Accounting for Valency in Service Composition

David Lambert

School of Informatics
University of Edinburgh
d.j.lambert@sms.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 composition is crucial. While current tools offer various tools and methods for selecting services based on various user-defined criteria, little attention has been paid to how such services interact. We believe that semantic coördination or agreement between services will be an important factor in the usability and success of service composition, and that this agreement cannot be guaranteed by semantic description alone. We have developed a simple but apparently effective technique for selecting services based on their record of performance with others.

1 Introduction

Grid and Web services are making possible a new style of programming. In fields ranging from bioinformatics, through natural language processing and on to astrophysics, domain experts can use graphical tools to compose scientific workflows that tie together a range of services in order to solve problems. Current tools aim at helping the user in the process of discovering appropriate services, constructing the workflow, and monitoring the execution. What is missing from this picture is information about how well the services actually act and, particularly, interact. Services are treated as fungible black boxes, when there is good reason to believe they are not. We present a technique which attempts to select an optimal set of services, in a manner that is transparent to the user.

Middle-agents [1] connect clients with providers, based on matching capability advertisements from providers with requests for service from clients. One of the most common types of such middle-agents is the matchmaker, on which we focus. Most matchmaking research has concentrated on sophisticated mechanisms for describing services and requirements, such as capability description languages [2]. It is unlikely that many services will fully match their specification, and so not all services will coöperate well with each other, even when they claim to. Mutual misunderstandings will occur, cliques will form. Why might this happen, and what can we do to overcome this obstacle to widespread use of multiple-service interactions in real-world operation?

2 Motivation

Previous research has produced powerful languages for describing service interfaces and semantics. Why do we believe this is insufficient? Regardless of how carefully service providers specify the behaviour of their systems, there will remain, for any non-trivial service, some level of 'semantic slack' such that matchmaking based on purely semantic service description can be improved by statistical or other learning techniques. Below, we examine several reasons for this encoding gap. Only the first is a technical problem: in the remainder, additional information is absented for other reasons.

Semantic descriptions of service suffer the same problems as any other logic-based knowledge representation:

1. **unknown theory** we might not understand what causes inter-relationships to work or fail
2. **unknown facts** even if we had a theory, we might not have adequate empirical data about the domain
3. **laziness** it would be impracticable to record all this information in the semantic description

The empirical data problem is hard to solve for individual service providers or users: it is best done by some middle-agent. But what explanations have we for the lack of a theory? Some possible reasons are: social concerns, such as different communities of practice clustering around particular service providers just because their friends are there; business partnerships, such as an airline having a deal with a particular car-hire company; organisations that seem to have nothing in common may well be using software created by a single group, causing them to inter-operate better within their clique than with outsiders; particular resources or constraints, such as network bandwidth, shared between providers (we use this as a running example). In the real world, particularly when money is involved, some services are likely to be misanthropic in some sense: some service providers will not want to work with certain others, while others might deliberately conceal information.

'Laziness' might be genuine laziness: service providers not bothering to provide a complete description of their offering. But there are other causes: the user being unaware of the language's ability to express a constraint, or of the effect of declaring the constraint; user expectation that the information will not be used by clients or matchmakers; comprehensive constraints might be too expensive to generate or use.

3 Lightweight coördination calculus

We use a language called the Lightweight Coördination Calculus (LCC), based on the Calculus of Communicating Systems (CCS) [3], to specify our service interactions. LCC provides a simple language featuring message passing (denoted $\Rightarrow$ for sending, and $\Leftarrow$ for receiving) with the operators *then* (sequence), *or*

(choice), *par* (parallel execution), 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 1), the roles (or types) 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.

4 Incidence calculus

To store this data, and to calculate the probabilities, we use the incidence calculus [4]. 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. 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.

5 The LCC matchmaker

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

When a dialogue is in progress and a message needs to be sent to role for which an service has not yet been selected, the matchmaker is engaged. It finds a service that maximises the probability of a successful outcome given the current protocol type

and role instantiations. Over time, the matchmaker builds a database of the various role/service instantiations, and the clients' satisfaction with the outcomes.

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.

Fig. 1. Astronomy workflow scenario with LCC dialogue framework

We take a hypothetical Grid workflow for our example scenario, called BLACK_HOLE_SEARCH. Astrid, our *astronomer*, wants 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 computationally 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*.

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

Fig. 2. 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 service name, e.g. $col(black\_hole\_finder, ucsd\_sdsc)\}$. The same rewrite rules hold regardless of the implementation of the matchmaking function *extendcollaborators*. This enables us to apply other LCC tools, such as model-checkers and the interpreter itself, without alteration while allowing us to change *extendcollaborators*, 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}' \\
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 \\
 & \quad 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 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}' = 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) \\
 & \quad \wedge satisfied(C)
\end{array}$$

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

5.1 Algorithms

We have developed three algorithms for choosing services, although 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 services in their respective roles, selecting the grouping with the largest probability of a good outcome.

The second approach, MATCHMAKE-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 broker to find an service to fulfil the role at hand. MATCHMAKE-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 MATCHMAKE-INCREMENTAL, imagine the workflow scenario. At first, Astrid 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*.

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 service 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 service 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 services 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 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.

5.2 Discussion

Having described the three algorithms, we must consider 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.

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 services. 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 services for that subtype.

One cannot determine in general which of the two solutions would provide the optimal selection of services. 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, services 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 service selections: this is not possible if all roles are filled at the outset.

5.3 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 services in a large society.

Since individual client services are responsible for the assigning of success metrics to matchmakings, there is scope for services with unusual criteria, or downright malicious intentions, to corrupt 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 how many services are deployed. Some areas will be dominated by 500lb gorillas (Google, Amazon, E-Bay), for others (personal calendar agents) millions will exist, and some will occupy a middle ground of dozens or hundreds of providers (insurers). We 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 interface, 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 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 brokering problem arises in agent systems, semantic web, and grid environments. The matchmaking problem is discussed in [1, 6, 7]. We consciously ignored methods like those found in [8], 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—is anticipated by the SELF-SERV system [9], 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 (i.e. specification of service parameters) from the client. Our use of performance histories is predated by a similar approach found in [10], although that, again, only examines the case of two-party interactions.

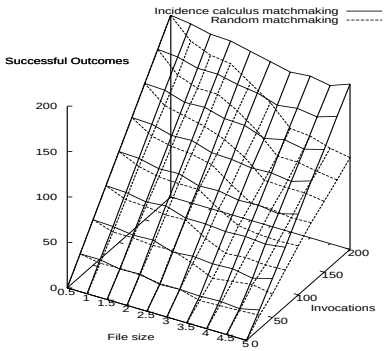
7 Conclusion and future work

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

We have simulated several scenarios, including the astronomy Grid example. Initial results are encouraging: figure 3 shows the improvement in the outcome of service invocations as the matchmaker acquires experience of the various services' interactions, as compared with a random matchmaking strategy. However, the gain is highly sensitive to scenario and the particular services available.

We are working to apply this technique to real-world web services in the bioinformatics domain, where we expect to find the kinds of inter-service mismatches we hypothesise here. We intend to examine other machine learning techniques, and also

Fig. 3. Service selection improves as matchmaking database grows



look at dynamically synthesising LCC protocols using the same principles presented in this paper.

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. Wickler, G., Tate, A.: Capability Representations for Brokering: A Survey (1999)
3. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
4. Bundy, A.: Incidence calculus: A mechanism for probabilistic reasoning. Journal of Automated Reasoning **1** (1985) 263–284
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. Wong, H., Sycara, K.: A Taxonomy of Middle-agents for the Internet. (2000)
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. Paulucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic Matching of Web Services Capabilities. In: The Semantic Web — ISWC 2002: Proceedings. (2002)
9. Liangzhao Zeng and Boualem Benatallah and Marlon Dumas and Jayant Kalagnanam and Quan Z. Sheng: 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
10. 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