \mathcal{C}, O_2} E_2 \\
C \leftarrow M \Leftarrow A \xrightarrow{M_i, M_i \setminus \{M \Leftarrow A\}, \mathcal{P}, \mathcal{C}, \emptyset} c(M \Leftarrow A, \mathcal{C}) & \textit{if} & (M \Leftarrow A) \in M_i \wedge satisfy(C) \\
M \Rightarrow A \leftarrow C \xrightarrow{M_i, M_i, \mathcal{P}, \mathcal{C}, \mathcal{C}', \{M \Rightarrow A\}} c(M \Rightarrow A, \mathcal{C}') & \textit{if} & satisfied(C) \wedge \mathcal{C}' = extendcollaborators(\mathcal{P}, \mathcal{C}, role(A)) \\
null \leftarrow C \xrightarrow{M_i, M_i, \mathcal{P}, \mathcal{C}, \emptyset} c(null, \mathcal{C}) & \textit{if} & satisfied(C) \\
a(R, I) \leftarrow C \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, \emptyset} a(R, I) :: B & \textit{if} & clause(\mathcal{P}, C, a(R, I) :: B) \wedge satisfied(C)
\end{array}$$

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

We can capture our various algorithms for matchmaking using the same rewrite rules: the issue is *when* the set of collaborators is actually decided. $extendcollaborators$ determines the selection of a new agent: it is the matchmaking function. It varies slightly, depending on the exact matchmaking algorithm in use. By using the same rewrite rules regardless of the matchmaking policy, we make it easier to re-use model-checking [12] and other tools on the protocols.

how many agents are deployed. There is a spectrum of possibilities, ranging from areas that are dominated by their 500lb gorillas (Google, Amazon, and E-Bay) through those that have dozens or hundreds of providers (insurers), to millions (personal calendar agents). We suppose that there will be a mix, and would presume that, for most purposes where one would use a matchmaker, we would be dealing with roles that supported numbers toward the lower end of the scale. Further, we must ask how many service types will be provided. Again, in each domain, we might have a simple, monolithic suck-it-and-see interface (Google again), or an interface with such fine granularity that few engineers ever fully understand or exploit it. Here, it is perhaps harder to predict the numbers.

While our technique handles large numbers of incidences, it does not scale for very large numbers of agents 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

$$\prod_{r_i \in R} providers(r_i)$$

This is an insuperable problem when considering large numbers of agents. No matchmaking system could possibly hope to discover all the various permutations of agents, although more sophisticated machine learning techniques might be helpful in finding non-obvious groupings of agents that simply could not be found by trial-and-error. How much of an issue this actually becomes in any particular domain will be heavily influenced by the outcomes to the issues discussed above.

5. Related work

The brokering problem arises in agent systems, semantic web, and grid environments. The matchmaking problem is discussed in [1, 13, 7]. We consciously ignored methods like those found in [14], though they would be crucial in any real-world deployment: we believe our technique would usefully augment such systems, and we intend to fuse the two approaches.

Our problem conception—matchmaking multiple roles for the same dialogue—appears novel in the matchmaking literature. Our use of performance histories is predated by a similar approach found in [8], although that, again, only examines the case of two-party interactions.

Finally, our take on the likelihood of roles being successfully discharged is comparable with some views of trust.

6. Conclusion and future work

We have shown that, in plausible scenarios, the successful completion of a task may depend not only on the advertised abilities of agents but on their collective suitability and inter-operability. We presented a simple, but effective, technique for detecting successful groupings of agents. We highlighted the intractability of the problem in environments with large numbers of available provider agents and/or roles.

Despite talk of disintermediation and peer-to-peer systems, multi-agent scenarios that are amenable to matchmaking seem sparse. Typically, multi-party scenarios, where they do exist, have a client, a central service which does the matchmaking itself—one example is a travel agent service which arranges the various flight, car-hire, hotel bookings etc, without consulting a general purpose matchmaker. This paucity may be due to the emergence only recently of formalisms that can express such interactions easily, or it may reflect a deeper problem with conceiving problems in such a distributed sense, as we assumed here. A standard library of such interactions would be helpful to research in the field.

Currently, only assignments of agents to roles are recorded. The utility of recording other events in the execution of the protocol will be investigated. For instance, 'partially satisfactory' protocol executions could be interrogated to discover which agents are performing well, and which are proving to be a bottle-neck. Providing richer feedback from the client on its satisfaction with the outcome would be useful in itself. This could be extended to treating matchmaking as an interactive process: an agent requesting a service to satisfy a task might have various information that, unbeknownst to it, could aid the matchmaking in ascertaining the best protocol and collaborators to use. We also ignore issues of ontological compatibility and alignment. More sophisticated techniques for inferring compatibility will be examined, including those such as [14].

Previous work on LCC has examined the possibility of backtracking in protocols, in this case, allowing the broker to re-choose providers if an interaction failed. This would be easy for interactions, such as information gathering, where no commitments are made, but would require careful consideration of actions and state in, for example, a purchasing environment.

Finally, although the incidence calculus provides a convenient framework for this model, it may not provide us with an optimal computational process. The use of machine learning techniques will be investigated.

References

[1] 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)

[2] R. G. Smith: The contract net protocol: high-level communication and control in a distributed problem solver. (1988) 357–366

[3] 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 1.1 (2004)

[4] Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web Services Choreography Description Language Version 1.0 (2004) W3C Working Draft 12 October 2004.

[5] 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)

[6] Blythe, J., Deelman, E., Gil, Y.: Planning for workflow construction and maintenance on the Grid. In: ICAPS03 workshop. (2003)

[7] 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

[8] 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

[9] Esteva, M., Rodriguez, J., Arcos, J., Sierra, C., Garcia, P.: Formalising Agent Mediated Electronic Institutions (2000)

[10] Milner, R.: Communication and Concurrency. Prentice Hall (1989)

[11] Bundy, A.: Incidence calculus: A mechanism for probabilistic reasoning. Journal of Automated Reasoning **1** (1985) 263–284

[12] Walton, C.: Model Checking Multi-Agent Web Services. In: Proceedings of the 2004 AAAI Spring Symposium on Semantic Web Services. (2004)

[13] Wong, H., Sycara, K.: A Taxonomy of Middle-agents for the Internet. (2000)

[14] Paulucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic Matching of Web Services Capabilities. In: The Semantic Web — ISWC 2002: Proceedings. (2002)