Thinking Outside the TBox

Multiparty Service Matchmaking as Information Retrieval

David J. Lambert

School of Informatics

University of Edinburgh

Doctor of Philosophy

Centre for Intelligent Systems and their Applications

School of Informatics

University of Edinburgh

2009

Abstract

Service oriented computing is crucial to a large and growing number of computational undertakings. Central to its approach are the open and network-accessible services provided by many different organisations, and which in turn enable the easy creation of composite workflows. This leads to an environment containing many thousands of services, in which a programmer or automated composition system must discover and select services appropriate for the task at hand. This discovery and selection process is known as matchmaking.

Prior work in the field has conceived the problem as one of sufficiently describing individual services using formal, symbolic knowledge representation languages. We review the prior work, and present arguments for why it is optimistic to assume that this approach will be adequate by itself. With these issues in mind, we examine how, by reformulating the task and giving the matchmaker a record of prior service performance, we can alleviate some of the problems. Using two formalisms—the incidence calculus and the lightweight coordination calculus—along with algorithms inspired by information retrieval techniques, we evolve a series of simple matchmaking agents that learn from experience how to select those services which performed well in the past, while making minimal demands on the service users. We extend this mechanism to the overlooked case of matchmaking in workflows using multiple services, selecting groups of services known to inter-operate well. We examine the performance of such matchmakers in possible future services environments, and discuss issues in applying such techniques in large-scale deployments.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*David J. Lambert*

School of Informatics

University of Edinburgh)

Acknowledgements

Thanks are due to my supervisor Professor David Robertson for his patient guidance, and to my second supervisor Dr Stephen Potter, who got me interested in the subject of brokering and matchmaking by telling me not to touch it.

This research was sponsored by the Advanced Knowledge Technologies (AKT) project, a six-year collaboration between groups at the universities of Aberdeen, Edinburgh, Sheffield, Southampton, and the Open University. AKT was funded by the United Kingdom's Engineering and Physical Sciences Research Council.

PhD dissertation acknowledgements usually conclude with a long list of friends who nursed the patient through their illness. In my case that list is very nearly, but not quite, $\prime()$. To the members of that list (you know who you are): thank you.

Table of Contents

List of Figures

Chapter 1

Introduction

Once upon a time, computer technology was described in terms of generations. Towards the end of that phase, popular opinion held that the fifth generation would be endowed with artificial intelligence through logical inference (Moto-oka and Kitsuregawa 1985; Feigenbaum and McCorduck 1983). Those machines never materialised, and people tired of counting. The logico-deductive, closed-world view gave way to the more open and messy one of the Internet. It is fair to call the real fifth generation the networked computer, and we could now speak of a sixth generation where computation itself is migrating into the network. A key element of this networked computation is the pervasive use of services. An essential element of service oriented architectures (SOA) (Singh and Huhns 2005) is the mechanism for connecting agents requiring services with those able to provide them. When automated, this task is known as 'matchmaking', and approaches to it remain rooted in the fifth generation's exclusively logical Weltanshauung. The lesson of the Web has been that openness and some sloppiness leads to scalability, and that the resulting size in turn can enable the exploitation of emergent structure and statistical wisdom. In this thesis, we re-examine matchmaking through the lens of the open, messy Web, and investigate the utility of an information-retrieval inspired approach to selecting appropriate services.

1.1 Services oriented computing

Distributed computing is the practice of constructing and operating systems of logically connected programs on physically distinct computer systems and having them communicate through networking hardware. Such systems can range from parallel computers running thousands of identical processing nodes in a single machine room to wildly heterogeneous ones like those found on the Web. The full promise of ubiquitous, distributed computation can only be realised if the various components can be mustered as a coherent system in a flexible and inexpensive way. The creation of such a framework has long been an aim of software engineering research, and the many attempts have appeared in various guises, from distributed computing standards like CORBA (Object Management Group 2004), DCOM (Microsoft Corporation 1996) and Jini (Arnold 2000), to multi-agent systems (Wooldridge and Ciancarini 2000). While such systems have met with some success, they have failed to achieve a critical mass, principally due to their complexity and a lack of openness.

In the last few years, 'service oriented computing' (Papazoglou 2003), implemented as 'web services' (Booth et al. 2004), has attracted sufficient attention for it to become an established part of the computing landscape. Today, there are many such services available, and increasing pressure to coordinate them in adaptable ways. Service-oriented computing is already a cornerstone technology for e-Science, business, government, and other large organisations. Services are software entities—possibly fronting for real-world end-effectors—which are accessible across the Internet via standard protocols. Services obviate the difficulties of distributing, installing, and ensuring currency of software that must otherwise be deployed on users' machines. And, because they can expose functionality across organisational boundaries, they enable easier establishment of contracts and the construction of virtual organisations constructed between distinct organisations.

The principle reason for this sudden surge in use of services is the success of the

HyperText Transfer Protocol (HTTP) (Fielding et al. 1999) which underlies the Web, and XML (Bray et al. 2008). Using HTTP to carry service requests and results instead of web browser data, engineers have found a light-weight, widely understood and ubiquitously deployed platform for distributed computing. XML, being just text, is easy to inspect and manipulate, removing the need for complex tool sets and making cross-platform and inter-language operation more easily achievable by working programmers. Because of this, web services (Gottschalk et al. 2002) offer a compelling vehicle for distributing software functionality, providing a common platform for replacing traditional remote procedure call approaches. They enable access to distributed resources like bioinformatics databases and algorithms (Wilkinson and Links 2002), virtual observatories (Walton, Lawrence, and Linde 2003), and web-based storage facilities (Palankar et al. 2008). This success has made web services an object of interest to three other major research threads: the Grid, the semantic web, and multi-agent systems. The resulting convergence of these fields impacts on a key aspect of service oriented computing: finding the right services.

1.2 Finding services

To use services one must be able to find them (both the Platonic existence of abstract, task-achieving processes, and the actual, invokable implementations), reason about how to compose them to solve tasks, and invoke them. And this must be done in the web world: huge scale in terms of number of providers and clients requesting access; variety of services that will be offered; the inevitable sloppiness of implementation and partial adherence to standards; and variability in intrinsic quality of individual services.

Today, when programmers manually construct systems using the service-oriented paradigm, they often simply use the services they have been told to use by an employer or colleague. If they have to discover a new service themselves, they typically do so by browsing or searching the human-readable web. In particular domains, there may be web

sites which record useful services, such as the simple web page in bioinformatics[1]. There are also web services search engines like startup company Seekda[2]. However the services are found, their integration still proceeds in the traditional mold of software development: crafting single-provider/single-user linkages using traditional programming languages. Most of these software interactions are hard-coded: programs interact with other software systems at predetermined network addresses, using predetermined sequences of calls.

Expectations in academia and industry are that the number of web services will grow into the hundreds of thousands, and possibly into billions. At the same time, ubiquitous services providing for every need will draw in many non-programmers—beginning with scientists—who will construct distributed computations using work-flow oriented graphical programming suites or automated compositions derived from formal specifications. A common assumption behind much research in middleware and semantic web services is that manual service selection will not suffice. In any case, it would be better to use more sophisticated middleware to at least aid humans in the process (Bernstein 1996).

Tools like Taverna (Oinn et al. 2004) make it possible for users to construct workflows without programming, and to select the services which enact them. However, the available services are generally either hard-coded into the tools, or manually acquired from web pages or UDDI registries. The invoker of a service must have some way of identifying that service. It is usually undesirable, and often not possible, for the various systems involved to know of each other's existence before the need for interaction occurs.

Service selection can often be done automatically, in a process called 'matchmaking'. The problem has been addressed in the multi-agent community, and before that in the distributed computing field where the challenge of service discovery was first identified as the 'connection problem' (Smith 1980). The solution taken by most of these has

[1] `http://www.ebi.ac.uk/Tools/webservices/`
[2] `http://seekda.com/`

been the introduction of 'middle-agents' (Decker, Sycara, and Williamson 1997) which provide a meeting point for service providers and clients. Middle agents can also solve related problems like load-balancing, and the dynamism of agents joining and leaving communities.

Most of this matchmaking research happened in the multi-agent field (Martin, Cheyer, and Moran 1999; Sycara, Klusch, and Widoff 1999; Kuokka and Harada 1995; Singh 1993) and continued into semantic web services (Paolucci et al. 2002*b*; Li and Horrocks 2003). The general procedure for these matchmakers is always the following: any agent offering a service lodges an advertisement of it with a matchmaker. The advert takes the shape of a formal description, in a 'capability description language'. In the past, this was often simply a set of keywords. Recently, it has become popular to use description logics (Nardi and Brachman 2003)—a core technology of the semantic web—to classify the service. Clients requesting a service similarly generate a description of their preferred service, and submit it to the matchmaker. The matchmaker compares the requested service description against its advertisement database, and selects the most appropriate, which it then names to the client.

These matchmaking schemes have worked well in laboratory settings where the services are relatively few in number and designed by small, homogeneous teams, but there is a real question of whether their purist logical nature is appropriate for the much larger, more open and less predictable web service environments we face now.

1.3 The modern service ecology

Matchmaking has traditionally focused on agents in homogeneous, in-lab scenarios. The environment of service oriented computing is considerably more hostile: there are more services from more providers, the descriptions will be apt to be less accurate, and the results less rigorously checked. In this section, we look at three areas of the the service oriented computing environment, and what they mean for matchmaking. We cover the

pragmatics of web services, the semantic web, and the impact of search engines.

1.3.1 The pragmatics of web services

Multi-agent systems research assumes a model of agency in which individual agents reason in a very rich way about how to achieve goals in a social setting, with insight into their own beliefs desires and intentions (BDI), and those of others (Wooldridge 2000). They create plans and request behaviours of other agents in order to achieve their goals. Doing this is hard, and still a research exercise, rather than a software engineering approach ready for large-scale deployment. In contrast, the success of service oriented architectures is largely attributable to their more pragmatic, less technically ambitious approach. Web services piggy-back on the standardised, widely deployed and understood standards of URLs, the HTTP protocol, and XML. Instead of automated planning, humans can design fixed workflows for execution (Andrews et al. 2003; Oinn et al. 2004). Web services can be easily glued together with simple programming tools, without retraining engineers in BDI models or artificial intelligence.

Agency presents a large jump in complexity from current technology, so there is a high barrier to entry. Web services provide a low-cost way to enter, by using a large number of existing and familiar technologies, adding distributed computing in a straight-forward way, and permitting layering of more expressive methods on top. Web services can provide the low-level infrastructure for communication, message passing, security and so on. Agents may be layered on top by some actors to provide flexibility, planning, or intelligent error recovery and compensation, but their presence will probably be hidden behind a normal web-services front (Payne 2008; Foster, Jennings, and Kesselman 2004).

It is this openness and low entry barrier that introduces perhaps the biggest change to the way matchmaking might be done. When systems were closed, those building them could be sure of the circumstances of their use, and those using them could be well trained or simply prevented from accessing them. In contrast, the natural consequence

of openness is heterogeneity and loss of control.

The situation mirrors the early Web. Although technically unsophisticated compared to other hypertext systems of the time, it has nevertheless revolutionised the way we deal with information. The key to the web seems to have been the low technical and social barriers to entry. Writing HTML is easy, and although writing correct HTML is more difficult than it might be, browsers and other technologies are forgiving by design and necessity. Tim Berners-Lee's injunction to "be liberal in what you require but conservative in what you do" lies behind the robustness and popularity of the web. The web did not, for example, enforce link integrity, which allowed conventional tools ignorant of links to be used. And anyone could create a link, since they did not need access to some centralised link database (Berners-Lee 1992). Such openness has led to problems beyond the technical domain, in terms of law and governance: spam, phishing and malware all cause users great inconvenience, but it is precisely this openness which has created the Web we value.

1.3.2 The semantic web

The Semantic Web augments the current Web with formal knowledge representation mechanisms, so that machines can understand and reason with more of the knowledge it contains. There are already several popular applications of semantic web technology, including 'friend-of-a-friend' (FOAF) (Brickley and Miller 2007) for describing social networks, and RDF Site Summary (RSS) (Beged-Dov et al. 2000) used for syndicating news and blog feeds. But the semantic web vision (Berners-Lee, Hendler, and Lassila 2001) clearly places services front and centre:

> . . . an agent coming to the clinic's Web page will know not just that the page has keywords such as 'treatment, medicine, physical, therapy' (as might be encoded today) but also that Dr. Hartman works at this clinic on Mondays, Wednesdays and Fridays and that the script takes a date range in yyyy-mm-dd format and returns appointment times. And it will 'know' all this without needing artificial intelligence on the scale of 2001's Hal or Star Wars's C-3PO. Instead these semantics were encoded into the Web page when the clinic's office manager (who never took Comp Sci 101) massaged

> it into shape using off-the-shelf software for writing Semantic Web pages
> ...

The effort to make available semantic descriptions of web services is known as 'semantic web services' (Burstein et al. 2005; McIlraith, Son, and Zeng 2001). The hope is that by describing formally the kind of task a service performs, its inputs and outputs, message formats, and its possible interactions with other services, it will be possible to automate to a large extent the discovery, selection, composition, execution and monitoring of the large-scale systems possible with distributed computing. There are several major frameworks, including OWL-S (Martin et al. 2004), WSMO (Lausen, Polleres, and Roman 2005), and SWSF (Battle et al. 2005). However, there has been no take-up so far by users.

Although the semantic web seems to promise a new golden age for formal knowledge representation and multi-agent systems, pragmatism is again a central theme. James Hendler's slogan that 'a little semantics goes a long way' is widely repeated, and the most successful deployment of RDF has been re-branded more prosaically as 'linked data' (Bizer, Heath, and Berners-Lee 2008). Heavy-weight semantics—careful engineered ontologies with rich constraints—are not only difficult to provide, they are difficult to reason with. Thus, if semantic web services are accepted, it may be in a lightweight form like WSMO-Lite (Vitvar et al. 2008) and SA-WSDL (Farrell and Lausen 2007), where the limited expressiveness will lead to problems with integration.

Much of the scepticism (Shirky 2003) of the semantic web comes from a misunderstanding that it ever intended to provide a single, global ontology. Instead, many different ontologies will be knitted together as needed. There will not even be a single standard for expressing those ontologies: OWL, the primary language for ontologies on the Web, already exists in three 'species', and the major semantic web services efforts (WSMO and SWSF) have defined their own, more expressive languages (Bruijn et al. 2005; Bernstein et al. 2005). Another Hendler coinage, 'semantic webs', illustrates the growing acceptance that there will be many ontologies, and ontology languages, used

in various corners of the Internet. Ontology mapping (Kalfoglou and Schorlemmer 2003) is imperfect, and mapping from more to less expressive formalisms is, in general, impossible. This heterogeneity will make it necessary to consider how services are chosen as a group, because the interactions of the agents will be as important as their individual performance.

1.3.3 Search engines

Web search engines are now such an essential element of the Internet experience that it is hard to remember how recent an arrival they are. Alta Vista, the first such engine to provide a full-text index of a large part of the web, opened to the public in late 1995. Prior to this, two methods were used to find relevant pages: subject-based hierarchies of links; and the use of author-chosen keywords embedded in the pages themselves.

In the early days of the world wide web, page authors would embed an HTML construction like

```
<meta name="keywords" content="matchmaking multi-agent">
```

in their page to enable search engines to index it. These keywords were often chosen ineptly due to obvious ambiguities or lack of precision, or malevolently so as to increase a page's visibility unjustly, and the technique has fallen into disuse. The other early navigational tool was the hierarchical, ontology-like directory, such as Open Directory[3] or Yahoo!—the name itself an acronym of 'Yet Another Hierarchical Officious Oracle'—and these have now all but died out.

They have given way to search engines using vector-space based information retrieval methods, augmented by studying the interconnections between the billions of pages. Such analysis uncovers the statistical patterns which show the more influential, and prized pages, in the eyes not of those who create them, but those who use and therefore link to them (Brin and Page 1998). We could say that modern search engines make the community the arbiter of the semantics of a page, not the author. In a similar vein,

[3] `http://www.dmoz.org/`

recommender systems like those at Amazon can pool the collected preferences of the community to highlight the better offerings (Resnick and Varian 1997; Schafer, Konstan, and Riedl 1999).

1.4 Thesis

How might these themes from the Web and web services reshape our view of matchmaking? To recap, web services, like the Web itself, are open to anyone, and thus less likely to implement standards fully and correctly. With thousands of services, there are bound to be the usual problems of bugs and incompatibilities. Since money is involved, many services will be described less than honestly. Lightweight semantics for services may well take off, but provide insufficient information for good matchmaking. The interactions between agents become important, due in part to agents using different ontologies or logics behind the scenes.

Just as search engines with their statistical approach have largely superseded carefully edited topic-based directories like the original Yahoo! or OpenDirectory, we hypothesise that a similar approach might help in service selection. There are certainly parallels. The hierarchical directory approach of the early Web is reminiscent of the use of description logics to classify services. The use of in-page keywords is similar to the use of service advertisements, and faces a similar problem of accuracy and honesty. In search engines, pages are described by keywords, where services are described by concepts, and registered and found accordingly. But pages also link to and from other pages. Services can be said to 'link' when they collaborate successfully in a interaction initiated by a client, and the linkages can be exploited to better select groups of services which interact better with each other than with a random grouping. By considering the services as keywords, and the interactions as documents, we can apply information retrieval techniques to find the correlations between successful services.

At present, a matchmaker gets no feedback about the quality of its service selection.

If, instead, each client registered with the matchmaker their satisfaction with the end result of a match-made interaction, the matcher can, over time, construct a database of which agents work best. Moreover, traditional matchmaking focuses on selecting single agents at a time. With workflows connecting many services, we should be able to consider the effects of interactions between services, and select groups of agents which work best with each other.

We use two key tools in our investigation. The Lightweight Coordination Calculus (LCC) (Robertson 2004) is a simple, logic-programming style workflow language. It has been used as to implement BPEL workflows in a peer-to-peer manner (Guo, Robertson, and Chen-Burger 2005), for supporting model-checking based verification of deontic constraints in agent dialogues (Osman, Robertson, and Walton 2005), and for studying on-demand ontology matching (Besana and Robertson 2007). LCC provides a simple means for handling multi-service compositions, which are central to our argument about the centrality of service interactions to matchmaking. The second tool is the Incidence Calculus (IC) (Bundy 1985). Incidence calculus provides a clean mathematical model for dealing with the probabilities that emerge when we consider the sets of interactions. Using these, we construct several proof-of-concept matchmakers which learn from experience to find those groupings of agents which work not only work well individually, but as collaborators.

The remainder of this dissertation is structured as follows: chapter 2 outlines the problem of service selection in distributed systems, and surveys approaches from other fields that parallel our own, while chapter 3 examines the previous research on service matchmaking, covering both architectures and the matchmakers designed to operate within them. Chapter 4 examines the problems that we feel have been glossed over by mainstream matchmaking research, and justifies the addition of probabilistic techniques to matchmaking. Chapter 5 reviews the Lightweight Coordination Calculus and Incidence Calculus. In chapter 6 we build a matchmaker for simple client-server interactions, where the historical data about past interactions is used to improve service

selection, and in chapter 7 we extend the approach to support multi-party matchmaking, providing experimental results demonstrating its efficacy. Chapter 8 looks at how the approach would scale in a moderately large scenario, using simulations based on plausible service interaction properties. Chapter 9 concludes by reviewing the contributions, and looking to possible future directions for the work.

1.5 Publications

Several publications have resulted from the work presented in this thesis:

- Key ideas from chapters 4, 6, and 7 were first published in *Accounting for Valency in Service Composition* (Lambert 2005) at the first Advanced Knowledge Technologies doctoral symposium.
- A refinement of that paper appeared as *Matchmaking and Brokering Multi-Party Interactions Using Historical Performance Data* (Lambert and Robertson 2005) at the fourth international joint conference on autonomous agents and multi-agent systems (AAMAS 2005).
- The work on selecting roles in chapter 7 was published in the paper *Selecting Web Services Statistically* (Lambert and Robertson 2006) at the tenth international workshop on cooperative information agents (CIA 2006). This was later reprinted in a collection of selected papers from the Advanced Knowledge Technologies project (Advanced Knowledge Technologies 2007).
- This approach to matchmaking formed part of a compendium paper of work on the Lightweight Coordination Calculus, *Models of Interaction as a Grounding for Peer to Peer Knowledge Sharing* (Robertson et al. 2009) in (Dillon et al. 2009).

Chapter 2

Background

In this chapter, we cover the background of matchmaking. We first examine the emergence, more than once, of distributed computation (section 2.1), and the 'connection problem' it poses (section 2.2). We examine the requirements for solving the connection problem (section 2.3), and in particular the conceptual approaches to describing service capabilities (section 2.4). We defer detailed examination of particular matchmaking frameworks and matchmakers until chapter 3.

2.1 Distributed computing

Distributed computing is the use of multiple physical computers to perform a task. The degree of physical separation between the computing nodes may be minimal in the case of parallel supercomputers or computer clusters, or globe-spanning, as in the SETI@home project. Likewise, the tasks engaged in by each node may be homogeneous, which each node running an identical algorithm on a parcel of data, or heterogeneous, where a node may offer a globally unique service.

Distributed computing has a long history, with its most famous early example being the development of the ARPAnet (Licklider 1963; Licklider and Taylor 1968). The ARPAnet's raison d'etre was providing cross-continent sharing of precious computing resources, which it achieved through a novel packet-switched network and applications

including file transfer, electronic mail, and remote access to machines. These services were tied to specific application protocols and the programs that implemented them, making it difficult to integrate functionality for new uses. This lead to the idea of generic protocols, based on the notion of remote procedure calls (RPC), and the first explicit mention of RPC seems to be in the Internet Engineering Task Force's 'Request For Comment' 707 (White 1975). RPC707 highlighted the problems inherent with the early ARPA¡net protocols which were designed for interactive use by humans. These protocols frustrated the programmatic use of remote computers: more effective resource sharing depended on making programmatic remote use easier, and this could be achieved by creating a general abstraction at the level of function invocation, so that individual applications did not need to implement, document and publicise a new protocol for each new application. Instead, programmers could simply offer an application programming interface (API) that could be trivially invoked with the same toolset. It could be hoped that the availability of such a generic interface would spur the provision of such functionality through it, since application developers would be encouraged to build their programs so that they could be invoked by RPC. Two other insights from White were that RPC protocols "permit the server process to invoke commands in the user process, that is, eliminate entirely the often inappropriate user/server distinction"; and the statement of agency's central notion of autonomy, making clear that a server was under no obligation to fulfil a request if it were unwilling to do so. By the 1980s, two main streams of distributed computing were apparent: systems based on remote procedure call, which were then developing into distributed object-orientation, and distributed AI, which itself morphed into the field of multi-agent systems.

RPC systems, so called because they resembled the familiar and simple abstraction of the procedure call, achieved widespread adoption. The Open Network Computing Remote Procedure Call (ONC/RPC) (Srinivasan 1995) still underpins modern systems like Sun's Network File System. Just as fashionable object orientation was replacing the procedural programming model on the desktop, distributed object mechanisms

were a natural development for the network. The principle standards in this field were the Common Object Request Broker Architecture (much better known as CORBA), and Distributed Component Object Model (DCOM). CORBA was a cross-platform, multi-vendor standard driven by the Object Management Group, while DCOM was a proprietary one, tied to the Microsoft Windows platform. Both schemes were object-oriented, and made object method invocations, from the programmer's perspective, independent of the object's location, whether it lay in the same application, in a logically separate address space on the same machine, or most profoundly, on another machine. An 'interface definition language' (IDL) is used to specify the object's structure and methods, and which is then compiled to language specific bindings, called stubs (for the client) and skeletons (in the provider). Both CORBA and DCOM saw significant use, but were not universally accepted: they were complex, and CORBA suffered problems of incompatibility between different vendors' tools.

Distributed AI (DAI) investigated the use of distributed computation to solve AI problems, principally in the sense of managing parallel computing. DAI's parallel problem solving model is now commonplace: systems like SETI@home[1] and BOINC[2] allow anyone to participate in searching for extra terrestrial intelligence, climate modelling, or other projects, by running a client program on their machine. Multi-agent systems, in contrast, emphasise the autonomy of agents, and their ownership and fealty to different actors with diverse goals, and a multiplicity of tasks and abilities. Because agents are goal-driven, they have options in which services to invoke to achieve a goal, and could only request other agents to perform actions, rather than demand them, as in RPC.

The greater dynamicity, sociability, and intelligence of such agents created a buzz about software agents that went unrealised in real applications. In the last few years, attention has switched back from the agency approach to the less sophisticated approaches of RPC, this time realised as web services and the Grid, themselves now

[1] `http://setiathome.ssl.berkeley.edu/`
[2] `http://boinc.berkeley.edu/`

converging on underlying technologies. No attempt is made in these systems to model beliefs, or negotiate. The format of the messages passed between client and service is strictly defined. Neither clients nor servers need any intelligence to interpret the messages or reason about the consequences of fulfilling requests, since the thinking was done by the programmers when they wrote the service. This simplification has meant, however, that open, distributed computation services are now becoming commonplace, with both web services and the Grid having achieved a critical mass of implementers and users. Agency researchers can now piggy-back on some of these services, rather than construct their own. And the notion of 'semantic web services' has now appeared as a research topic, occupying a space somewhere between dumb RPC and intelligent multi-agent systems.

2.2 The connection problem

Distributed computation is concerned with the decomposition of problems, and the subsequent distribution and coordination of the sub-problems. Subsequently, there is a need to locate appropriate services and invoke them. This task of locating appropriate agents has become known as the 'connection problem' (Smith 1980). This entails communication with external entities, and thus some mechanism for discovering their identity and how to communicate with them. Further, since service providing entities will differ in their abilities—most obviously because they cannot all implement all possible behaviours, but also because providers will face individual and different resource limitations, security concerns, economic constraints and so on—we must identify our needs and select only those agents capable of fulfilling them. There are three principle ways of identifying agents:

1. *apriori knowledge* The client agent is endowed by its creator with knowledge about other agents and their abilities. This may be done through hardwired code, or in data files that are configured by the user or an administrator.

2. *broadcast discovery* A client broadcasts a service requirement, which is responded to by agents in a position to fulfil the request. The 'contract net' protocol is the best known example of this style (Smith 1980).

3. *matchmaking agents* The client queries a known middle-agent for a set of agents which can fulfil the request. The matchmaker has a database of capabilities, which stores the capability advertisements sent to it by service providers.

The use of apriori knowledge is the easiest to implement initially, but fails to solve several problems. The information can become dated very quickly, especially in dynamic environments. It is difficult to scale, since every client must know about many, and perhaps all, the possible servers: for instance, a mobile email device might need to know about mail servers in every geographical location. There is no intrinsic support for balancing load between the servers. And hard coded references to particular services leave the system prone to downtime, or the modification or movement of the service. The broadcast approach is common in smaller systems, where the level of broadcast communication does not overwhelm the members.

Matchmaking is generally considered the most appropriate for large systems. Because binding between clients and servers happens at a late stage, without involving large numbers of queries passing around, it can be more efficient than broadcasting.

The overall process of enabling matchmaking involves the following actions:

1. Service providers send advertisements, listing their capabilities, to some matchmaking middle agent

2. The matchmaker receives and stores the advertisements

3. A client constructs a description of a service it requires, and sends it to a matchmaker

4. The matchmaker applies some selection algorithm to find suitable providers, and forwards the list to the client.

Some matchmakers offer the client a list of suitable services, leaving the client the final choice. Some systems might carry the interaction further, for instance by executing the client's request and returning the final result.

These three classes are not exhaustive, and there are variations and hybrids. The distributed file transfer system 'BitTorrent'[3], for example, by default uses central lists of peers (called 'trackers'), but has been extended to support operation using distributed hash tables to find peers.

2.2.1 Terminology

Before proceeding, we should pause to consider the various terms used in matchmaking research, which are sometimes vague or have multiple meanings. There is no firm agreement on what constitutes an agent (or more fully, a 'software agent'), but one commonly accepted definition is given in (Wooldridge and Jennings 1995), according to which agents are software-based computer systems possessing four properties: autonomy, social ability, reactivity, and pro-activeness. This goes somewhat further than we need: we will call an agent any entity—software, human, or organisation—which owns and controls a set of services which it provides to others, or any entity which makes use of such services offered by another agent. Since matchmaking, even in multi-agent systems, is principally about the services, the precise definition of agency itself is of no great importance.

A 'middle-agent' is a software program that mediates between other agents, providing infrastructural assistance. There are many classes of middle-agent, but we are concerned only with those that match service-requesters with service-providers. There are several typologies of such agents (Klusch and Sycara 2001; Decker, Sycara, and Williamson 1997), but there seems to be no acknowledged term for the general class of 'matcher' agents. Conventionally, the term 'broker' is used to cover the general term, as well as a more specific one which we look at next.

[3] `http://www.bittorrent.com/`

Figure 2.1 shows the middle agent taxonomy introduced in (Decker, Sycara, and Williamson 1997). Following this classification we define the following: A 'requester' agent has 'preferences', specific characteristics by which it will evaluate offers of service. It may not want to disclose these, and so will make a 'request'—which will hide some aspects of its preferences—that will cause a middle-agent to generate a response consisting of a number of possible 'provider' agents. The requester is then free to evaluate those according to its preferences. Provider agents offer 'capabilities' in 'advertisements' to brokers. Providers are required to fulfil (or 'honestly' attempt to fulfil) any request made to them that they have advertised they can fulfil.

The table in figure 2.1 shows nine types of middle agent, but only three are much discussed in the literature. The first is the 'matchmaker', which stores advertisements from providers, and replies to clients by offering them the names of provider agents: it acts like a 'for-sale' section in a newspaper, or indeed, a Yellow Pages directory. The 'blackboard' performs the inverse function of storing requests, and giving them to providers which contact the middle-agent. This acts like a newspaper's 'wanted' section. A broker hides the client's preferences and the provider's existence—requester and provider remain ignorant of each other's ignorance throughout. In addition, the broker may translate requests across ontologies, ensuring anonymity for client and/or provider, or judge that a not-quite-exact match between request and capability will be acceptable. In common use of the term, a broker is seen as being a more general purpose agent capable of performing several actions on behalf of the requester, such as dealing with invoking several services in a workflow, or managing the choreography of a single agent's several services. We focus in this thesis on matchmakers in the 'matchmaker/yellow-pages' category, but the contribution of our approach can be applied to the others.

Figure 2.1 Types of middle-agent

preferences initially known by	capabilities initially known by: provider only	provider+middle-agent	requester
requester only	broadcaster	front-agent	matchmaker/ yellow-pages
requester +middle-agent	anonymizer	broker	recommender
requester +middle-agent +provider	blackboard	introducer	arbitrator

The well-known taxonomy of middle agents presented in (Decker, Sycara, and Williamson 1997).

2.3 Requirements for matchmaking

What must be provided in order for a matchmaker to operate? From a client's point of view, a matchmaking infrastructure should provide the following:

- *Describe service capability* for both the provider (the advertisement), and the requester (query).
- *Discover services*. 'Discovery' refers to the identification of services and their abilities. Much semantic web services literature uses the term to mean the 'discovery' by a client via a query. We prefer to reserve 'discovery' to the matchmaker when it gains knowledge about services. This can be done by having services send advertisements to the matchmaker, or by having the matchmaker proactively search the Internet for descriptions.
- *Query* When the client, with a description of a required service, makes a query to the matchmaker. The response may be list of suitable agents, or the direct invocation of the service by the matchmaker (acting as a broker) of the selected service.

In addition, the matchmaking process may support the following:

- *Requirements formulation* Often, we are not sure of what exactly it is we want to do. By examining the services offered, we constrain and direct our search for our own requirements.

- *Service ranking* In which the matchmaker rates services in a more precise manner than a simple appropriate/inappropriate decision. This might be done by the proximity of the requested and proffered services as determined by the selection mechanism, or by another means, such as quality of services information stored in the services' non-functional properties, or based on feedback from previous users. In recent literature, the terms 'selection' and 'ranking' have appeared to differentiate the selection a set of possibly applicable services (using, say, subsumption reasoning) from the ordering of those services (e.g. by declared quality of service).

- *Compose component services to meet the requirement* Such a facility is separable from the selection process, but as this thesis suggests, the interaction between service composition and selection may be more important than a simple division of labour would suggest. Having a broker which considered both simultaneously could be beneficial.

- *Invoke services* Many matchmakers only identify a suitable services, while others will perform the invocation on behalf of the client (the `recruit-*`/`broker-*` performatives in KQML/FIPA-ACL). A broker may need to alter the client's requirements in order to find suitable service providers.

- *Monitor invocations* Particularly in long-running Grid processes, we wish to know how far a request has progressed, and how long it will take to complete.

(Burstein et al. 2005) enumerates a three-part process of discovery, engagement, and enactment. 'Candidate service discovery' is the search for a service which may satisfy an agent's current goal. This may be done through peer-to-peer search, querying

a matchmaker or a registry. 'Service engagement' is the checking of constraints against candidates, and negotiation with candidates with regard to the possible costs of the operation, quality of service and so on. 'Service enactment' is the actual carrying out of the requested service. The enactment may be monitored, subject to 'compensation' in the event of faults or the client being unhappy with the resulting transaction. In this work, we are concerned principally with the first stage of finding (possibly) appropriate services. We will cover the other two task only when they impinge on the first.

The service description language is central, and we discuss several approaches in the next section, and particular frameworks in chapter 3. At this stage, we can simply say that it must permit the labelling of the purpose of a service, and its parameter types. In recent years, web services descriptions have made moves towards including issues of grounding, choreography, and orchestration in the service capability descriptions. The language used to must fulfil several criteria. (Sycara et al. 2002) lists four dimensions through which a language can be evaluated:

1. *Expressiveness* The more information clients and servers can provide, the better the match. But overly-rich descriptions can be hard to reason with, and may exclude possibly functional matches. They may also conflict with the 'ease of use' goal.

2. *Inference* Brokers should be able to perform inference on the descriptions.

3. *Ease of use* Engineers who offer services, and clients who wish to use them, must be able to do so without unnecessary difficulty.

4. *Applicability on the web* The web has become the overwhelmingly important venue for agent-like systems, and hence capability description mechanisms should be portable to this environment.

Another viewpoint is provided in (Wong and Sycara 2000), where the authors lay out several dimensions along which middle agents can differ. They define 'end-agents'

as being clients and servers, and 'middle-agents' as those parts of the infrastructure that facilitate end-agents. Their dimensions are:

1. *Who sends information to the middle agent?*
2. *How much information is sent to the middle agent?* In which the options seem binary: either the capability/requirement or the parameters/preferences.
3. *What happens to the information middle agents receive?* Is it broadcast to subscribers, or kept in a database?
4. *How is the content of the database used?* Is it queried or browsed?
5. *How much information is specified in a query to the middle agent?* Does the client divulge its private preferences, or only a very general description of the required service?
6. *Does the middle agent intermediate transactions between end-agents?*

We can also consider a capability description through the various kinds of metadata about a service. (Wroe et al. 2004) identifies seven varieties:

1. *Concept based notion of service*, which explains what the service does (as per a yellow pages).
2. *Configuration metadata* Which enumerate properties on the service which can tweak behaviour. One example is selecting a database for a protein comparison operation.
3. *Provenance* Information about how others have used the service in the past, such as the task/workflow or workflow which engaged the service. (Wroe et al. 2004) mentions hand written annotations to the descriptions, but also automatic aggregation.

4. *Operational description* Information such as the cost, access rights, and quality of service provided.

5. *Invocation model* A description of the choreography of a service, such as the ordering of calls to a stateful service.

6. *Interface* Describing the low-level message format, such as provided by WSDL.

7. *Format* The format of the input and output data, again as might be provided through WSDL.

We could add several more dimensions of our own:

1. *Does the middle agent understand multi-party interactions?* As multi-service workflows become commonplace, should the matchmaker attempt to deal explicitly with the collaborations of several services?

2. *Does the middle agent collect performance data?* That is, might the client inform the middle agent during or after an interaction, of its satisfaction with the recommended agents?

3. *Does the middle agent behave deterministically?* That is, can a matchmaker be regarded as an inert repository of advertisements with known and predictable behaviour, or is it performing computations which are not transparent to the user.

4. *Does the middle agent collaborate with other middle agents?* Is matchmaking somehow distributed?

In this thesis, we argue for matchmakers which do understand multi-party interactions, record performance data, and behave in a (somewhat) non-deterministic fashion to explore possibilities. We do not investigate distribution or interaction amongst matchmakers, but we do discuss it as further work in section 9.2.3.

2.4 Service description

So, how are services described? The literature is full of many approaches, and we remain far from agreeing on a language for capability representation. There are too many trade-offs, and some of the issues are too poorly understood to allow for an optimal, multi-domain solution. For instance, the software design language Z (Spivey 1992) is suitable for describing interfaces and processes in software, but it is too rich or complicated to support the needs of on-demand brokering. Conversely, planning languages are easier to reason with, but impose restrictions on expressiveness. In this section, we survey the more prominent styles of capability description that have been used in matchmaking systems.

Strings

Ultimately, almost all means of identifying anything come down to strings. The simplest service description language, then, is one in which we simply name, with a string, a service's type. Agreement on the meaning of a service identified by a string can be achieved by having a global ontology of such types. Such a system is used in the Lightweight Coordination Calculus, on which we build our matchmakers, as well as the Open Agent Architecture (Martin, Cheyer, and Moran 1999). Using strings is simple, and often effective. Because the strings express little structure over which to reason, matchmakers are unable to find approximate matches for requested service functionality.

Vector spaces

Vector space models are one approach to introducing approximate matches. A service is represented by a point in a vector space of terms describing the service offering. The vectors are typically derived from a controlled vocabulary or ontology, and nearness is determined using some metric like cosine distance. For example, a request for a service "weather forecast Birmingham" might match against services described as "weather

forecast Birmingham Alabama" and "weather forecast Birmingham England", and have near misses with "weather report Birmingham". The more terms which match, the more appropriate the service. The notion of boolean matches can be extended to a more general concept of distance between services in some kind of space, so that the "Birmingham" weather service could be selected if the user had requested a service for the physically nearby town of "Telford" which did not have its own service, as long as "Telford" and "Birmingham" are nearby in the location dimension of the vector space. This approach provides a basis for systems based on nearest neighbour selection, such as that found in the IMPACT system (Subrahmanian et al. 2000) and the information retrieval inspired MX matchmaker for semantic web services (Klusch, Fries, and Khalid 2005).

Subsumption

By subsumption, we mean the hypernym-hyponym relationship, or more prosaically, *is-a*. Thanks to object orientated software engineering's concept of sub-classing, this notion is probably the dominant model of thinking in software development. It is also the most prominent form of reasoning on today's semantic web, because of the dominance of description logics (Baader et al. 2003).

Description logics are a fragment of first order logic, selected to provide a balance between computational performance and expressiveness. Expressive DLs are decidable, but computationally intractable in the worst case—in the EXPTIME to NEXPTIME complexity classes. Modern implementations offer good average case performance (Tobies 2001; Horrocks and Patel-Schneider 1999). Description logics underpin the more sophisticated current applications of the semantic web, in particular the OWL family of languages (Smith, Welty, and McGuinness 2004). Because of the Semantic Web's use of OWL, and the availability of reliable subclass relationship reasoning, most semantic web services frameworks use a subsumption model as a basis for service selection (Sycara et al. 2003*b*).

Description logics split their knowledge base into a 'terminological box' (or 'TBox') and an 'assertion box' ('ABox'), and often written $\mathcal{KB} = \langle \mathcal{T}, \mathcal{A} \rangle$. The TBox holds the definitions of classes and relationships between, while the ABox holds assertions of particular individuals. Subsumption reasoning works in the TBox to decide the is-a relation.

DLs use a specialised syntax intended to be suggestive of the first order logic statements they map to. Concepts (that is, classes) are strings, usually capitalised, such as Person or Car. These can be combined using a variety of operators, which vary between logics, and commonly include equivalence $\equiv$, conjunction $\sqcap$, disjunction $\sqcup$, negation or complement $\neg$, and the relations for subsumes $\sqsubseteq$ and properly subsumes $\sqsubset$. A concept A subsumes a concept B, written $A \sqsupset B$, if all possible instances of B are also members of A. The 'top' concepts, which subsumes all other concepts is written $\top$, and the unsatisfiable concept is $\bot$. Relations between concepts are defined by 'roles', and role restrictions on concepts which are written . roleName. Other features from the OWL DL are shown in figure 2.3. DLs are frequently referred to by acronyms which identify the features used in the language (table 2.2). OWL-DL is effectively $\mathcal{SHOIN}(D)$.

The following is a simple ontology about pizzas:

Figure 2.2 Description logic features

$\mathcal{S}$	A shorthand for $\mathcal{ALC}$ with transitive roles. $\mathcal{ALC}$ is a fundamental DL which allows complementation of complex concepts, not just atomic ones
$\mathcal{H}$	role hierarchy: roles can have super- and sub-roles, and subsumption, like concepts
$\mathcal{O}$	Enumerated classes (e.g. a week as the union of the days in it)
$\mathcal{I}$	Inverse properties
$\mathcal{N}$	Cardinality restrictions on roles
$\mathcal{Q}$	Qualified cardinality restrictions (OWL 1.1)
(D)	Indicates support for data types, such as strings or integers

$$
\begin{aligned}
\mathsf{Pizza} &\equiv \mathsf{hasBase}\,.\,\mathsf{PizzaBase} \sqcap \mathsf{hasTopping}\,.\,\mathsf{PizzaTopping} \\
\mathsf{PizzaBase} &\sqsubset \top \\
\mathsf{DeepPanBase} &\sqsubset \mathsf{PizzaBase} \\
\mathsf{ThinBase} &\sqsubset \mathsf{PizzaBase} \\
\mathsf{Vegetable} &\sqsubset \top \\
\mathsf{Tomato} &\sqsubset \mathsf{Vegetable} \\
\mathsf{Olive} &\sqsubset \mathsf{Vegetable} \\
\mathsf{Fish} &\sqsubset \top \\
\mathsf{Anchovy} &\sqsubset \mathsf{Fish} \\
\mathsf{Fish} \sqcap \mathsf{Vegetable} &\equiv \bot \\
\mathsf{VegetarianPizza} &\equiv \mathsf{Pizza} \sqcap \forall\, \mathsf{hasTopping}.\mathsf{Vegetable}
\end{aligned}
$$

Quibbles about the vegetable status of certain fruits aside, the TBox above shows how

a vegetarian pizza can be defined. We need to state explicitly that Fish and Vegetable are disjoint, so we define their intersection to be empty ($\bot$). If we then talk about a pizza type

$$\mathsf{Pizza} \sqcap \mathsf{hasBase} \,.\, \mathsf{ThinBase} \sqcap \mathsf{hasTopping} \,.\, \mathsf{Anchovy}$$

we can infer that is not a vegetarian pizza: it violates the definition of VegetarianPizza in having Anchovy as a topping. The role VegetarianPizza . hasTopping must always have a value of type Vegetable, according to the last line in the TBox, and we know that since Anchovy $\sqsubset$ Fish and Fish and Vegetable are disjoint, Anchovy cannot be a vegetable.

In this way, subsumption can be used to determine if a requested service can be satisfied by an advertised service by checking if the former is a sub-concept of the latter. This is explored in greater detail when we explore semantic web matchmakers in section 3.5.

Figure 2.3 Description logic operations

Operation	DL Syntax	First order logic
Atomic concept	C	$C(x)$
Top or universal concept	$\top$	$true$
Bottom or unsatisfiable concept	$\bot$	$false$
Concept equivalence	$\mathsf{C_1} \equiv \mathsf{C_2}$	$\forall x.C_1(x) \leftrightarrow C_2(x)$
Concept intersection	$\mathsf{C_1} \sqcap \ldots \sqcap \mathsf{C_n}$	$C_1(x) \wedge \ldots \wedge C_n(x)$
Concept union	$\mathsf{C_1} \sqcup \ldots \sqcup \mathsf{C_n}$	$C_1(x) \vee \ldots \vee C_n(x)$
Concept complement	$\neg C$	$\neg C(x)$
Value restriction	$\mathsf{P} \,.\, \mathsf{C}$	$\forall y.P(x,y) \rightarrow C(y)$

DL concepts map to unary predicates in first order logic, while roles correspond to binary predicates.

Horn clauses

Horn clauses, or Prolog goals, are a common way to represent capabilities. Because Prolog (Clocksin and Mellish 2003) is a popular language in AI, and its notion of goals approximates the notion of service, it is a natural approach. Using this, we can very quickly build a service composer, matchmakers and so on. In the simplest scenario, a service is described simply as a predicate, perhaps with typing information:

$$send\text{-}email(Recipients : setof(EmailAdress), Message : Text)$$

or, alternatively, using explicit predicates to describe the types:

$$send\text{-}email(Recipients, Message) \leftarrow$$
$$type(Recipients, setof(email\text{-}address)) \wedge type(Message, text)$$

A more sophisticated language would allow constraints to be expressed as part of the Horn clause. We will see several examples of systems based on Horn clauses in section 3.2.

Problem solving methods

Problem solving methods are a unique approach with the area. The notion of 'problem solving methods' (PSM) emerged when first-generation expert systems matured and researchers discovered common patterns of inference in them. Such patterns include 'heuristic classification' (Clancey 1985) and 'propose and revise' (Marcus and McDermott 1989). Once identified, these implicit behaviours can be made explicit in the knowledge base, by recording them as goals to be achieved, and knowledge required to fulfil them. In so doing, the PSM becomes reusable across tasks. According to (Studer, Benjamins, and Fensel 1998), a PSM must specify:

- The inferences required to implement the method
- The ordering of those inferences

- The domain knowledge inputs to the method in terms of domain independent knowledge roles

'Bridges' are used to connect the generic PSMs to 'tasks' in specific problem domains. The PSMs can then be reused across many knowledge based systems, and in different domains.

The principles of PSMs have been used directly in the web services field, in (Teije, Harmelen, and Wielinga 2004) developed from earlier work in pure PSMs (Teije et al. 1998), and in the IBROW project (Benjamins et al. 1998). PSMs are alleged to underpin the WSMO approach (Fensel et al. 2006), but the link can appear tenuous: WSMO retains the clear separation of goal (task) from service (PSM), and the notion of placing mediators (bridges) between the goal and service, but the equation of services with PSMs themselves is not so clear. PSMs are more similar to general-purpose algorithms, requiring configuration to a task, while typical web services are more special-purpose, state-altering, and are composed together rather than refined to achieve particular tasks (Teije, Harmelen, and Wielinga 2004).

2.5 Related fields

Since our approach to matchmaking is to look beyond the purely logical and symbolic machinery, we now briefly look at approaches which border on our own. These include recommender systems, self-organising agent systems, and autonomic computing. Because of the close coupling of our approach to information retrieval and reinforcement learning, we cover both those topics later, in section 7.4, after we have introduced the matchmakers.

Recommender systems collect individuals' ratings of items into a collective valuation of objects (Resnick and Varian 1997) that can be used by a community. Recommender systems emerged from work on 'collaborative filtering', such as the GroupLens work on Usenet articles (Konstan et al. 1997). Simple aggregations policies may value items

based on the average of users' ratings, while more sophisticated ones can characterise the preferences of individuals, and by clustering those preferences, suggest particular items to individuals based on the the intersection of their own recommendations and those of others. Such systems are becoming common in e-commerce settings (Schafer, Konstan, and Riedl 1999), where users' preferences for books or films, for instance, can be processed to discover groupings of like-minded users who can then have the recommendations of their cohort forwarded to them.

In multi-agent systems, distributed mechanisms have been developed to enable agents to build a team or coalition of peers with which to work. Such 'agent organised networks' (AONs) are set of agents which have chosen a set of peers with which they directly interact, based on local decisions made by the individual agents (Gaston and desJardins 2005). Such systems are often used to perform network management, where the networks can be for computer communications, logistics, or social networks. For example, in sensor networks or 'pebble-nets', physically small devices with limited communication range must collaborate with their peers to provide the community with communications. Their options are limited by the physical connectivity of the individual units, but the agents can reason over their individual goals such as power conservation and task achievement, and their own appraisal of local network structure, to perform network formation and routing (Glinton, Scerri, and Sycara 2008). The agents in AONs are typically homogeneous, interact on an ongoing basis, and have some kind of reciprocity.

Autonomic computing is an attempt to instill computers, particularly distributed ones, with the ability to regulate themselves in the same way that humans and other animals automatically manage their own complex internal state through their autonomic nervous system (Horn 2001). The concerns of autonomic computing focus on taming the complexity of contemporary computer systems, and the complexity of integration is one of their key concerns. Since humans are struggling to cope, they argue that machines themselves should shoulder more of the responsibility for their own robust operation.

They invoke notions of ‘self-configuring’, ‘self-optimising’, and ‘self-healing’ systems, but anticipate an evolutionary approach rooted in improved engineering, rather than one based on artificial intelligence (Ganek and Corbi 2003).

2.6 Summary

In this chapter we looked at the background to service matchmaking: its setting in distributed systems, the identification of the connection problem, and the typical approaches to solving it. Key points from this chapter are:

- The notion of agents or services which can be invoked over the network by other services or users is common to a number of fields including distributed problem solving, multi-agent systems, web services, Grid computing.
- In heterogeneous systems or autonomous systems, the agents involved need a way to both describe their abilities and to locate other agents which can provide a required service.
- Service capability languages are used by service providers and requesters to define a service’s competence. Such languages have taken many forms, including free text descriptions, Prolog-like goals, and description logics.

Chapter 3

Matchmakers

The previous chapter outlined the background against which matchmaking happens. In this chapter, we advance to the foreground, studying specific frameworks for matchmaking and brokering, and several implemented matchmakers. We take a historical approach, proceeding through distributed AI (section 3.1), multi-agent systems (section 3.2), web services (section 3.3), and the workflow systems built on web services (section 3.4). Finally, we look at semantic web services in section 3.5.

3.1 Distributed AI

As noted earlier, distributed AI (or distributed problem solving) was the first field to encounter and formalise the connection problem as we now know it. Two early systems were ETHER and the much better known Contract Net.

ETHER

ETHER (W A Kornfeld 1979; William A. Kornfeld 1979; William A. Kornfeld 1981) is a language and platform for writing distributed applications. ETHER is based on the notion of a distributed database of logical assertions. The statement `(broadcast` ϕ`)` asserts the fact ϕ, while the corresponding `(when` ϕ`)` can read such assertions. These are used by so-called 'sprites', which are analogous to daemons in frame languages. For

example, consider the following sprite code which deals with information about the location of objects using the `in` relation:

```
(when (in ?x ?y)
  (when (in ?y ?z)
    (broadcast (in ?x ?z))))
```

This sprite updates the distributed database with relations resulting from the transitivity of `in`: upon discovering `?x` is in `?y` and that `?y` is in `?z`, assert the fact that object `?x` is in `?z`. In this manner, the sprites operate as forward-chaining rules. To implement goal-solving behaviour, ETHER simply marks the patterns with an additional `goal` predicate:

```
(when (goal (in ?x ?y))
  (when (in ?x ?z)
    (broadcast (goal (in ?z ?y))
      (when (in ?z ?y)
        (broadcast (in ?x ?y))))))
```

By delegating goal solving to 'activities' (processes) running in parallel, and using goal matching to communicate, a primitive form of matchmaking is achieved. This style of matchmaking makes for a simple programming model, but does not leave much space for decision-making by client agents. While not commonly used as a model for matchmaking, it is a fore-runner not only of Linda-like systems (Carriero and Gelernter 1989), but of a proposed but unimplemented matchmaker system based on triple spaces (Toma et al. 2005).

Contract Net

Contract Net was developed in the context of a distributed sensing application (Smith and Davis 1978), and more recently, a standardised version of Contract Net for the FIPA-ACL agent language has emerged (Foundation for Intelligent Physical Agents 2002).

A contract net is a collection of nodes which engages in a protocol of issuing tenders and contracts to one another for services. The nodes have a shared objective (the solving of the distributed problem), and cooperate to share computational resources by handing

off tasks to the most appropriate nodes. This is done through the creation of tenders, bids, and contracts between the nodes. A 'manager' is a node which requires a task to be done, and which initiates an invitation for tenders, appoints the successful winner, and monitors task completion. The 'contractor' is a node which bids for the task, and performs the required computation. Nodes can be managers or contractors as needs dictate, and can be both simultaneously (if, for instance, a task can be split into sub-tasks which are in turn put out to tender to other nodes). Tenders can be sent to specific nodes, groups of named nodes, or broadcast. Nodes with tasks to achieve need to find nodes which are best suited to achieving them: the node must not only be able to carry out the task, it should also be less heavily loaded than those nodes which lose the tender.

A 'task announcement' message is sent (often as a broadcast message), detailing the work required by the manager node. A simple announcement requesting a temperature reading might be:

```
To: *
From: 43
Type: TASK ANNOUNCEMENT
Contract: 12345
Task abstraction: TASK TYPE SENSE-ENVIRONMENT
Eligibility: MUST-HAVE THERMOMETER
             MUST-HAVE LOCATION LONDON
Bid specification: POSITION LAT LONG
                   EVERY SENSOR NAME TYPE
Expiration time: 0930 1 May 2009
```

The 'eligibility' criteria state that only bids from environment sensors with a thermometer, located in London, are of interest. Any node responding must meet the 'bid specification' by supplying its position in latitude and longitude, and a list of its sensors. Those nodes which satisfy the criteria, and which find themselves underutilised, construct a bid according to the specification and send it to the manager node. The manager node considers incoming bids, and whenever it is sufficiently happy with a bid, can award the contract by issuing an 'announced award' message. Contract Net defines the slot/attribute framework used to describe tasks, but the actual terms and format of the task definitions are left to the designers of domain specific Contract Nets.

3.2 Multi-agent systems

The deepest well of matchmaking research is to be found in the multi-agent systems domain. Because of their heterogeneous nature, and the high degree of self-reflective knowledge available to an agent, these systems have a richer set of formalisms to build on, and thus more scope for exploring the matchmaking design space. We begin with a look at the widely used agent communication language KQML (which later became FIPA-ACL) since many of the matchmakers use it.

KQML and FIPA-ACL

Agent communication languages (ACLs) are used by agents to communicate with one another. Of the many ACLs invented, one in particular has been widely adopted: the Knowledge Query and Manipulation Language (KQML) (Labrou and Finin 1997). KQML was developed as part of the DARPA Knowledge Sharing Effort programme, and was later adopted—with modifications—by the Foundation for Intelligent Physical Agents (FIPA) as the FIPA Agent Communication Language (FIPA-ACL). The differences between KQML and FIPA-ACL are relatively small, and for our present purpose we will treat them as interchangeable.

KQML is based on the notion of 'speech acts' (Austin 1976), (also known as 'illocutionary acts') a theory in which an agent effects an action or change in the world by the mere action of saying something. For instance, my saying 'I will meet you for lunch' is not 'stating a fact' in the sense of a sentence in traditional logic, since I may be run over by a bus on the way there. I am, instead, establishing a contract between us that I will endeavour to meet you for lunch. Such statements are said to have 'illocutionary force'. Agents' messages are labelled with 'performatives', such as `tell`, `ask`, and `request`, but KQML does not specify the content of those messages, and is instead a wrapper around statements made in a 'content language', such as the Knowledge Interchange Format (KIF) or Prolog.

KQML defines several performatives related to facilitation[1]. The first allows an agent to advertise its ability to accept a performative:

```
(advertise :sender <advertiser> :receiver <facilitator>
  :content (<performative> <form>))
```

Where `<performative>` and `<form>` together define a message that can be handled by the `<advertiser>`. For example

```
  (ask (weather-in spain ?weather))}
```

where the `<performative>` is the `ask`. Similarly, an agent can subscribe to updates on another agent's knowledge base:

```
(subscribe :sender <tourist> :receiver <weather-agent>
  :content (weather-in spain ?weather))
```

whereupon the `<weather-agent>` will send a `tell` message to the tourist each time the weather agent's knowledge about the `weather-in` relation changes. Agents can query a facilitator's knowledge of advertisements with the `recommend-one` performative:

```
(recommend-one :sender <requester> :receiver <facilitator>
  :content (ask (weather-in spain ?weather)))
```

If the facilitator can unify the value in the content slot of the message with that from a prior `advertise`, it will forward to the `<requester>` the original advertising message. The `<requester>` can then process that advertisement, and engage with the advertiser directly. A similar performative called `recommend-all` results in the facilitator notifying the client of all matching adverts. Alternatively, the client can use the `broker-one` or `broker-all` performatives, which cause the facilitator to communicate with the advertisers directly, on behalf of the client, and to forward the results back to the client. Finally, `recruit-one` and `recruit-all` behave like their `broker-` equivalents, but the results from the service providers are sent directly to the client.

We will see some of these in use later in this section. Note that although KQML imposes the severe restriction that advert and request must unify syntactically, most

[1] We simplify the syntax of the operations somewhat, to draw attention to the essentials. Details can be found in (Labrou and Finin 1997).

matchmakers—including those reviewed here—go beyond this to offer richer behaviour.

ABSI

ABSI (Agent-Based Software Interaction) (Singh 1993) was one of the first brokering systems. It extended the basic KQML brokering model with the capability of matching requests and advertisements that are not exact matches, using unification over the queries and adverts. An interesting, arguably fatal, limitation on ABSI brokering is the requirement that service providers are not allowed to fail when requested to fulfil a capability they have advertised.

SHADE and COINS

Developed by the same researchers, SHADE (Kuokka and Harada 1995), and COINS (Kuokka and Harada 1996) were used in information management systems. Both use KQML to convey the matchmaking operations, but while COINS uses a weight similarity measure from information retrieval, SHADE uses logic rules and unification. A typical SHADE service advert is

```
(advertise :sender inf.ed.ac.uk :receiver mm
           :language kqml :content
  (ask-one :language kif :content (supervises ?x ?y)))
```

which offers an information providing service which determines if two individuals have a supervisor/supervisee relationship. A corresponding query is of the form

```
(recruit-all :sender c :receiver mm :language kqml :content
  (ask-one :language kif :content (supervises dave ?y)))
```

SHADE uses unification over the KIF query form to determine suitable matches: in this case, `dave` is unified with `?x` and the service `inf.ed.ac.uk` can be recruited. The unification is done only at a syntactic level, so an equivalent request like `(supervised-by ?y dave)` would not match, even if it is known the relations `supervised-by` and `supervises` are inverses.

In contrast, the COINS matchmaker works on free-text services capability descriptions. Clients subscribe to the matchmaker using a query which can be either a text

document or a weighted term vector. For example, a query for documents related to the subject of this thesis might be described by the following:

```
(subscribe :sender student4 :receiver mm :language kqml :content
  (stream-all :language document-vector :content
    (?document matchmaking 5 service 4 information 3 retrieval 3)))
```

A library agent could advertise documents to the matchmaker as they become available:

```
(tell :sender library :receiver mm :language kqml :content
  (stream-all :language document-vector :content
    (matchmaking-thesis.pdf matchmaking 5 service 4
     information 3 retrieval 3)))
```

On receiving such an advert, the matchmaker would send a notification to `student4` about the new document which matched their query. Query matching is done by a vector similarity computation, as typically found in information retrieval systems.

IMPACT

IMPACT is an agent architecture detailed in (Subrahmanian et al. 2000). For service description, it primarily relies on a naming convention where services are named in the form 'verb:noun': e.g. `forecast:weather` or `plan:flight`. The verbs and nouns are both drawn from controlled taxonomies. Subclass relations in the taxonomies have attached weights, indicating the conceptual difference between direct super- and sub-concepts. Using the weights, a distance metric is defined over all pairs of verbs, and all pairs of nouns. From this, a distance is defined between service descriptions, and a nearest neighbour algorithm used to match queries to services.

RETSINA/LARKS

RETSINA (Sycara et al. 2003*a*) is a major multi-agent architecture, based on KQML and using middle agents. RETSINA's capability description language is known as LARKS (Sycara et al. 2002), for 'Language for Advertisement and Request for Knowledge Sharing'.

A LARKS specification is a frame with slots specifying context, typed input and output variables, constraints on the inputs and outputs, conceptual descriptions for

ontological definitions, and a textual description of the advertisement. Constraints are written as Horn clauses, while the optional conceptual descriptions are used to link other elements in the specification to an ontological representation. Terms used without an explicit conceptual description are required to be known to the matchmaker already. Both advertisements and requests are communicated as specifications, and KQML performatives must be used to distinguish them. Since requester and provider need not share an ontology, the broker attempts to map between ontologies, maintaining a global ontology of all concepts contained in service advertisements, and assuming certain basic concepts are shared between all ontologies. RETSINA's ontology language, ITL, is specific to it, but is similar to KL-ONE (Brachman et al. 1990).

RETSINA defines several classes of 'match', a classification which is a precursor to that which now dominates matchmaking in semantic web services. The matches, in decreasing precision, are:

- *exact* match where the request and advertisement are identical, modulo variable renames or some other equality preserving inferences.
- *plugin* match, where the request is essentially a subconcept of an advertisement. For example, a request to hire a small car is a plugin match with a company which hires out all kinds of vehicles.
- *relaxed* match, where the logical subsumption matches have failed, and an approximate match is made by measuring numerical similarity between request and service.

Unusually, LARKS uses a series of techniques to match queries. The exact set and order of application can be determined by the client. Most of these filters make use of a measure of semantic distance between concepts, using notions of generalisation, specialisation, and general positive association. LARKS generates these scores by comparing terms through subsumption reasoning and WordNet. In increasing computational cost, the filters are:

1. Content matching
2. Profile comparison
3. Similarity matching
4. Signature matching
5. Constraint checking

The context filter discards adverts which do not match the query context (for example, a 'travel' context would not be of use in a query looking for television schedules). Context labels are compared using the semantic similarity score. Profile comparison treats adverts and queries as documents, and applies a TF-IDF[2] metric to compare them, just as in an information retrieval system, and the COINS matchmaker. Similarity scoring compares the inputs and outputs pairwise, ensuring a tighter check than the profile comparison, which does not distinguish input from output. Signature matching applies subsumption reasoning to the inputs and outputs to determine the exact/plugin relationship. Finally, constraint checking applies an inference engine to the Horn clauses which specify the pre- and post-condition constraints.

Open Agent Architecture

The Open Agent Architecture (OAA) (Martin, Cheyer, and Moran 1999) was developed at the Stanford Research Institute. It uses a Prolog style syntax, but like KQML, makes a distinction between the communication language and the content language. Services are represented as Prolog goals in the form

```
solvable(Goal,Parameters,Permission)
```

The `Goal` is a Prolog term, such as

```
get_weather_report(+Location,-Report)
```

[2]Term frequency inter-document frequency, which weights terms in proportion to their frequency in a single document, and that term's frequency in the corpus as a whole.

while `Parameters` adds information about the service (such as whether it returns information or carries out an action, and whether the service is synchronous or asynchronous). These `solvable`s are advertised to a facilitator. A service requester invokes a goal using a library goal `ooa_Solve`, which will request the facilitator to find an appropriate provider and invoke the required goal. Matching is done by unification of the goal form with advertised forms: no approximate matching is done.

LCCM

Another system using Prolog goals to describe capabilities is the Lightweight Capability Communication Mechanism (Robertson et al. 2000). The objective of LCCM was to provide a lightweight description language using the well-understood formalism of Horn clauses. Agent capabilities are expressed as Horn clauses, with an agent asserting it has a capability when it can satisfy a goal head provided certain preconditions (the 'goal body') are met. For instance, the Internet Domain Name Service might advertise its primary function of domain name lookups as

$$capability(dns, ip\text{-}address(Ip, Name) \leftarrow domain\text{-}name(Name))$$

indicating that the DNS agent would provide an IP address given a valid domain name. The preconditions for a capability may be provided by the same agent, or another the broker is aware of. An advantage of LCCM descriptions is that they combine naturally with Prolog's backtracking search to provide service composition. That is, by invoking a meta-interpreter over LCCM descriptions, which are essentially Prolog goals, a solution to the search results in a broker structure which captures the necessary tree of actual service invocations. In response to a client query, the broker 'solves' the problem in terms of advertisements, producing from the resulting goal tree a list of the required interactions and dependencies amongst the real agents.

By way of example, we will look at the case of a dietitian who calculates the body mass index (BMI) of patients. The dietitian performs this by dividing the patient's

weight by the square of their height. This gives a number that should be in the range 19 to 25. But this must be applied to measurements in metric units. Should an American patient use this service, they would need a converter. The conversion can be done before or after the dietitian performs the calculation. This gives us the capability advertisements in figure 3.1. The LCCM broker, in response to the request $broker(bmi(george, BMI_{metric}, \text{-})$, will supply two performative sequences, shown in figure 3.2. The LCCM broker incorporates a mechanism for dealing with ontological alignment. This operates by agents specifying explicit correspondences with other agent's capabilities.

3.3 Web services

Web services have become the most popular means for providing distributed computing functionality. They operate using open standards, primarily the widespread HTTP protocol (Fielding et al. 1999), the URL (Berners-Lee, Fielding, and Masinter 2005), and XML (Bray et al. 2008). There are three principle flavours of web services: SOAP, XML-RPC, and RESTful, which we discuss over the next few pages. For vanilla web services, unadorned with semantic descriptions, the means for 'matchmaking' are three-fold:

1. *Implicit knowledge*, where the developer knows about the service already, or discover it through soft means like documentation.

2. *Web search*, where a developer finds a service through standard web search engines, or browsing.

3. *UDDI directories*, the officially sanctioned mechanism for registering and discovering SOAP services described with WSDL, the Web Service Description Language (Christensen et al. 2001).

Figure 3.1 LCCM agent capability advertisements

$$capability(dietitian, (imperial\text{-}bmi(Person, BMI, Height, Weight) \leftarrow$$
$$height(Person, imperial, Height, \text{-}) \wedge$$
$$weight(Person, imperial, Weight, \text{-})))$$
$$capability(dietitian, (metric\text{-}bmi(Person, BMI, Height, Weight) \leftarrow$$
$$height(Person, metric, \text{-}, Height) \wedge$$
$$weight(Person, metric, \text{-}, Weight)))$$
$$capability(dietitian, (bmi(Person, BMI, BMI) \leftarrow$$
$$metric\text{-}bmi(Person, BMI, Height, Weight)))$$
$$capability(converter, (height(\text{-}, metric, Height_{imperial}, Height_{metric}) \leftarrow$$
$$height(\text{-}, imperial, Height_{imperial}, \text{-})))$$
$$capability(converter, (weight(\text{-}, metric, Weight_{imperial}, Weight_{metric}) \leftarrow$$
$$weight(\text{-}, imperial, Weight_{imperial}, \text{-})))$$
$$capability(converter, (bmi(Person, BMI_{metric}, BMI_{imperial}) \leftarrow$$
$$imperial\text{-}bmi(Person, BMI, Height, Weight)))$$
$$capability(american, height(george, imperial, H, \text{-}))$$
$$capability(american, weight(george, imperial, W, \text{-}))$$
$$capability(european, height(jacques, metric, \text{-}, H))$$
$$capability(european, weight(jacques, metric, \text{-}, W))$$

Figure 3.2 Two performative sequences for calculating body mass indices

$ask(american, height(george, imperial, Height_{imperial}, \text{-}))$

$ask(converter, height(george, metric, Height_{imperial}, Height_{metric}))$

$ask(american, weight(george, imperial, Weight_{imperial}, \text{-}))$

$ask(converter, weight(george, metric, Weight_{imperial}, Weight_{metric}))$

$ask(dietitian, metric\text{-}bmi(george, BMI_{metric}, Height_{metric}, Weight_{metric}))$

$ask(dietitian, bmi(george, BMI_{metric}, BMI_{metric}))$

$\Rightarrow BMI_{metric} = 23.8$

$ask(american, height(george, imperial, Height_{imperial}, \text{-}))$

$ask(american, weight(george, imperial, Weight_{imperial}, \text{-}))$

$ask(dietitian, imperial\text{-}bmi(george, BMI_{imperial}, Height_{imperial}, Weight_{imperial}))$

$ask(converter, bmi(george, BMI_{metric}, BMI_{imperial}))$

$\Rightarrow BMI_{metric} = 23.8$

For XML-RPC and RESTful services, the third is not an option, since there are no accepted means for describing such services in any formal language. First, we look at SOAP and UDDI.

3.3.1 SOAP

Originally an abbreviation for Simple Object Access Protocol, SOAP is no longer an acronym: it was never an 'object access protocol', and is now not so simple. SOAP is based on XML messages formatted in 'envelopes' with a core message inside. The idea of the envelope is to allow intermediate services which the message encounters en-route to be instructed by the envelope to perform certain tasks. The core of a soap message is in the `Body` element:

```
POST http://weather.org/soap
SOAPAction: WeatherForecast

<?xml version="1.0"?>
<soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:w="http://weather.org/soap/">
  <soap:Body>
    <w:GetWeatherForecast>
      <w:Country>Spain</w:Country>
      <w:City>Madrid</w:City>
    </w:GetWeatherForecast>
  </soap:Body>
</soap:Envelope>
```

In this example, we see an invocation of a hypothetical weather forecasting service. The structure of the message is tightly defined, and may be described in a document containing 'WSDL'. The Web Services Description Language (WSDL) (Christensen et al. 2001) and its later incarnation 2.0 (Booth and Liu 2007) is an XML vocabulary for describing SOAP services. It is essentially an interface definition language, and a file of WSDL defines the web equivalent of a function library in a conventional programming language. Bindings are provided to various extant data transport mechanisms, including SOAP, HTTP, and email[3].

[3]The justifiability of calling these services over non-HTTP transports *web* services has been questioned (Richardson and Ruby 2007). In practice, SOAP is used over HTTP.

Owing to the WSDL definition, SOAP is the most discoverable of the web services styles. Not only does WSDL provide a syntactic interface, the content itself is identifiable as WSDL, and there is a framework for registering it: UDDI.

UDDI (Universal Description, Discovery, and Integration) (OASIS UDDI Specification TC 2005)[4] was created through the OASIS standards group[5]. UDDI is a registry-based scheme, where service providers lodge three types of information with a registry:

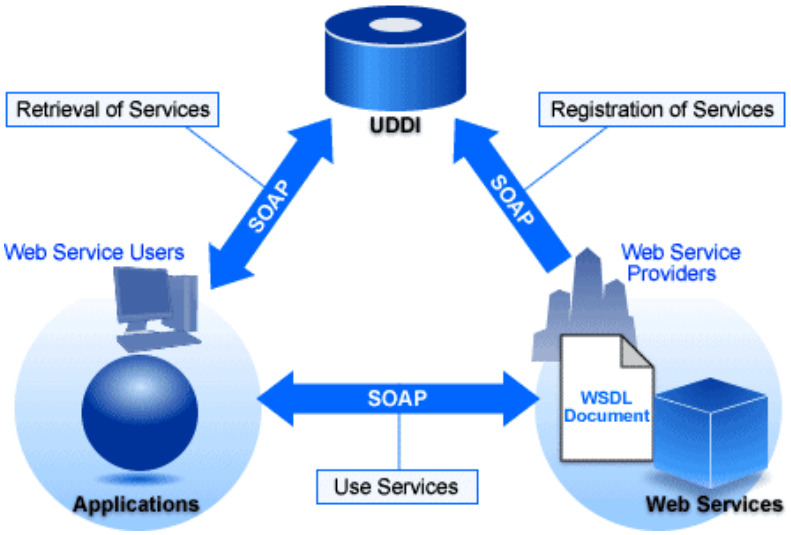
- *white pages* holding the contact information for the provider, such as the business name, phone numbers, and email address.
- *yellow pages* which describe the the nature of the service provided, by reference to a taxonomy.
- *green pages* containing technical information, including, but not limited to, WSDL descriptions.

This UDDI registry is then queried, either through a web browser interface, or programmatically through SOAP, to find an appropriate service provider (figure 3.3). However, UDDI relies on human intervention, and inflexible patterns of service provision. The taxonomies used to classify services under UDDI tend to be difficult to navigate.

UDDI was created as a key component of the Web Services stack, but initial optimism about its use has waned. The large public UDDI repositories (known as 'Universal Business Registries') hosted by IBM, Microsoft and SAP have been shut down (SAP News Desk 2005). Vendors say that the public registries were created to demonstrate UDDI's scalability and inter-operation abilities, which has been achieved. Critics counter that the public registries have been retired because businesses don't operate with the public registry model of dynamic binding of service providers, but instead have private relationships supported by private UDDI registries. Internal company use of UDDI is hard to gauge. In semantic web services work, semantic descriptions have been embedded in UDDI using the green pages 'TModel' facility (Paolucci et al. 2002*a*).

[4] `http://uddi.xml.org/`
[5] `http://www.oasis-open.org/`

Figure 3.3 Service providers, consumers, and UDDI



Graphic from Fujitsu.

3.3.2 XML-RPC

XML-RPC (Winer 1999) was the original inspiration for SOAP, but it has recently gained some traction in its own right, filling a niche for simple XML RPC over HTTP. The relation with SOAP is obvious from its appearance:

```
POST http://weather.org/URL2

<?xml version="1.0"?>
<methodCall>
  <methodName>WeatherForecast</methodName>
  <params>
    <param><value><string>Spain</string></value></param>
    <param><value><string>Madrid</string></value></param>
  </params>
</methodCall>
```

The biggest difference is the simplicity: XML-RPC does not use XML schemas, encoding styles, or bindings to different protocols. There is also no service description language, and no desire for one from users.

3.3.3 REST

'Representational State Transfer' (REST) is an architectural style typical of hypermedia systems, and epitomised by the Web. REST was formalised in (Fielding 2000), and has become a popular way to express web services. The core concept is the notion of resources and using globally unique identifiers for them. In HTTP, the resources are identified by URIs, which can be accessed through the standard HTTP. In REST, a request for information becomes a simple HTTP `GET` request:

```
GET http://weather.org/weatherForecast/Spain/Madrid HTTP/1.1
```

Other HTTP methods—most commonly `POST`, `PUT`, and `DELETE`—are used to effect changes of state. The RESTful or RESTian approach has recently gained mindshare amongst web services developers and users for two reasons. First, its alignment with the HTTP protocol makes for a more 'Webby' feel, with URIs being used not just to name an operation in a transient remote procedure call, but to identify resources which have an ongoing identity. As a consequence, REST takes better advantage of existing HTTP infrastructure—in particular, many operations benefit from having their results memoised through the standard distributed caching mechanisms for HTTP. Secondly, and probably more influential in its popularity, REST is extremely light-weight to work with, requiring no new tooling as SOAP tends to.

Despite a cultural bias against tooling and therefore machine readable descriptions for RESTful services, several non-semantic description languages for REST services have been promoted. WSDL version 2 (Booth and Liu 2007) includes some support for describing the interface, but it is limited to simple cases of parameter insertion in the URL, and is constrained to XML representations. The Web Application Description Language (WADL) (Hadley 2006), a more radical and REST-specific reworking of the WSDL approach, is more flexible in its modelling, supporting, for instance, MIME types other than XML. Another scheme is hRESTS (Gomadam and Sheth 2008; Kopecky, Gomadam, and Vitvar 2008), which embeds descriptions as a 'microformat' in HTML pages. Microformats are data formats that are often (nearly) isomorphic to already

widely adopted formats such as vCard and iCal and that can be easily embedded in existent HTML pages. hRESTS is still at an early stage of development, but serves as a starting point for the semantic efforts of SA-REST and MicroWSMO, discussed in section 3.5.3.

3.4 Grid and workflow systems

Perhaps the least heralded but most used class of matchmakers is that found in scientific workflow systems, and in business process execution using the Business Process Execution Language for Web Services (BPEL4WS) (Andrews et al. 2003). Workflow systems are typically focused on solving problems for real users, and do not have attempt intelligent, automatic service selection. They do, however, have to manage service registration in some fashion, and do so in the wild, for working scientists using the tools as a means, not an end. In this section, we review some of the more prominent of these systems.

3.4.1 Grid

Grid computing—an analogy to the distribution grids for utilities for electricity or gas—is an approach to providing dynamic 'virtual organisations' which can share computational and database resources, on demand and at a global scale (Goble and Roure 2002). (Foster, Kesselman, and Tuecke 2001) defines a Grid as being for "flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources—what we refer to as virtual organisations".

Grids are currently used principally for high-end scientific endeavours, such as the Network for Earthquake Engineering Simulation (NEESgrid[6]), Grid Physics Network[7], and the EU DataGrid Project[8].

[6] http://www.nessgrid.org/
[7] http://www.griphyn.org/
[8] http://www.eu-datagrid.org/

The Grid initially defined its own means for invoking services, but is now converging on web services standards, as part of the Open Grid Services Architecture[9] (Foster et al. 2002). As with web services themselves, the aim is to move the Grid away from a kind of ARPAnet era level of stove-piped answers such as LDAP and GRIDFTP to an open and reusable set of web services. Grid services can contain state, which differentiates them from services, even when they have identical implementations (for example, they may all be running the version of a bioinformatics algorithm, but have different databases or parameter setups). This is in contrast to most web services, which at least try to affect a stance of statelessness.

Triana (Churches et al. 2006) offers an abstraction of services, and can call out to web services. For managing those web services, it uses UDDI. Another prominent Grid workflow system is Kepler (Altintas et al. 2004), which also uses UDDI, and incorporates a harvester component for finding and storing WSDL descriptions.

Two interesting papers by Raman, Livny, and Solomon emerged from the Condor workflow system (Frey et al. 2001). A Grid-inspired notion of service capability and matchmaking is introduced in (Raman, Livny, and Solomon 1998), where services are advertised not on the kind of software service they provide, but the kind of hardware: available memory, CPU type and power, scheduling availability, and so on. The adverts are semi-structured database records. This is a strong fit to the Grid, where computational power is considered the primary resource. In (Raman, Livny, and Solomon 2000), the same authors introduce the notion of 'gang matching': matching several inter-dependent resources. They give an example of a client requesting use of particular software on a machine, where the software requires a license, which itself can be obtained from a set of license resources but which have their own dependencies in terms of which machine they will run on.

In more of an AI vein, reinforcement learning has been applied to improve resource allocation in the Grid (Galstyan, Czajkowski, and Lerman 2004). This work modelled

[9] `http://www.globus.org/ogsa/`

individual clients using a simple reinforcement learning technique to gauge to which Grid resources (computers) they should submit jobs. The reward function was time-to-completion of a job at a given resource, and individual agents learned which resources would provide them the quickest turn around. Resources were homogeneous in their ability to execute a job, but varied in the time taken to do so and the scheduling policy they used locally.

3.4.2 myGrid

The myGrid project[10] (Goble, Wroe, and Stevens 2003) has established informatics tools enabling bioinformatics scientists to conduct 'in silico' experiments using databases of genetic and proteinomic information, and computational services. The toolkit comprises three main elements: Soaplab, which presents legacy tools as web services; Taverna Workbench, which enables the graphical design of scientific workflows which are represented in the Simple Conceptual Unified Flow Language (SCUFL), which can be saved and exchanged with other scientists; and Freefluo, the execution engine which runs the workflow and makes calls to the web services referenced by the flow. Taverna has been widely adopted by the target user community, and is now maintained as part of the Open Middleware Infrastructure Institute UK [11] software suite.

The 'processors' in SCUFL are principally web services, but can also be inline code fragments of Java or R. Taverna's developers invested considerable resources in making many web services available in Taverna: public UDDI registries can be harvested, although the developers find them underpopulated and users see them as heavyweight; myGrid developed its own UDDI registry with additional semantic information; and users can directly add services they know about by pointing at the WSDL. The developers also worked on shim services, to mediate data incompatibilities.

Since Taverna was not aiming at automated composition, most attention focused on

[10] http://www.mygrid.org/
[11] http://www.omii.ac.uk/.

what OWL-S calls the service profile, while the service process and grounding were dealt with via service-specific Java programming. Initially, it was intended to use heavyweight semantics (Wroe et al. 2004), broadly following OWL-S. This approach was eventually rejected on the grounds that it was difficult to create high quality descriptions of services due to the very high precision necessary to avoid false matches, and that users were not benefiting from computationally costly subsumption reasoning. myGrid then moved to using a simpler classification, using only a subset of the features of RDF and OWL.

The lightweight approach is called FETA (Lord et al. 2005). A core domain ontology was built and curated using DAML+OIL and (later OWL), but compiled to RDF-S, and combined with UDDI information, for presentation to the user. Queries can be run at workflow design time to find service types which operate on particular types or perform a particular task. When service selection occurs at execution time, automated selection may take place if only one service is found which matches the required type. Usually, however, the system selects a range of suitable services, using RDF entailment, leaving the final selection to the user. Service fail-over is sometimes performed when services are known to be equivalent, and shim services, which translate between data formats, are automatically inserted, but only when it is known that they will not affect the experiment outcome.

Automated selection is the exception, not the rule, and this suits the users, who tend to distrust fully automatic service selection (Lord et al. 2004). One reason is that services are not completely described: some services provide far more extensive provenance information than others, but don't disclose this in the description. More importantly, the users are scientists with a lot of experience of the services, and they must be able to trust the results of computations. They select services based not only on the service description, but the context of the experiment and their personal experience with particular services.

Based on myGrid, (Miles et al. 2003) extends UDDI to a system the authors call UDDI-M^T. They add semantic metadata to the UDDI TModels using RDF to the UDDI

descriptions, and query it using a graph-based RDF query language. The additional metadata relating to quality of service, cost, semantic description and so on, is stored on the client, leading to the personalisation of service descriptions.

Throughout, the aim has been to create a usable tool, so the Taverna group's practical experiences should be taken seriously. An important part of Taverna's philosophy is reaching out to extant users and services by imposing no demand that resources subscribe to a single ontology. Some other projects, like BioMOBY (Wilkinson and Links 2002) sidestep many issues by using a single ontology. The Taverna project has therefore encountered precisely the kinds of problems with automated service selection we are considering, and has come out in favour of lighter-weight approaches that consider the user.

Finally, we mention Taverna's attitude to workflows, which parallels some of the ideas of LCC and this thesis. In an e-science context, the workflows are the 'method' by which the experiment has been conducted, and should therefore be open for inspection and peer review (Goble, Wroe, and Stevens 2003). Workflows, which may mention the services they use, can now be exchanged through the myExperiment[12] website and software (Roure and Goble 2007). myExperiment is based on the principles of Web 2.0 and social networking sites, and allows users to upload, search, and comment on their scientific workflows. As noted in (Wroe et al. 2004), scientists sometimes want to publish their workflow without revealing the services they used, since the services themselves often hold precisely the data sets which give their owners an advantage.

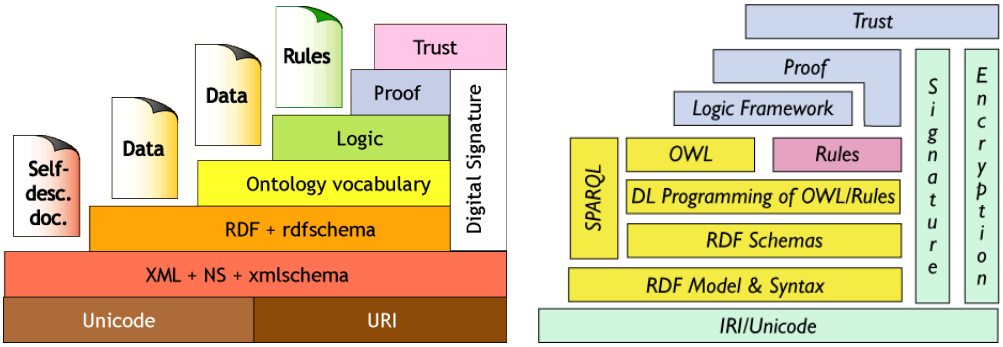
3.5 Semantic web services

Which brings us neatly to the semantic web and the services that will deploy on it. The idea of the semantic web is to extend the current web with formal knowledge representation, to allow machines to interpret the web as information rather than just

[12] http://www.myexperiment.org/

data. The semantic web's architecture is often represented as a layered cake (see figure 3.4), and although this has changed over the years, and been criticised for being unprincipled (Gerber, Merwe, and Barnard 2008), it remains a useful guide. At the bottom lie existing standards such as URI, HTTP, and others. XML is the fundamental syntax, along with XML Namespace, and these serve as a serialisation format for the knowledge representation languages.

Figure 3.4 Semantic web layer cakes old and new



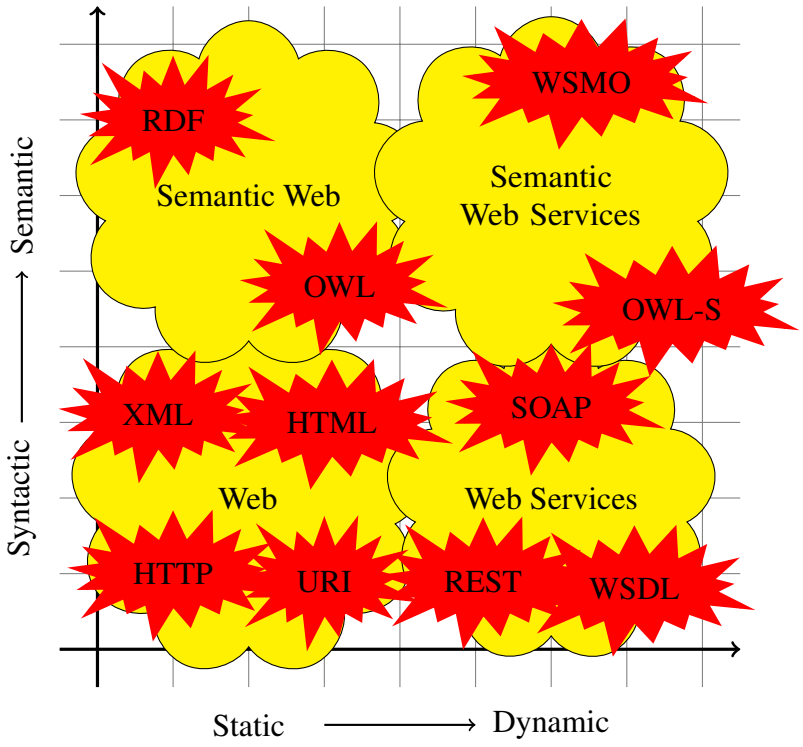
Graphic from W3C.

The Resource Description Framework (RDF) provides a concept-graph model. RDF Schema adds an ability to define classes and very lightweight ontologies. The Web Ontology Language (OWL) (Smith, Welty, and McGuinness 2004; Bechhofer et al. 2004) provides a means for specifying much richer ontologies which can still be partially understood at the RDF level. Semantically, OWL is based on a formalism known as description logic (DL) (Baader et al. 2003), which developed as a formalisation of semantic networks. OWL itself is a development of work on the DARPA Agent Markup Language (DAML) and the European Ontology Inference Layer (OIL) (Fensel et al. 2000; Connolly et al. 2001; Horrocks 2002), programmes. It marries the description logic formalism[13] with a frame-based interface and an RDF/XML serialisation and upward compatibility with RDF. Description logics provide efficient mechanisms for reasoning with categories of objects, keeping the computation tractable by placing various limitations of what can be expressed: the limitations and computational efficiency

[13]The OWL-Full variant adds features which take it outside the DL space.

varying with the exact logic in use.

Figure 3.5 Evolution of the world wide web



The cake described above applies primarily to the 'static' semantic web. The web has also been evolving in its dynamicity, from DHTML and CGI scripts to explicitly service based functionality. Figure 3.5 shows how semantic web services form a conjunction of those two major trends on the web: greater dynamicity, in the form of services, and more semantics. By creating semantic representations of services' purposes and interfaces, the intent is to enable the automatic or semi-automatic discovery, selection, composition, invocation and monitoring of services. The semantic web is unclear with regard to agents or services. They are certainly mentioned in (Berners-Lee, Hendler, and Lassila 2001), but they are not central. The semantic web is neutral with regard to how implementers view their service providers and clients. The software systems on the semantic web are not expected to adopt the 'intentional stance' of agents. Tim Berners-Lee's hypothesis for the success of the original web where more advanced hypertext systems had failed was that it was *simple*. To that end, semantic web services

attempt to piggyback on the already established non-semantic web services standards. Semantic web services are 'normal' web services that have additionally been given a semantic description, often not by the owner or creator of the actual service. In the cake, semantic web services conceptually sit on top of OWL, where we find OWL-S, the main web standard for specifying semantic web services. OWL-S, in addition to being built atop OWL, is also a development of the DAML-Services (DAML-S) work. The other major semantic web services framework, WSMO, sits in the same place conceptually, but uses its own more expressive knowledge representation language, including rules, rather than OWL.

In the rest of this section, we consider the main semantic web services frameworks: OWL-S (section 3.5.1), WSMO (section 3.5.2), and the new lightweight semantic annotation standards (section 3.5.3). One other standard in the works is the Semantic Web Services Framework (Battle et al. 2005), which is similar in approach to WSMO, but we do not cover it here since we are not aware of any matchmaking work based on it.

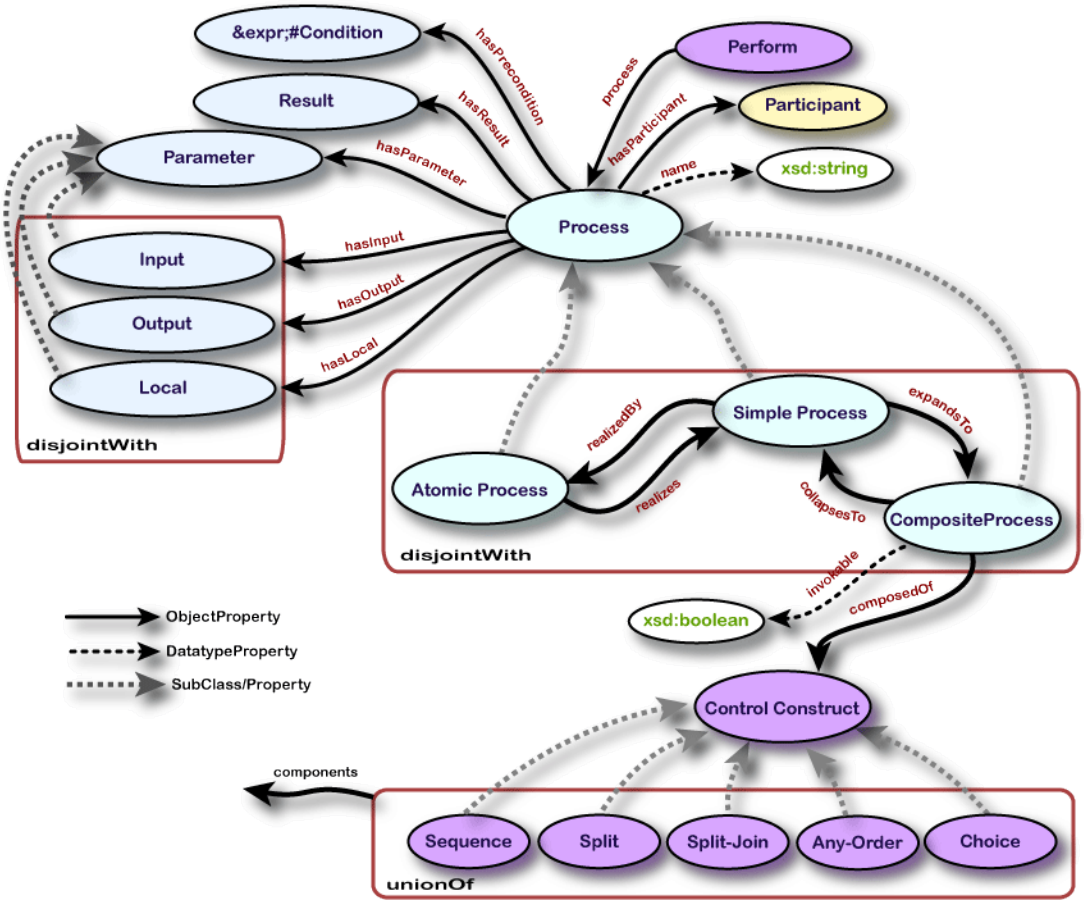
3.5.1 OWL-S

The first major, and still the best known and most widely used semantic web services model, is OWL-S. OWL-S is standardised as a W3C member submission (Martin et al. 2004).

OWL-S is an OWL ontology for describing web services. A service is specified by three models: the 'service profile', 'service model', and 'service grounding'. The profile is effectively the service's capability advertisement and specifies who provides the service and what the service does. This is done by identifying the inputs and outputs of the service concepts by reference to some domain-specific ontology external to OWL-S. The process model describes how the service operates, by means of atomic processes and a workflow-like language to combine those processes into composite ones. Figure 3.6 shows the key concepts of the process model, including notions of atomic, simple and composite services, and the flow control operators. Atomic services

map directly to a single real web service invocation, while composite services represent a workflow involving multiple service invocations. Simple web services are either alternative 'wrappers' around atomic services, or a simplified, single-step abstraction of a composite service. The grounding describes how to map between the semantic level description and the serialised messages in which the services communicate.

Figure 3.6 OWL-S process model



Graphic from W3C.

A service profile describes a service principally in terms of the transformation between inputs and outputs, and the change of state implied by the pre- and post-conditions. Collectively, these inputs, outputs, pre-conditions and effects are labelled 'IOPEs'. Additional information about the service can be supplied:

- An indication of the kind of task it performs: does it search libraries for books, or

book airline flights?

- Information about the provider such as business contact details.
- Non-functional properties about the service, especially quality of service.

While the use of OWL concepts to type inputs and outputs is straightforward, preconditions and effects do not have a native DL representation. Instead, they are embedded as string or XML literals (`expressionBody` in the code below), and labelled with some representative URI naming the formalism used (`expressionLanguage`). Because these expressions are outside OWL, use of them has so far been limited. None of the matchmakers reviewed here make use of them.

```
<owl:Class rdf:ID="Expression">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#expressionLanguage"/>
      <owl:cardinality>1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#expressionBody"/>
      <owl:cardinality>1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The service profile is created separately from the process model. Since it is the process model that actually defines how a service is used, the two views should align closely, but this is not enforced. The OWL-S standard suggests that the profile should expose a subset of the IOPEs. The process model itself could be used to further decide if a service met a user's requirements (cf (Martin et al. 2004)), but to our knowledge no concrete proposals have been advanced.

OWL-S makes no specific provision for matchmaking: service discovery and matchmaker querying, while enabled by the ontology's design, are not defined. There are several OWL-S matchmakers which have been implemented, and we look at them now.

Semantic MatchMaker

Probably the the best known OWL-S matchmaker is Semantic MatchMaker (Sycara et al. 2003*b*). The work has three main threads: dealing with service registration, service selection, and extending OWL-S to handling brokering. Semantic MatchMaker (hereafter SMM) evolved from the RETSINA/LARKS matchmaker described in section 3.2. In contrast to the original MatchMaker, which used an involved, multi-stage algorithm to match client requests to providers, SMM uses subsumption only. The matchmaking happens over the output types of the requested service R and advertised service A. Each output type in the request (out_R) is compared with a corresponding output from out_A, and a subsumes relationship is determined between them. The subsumption match specifies four types of match between OWL classes R (requested) and A (advertised):

- *exact* if $out_R = out_A$ or $out_R \; subClassOf \; out_A$
- *plugin* if $out_R \sqsubset out_A$
- *subsumes* if $out_A \sqsubset out_R$
- *fail* if there is no subsumption relation

If the scoring over the outputs results in several candidate services having the same score, then a similar process is applied to the inputs to break the tie. One might wonder about the distinction between $subClassOf$ and the subsumes relation $\sqsubset$. In this matchmaker, $subClassOf$ refers to direct subclasses only, while subsumes includes those, and any subclasses further down the hierarchy. The idea is to give greater weight to very near misses, but how well this might work in practise is uncertain, since very rich ontologies may have many levels of subclasses while others might have sparse ones. The match classes are assigned various numerical values—$exact$ matches are preferred to $plugin$, which are in turn preferred to $subsumes$—and these numerical scores are used to calculate the total score for each requester-provider match.

The SMM matchmaking work also led to two other related developments. First, the use of UDDI repositories to store the OWL-S description (Paolucci et al. 2002*a*). OWL-S descriptions are stored in user-defined fields in the TModel ('technical model') of the UDDI record. Semantic matchmakers can then query UDDI registries to recover it. Secondly, OWL-S was extended to support brokering. Standard OWL-S provides no means to change the process during execution, nor to replace it. To resolve this 'broker paradox', Semantic MatchMaker introduced an 'exec' primitive to OWL-S to indicate that a new, brokered, process should be substituted in place of the executing one which negotiated the matchmaking.

Li and Horrocks

(Li and Horrocks 2004) discusses a matchmaker for DAML-S (The DAML Services Coalition 2003) (it is based on DAML-S 0.6), the precursor to OWL-S. The model is very similar to that in (Sycara et al. 2003*b*), in that levels of subsumption between request and advert determine the matching. The grades of matching have been extended to the following classes:

$$\begin{aligned} \textit{exact} \quad & \text{if} \quad A \equiv R \\ \textit{plugin} \quad & \text{if} \quad R \sqsubseteq A \\ \textit{subsume} \quad & \text{if} \quad A \sqsubseteq R \\ \textit{intersects} \quad & \text{if} \quad \neg(A \sqcap R \sqsubseteq \bot) \\ \textit{disjoint} \quad & \text{if} \quad A \sqcap R \sqsubseteq \bot \end{aligned}$$

Note the new 'intersects' match, which indicates some degree of overlap between advert and request, and the 'fail' match is renamed 'disjoint' here. A request matches an advert if the query q and the advertisement a are compatible:

$$matches(q) = \{a \in A | compatible(a, q)\}$$

In turn, two concepts are compatible if their intersection is satisfiable (non-empty): $satisfiable(C_1, C_2) \Leftrightarrow \neg(C_1 \sqcap C_2 \sqsubseteq \bot)$. The algorithm (figure 3.7) checks the query

every available advertisement, and offers the client the resulting (ranked) list of suitable advertisements.

Figure 3.7 Service subsumption algorithm from (Li and Horrocks 2004)

DOMATCH($request$,$advert\text{-}database$)

1 **for** $advert \in advert\text{-}database$

2 **do**

3 $input\text{-}match \leftarrow match\text{-}degree(request_{inputs}, advert(inputs))$

4 $output\text{-}match \leftarrow match\text{-}degree(outputs(request), outputs(advert))$

5 $match\text{-}degrees[advert] \leftarrow$ MIN$(input\text{-}match, output\text{-}match)$

6 **return** $sort(match\text{-}degrees)$

Others

The first mention in the literature of using subsumption in semantic web DLs to drive matchmaking appears to be (Trastour, Bartolini, and Gonzalez-Castillo 2001). Using the DAML+OIL precursor to OWL, and without using any particular services ontology, the authors again developed a matchmaker using the exact/plugin/subsume/intersect taxonomy.

In (Sirin, Parsia, and Hendler 2004), interactive workflow composition is assisted by filtering of available services according to context. As a user constructs a workflow, the types of the various data in play are used to determine which services use them as input or output, and thus to present the user with appropriate services. Further filtering, such as that based on service location, can be enabled selectively.

A problem with the service profile which has been often been raised is that the types are not enough to specify the operation. For example, if we have two services S_1 and S_2 which both take as input p : Person and return a date d : Date, we cannot distinguish them by signature, but we certainly cannot assume they perform the same task. While S_1

could return the date of birth of p, S_2 might return the wedding anniversary. The solution proposed in (Hull et al. 2006), for stateless services, is to use what are effectively post-conditions to the inputs and outputs. In the case of the services above, suitable post-conditions are $\mathsf{BornOn}(p, d)$ for S_1 and $\exists m.\mathsf{Marriage}(m) \wedge \mathsf{MarriedIn}(p, m) \wedge \mathsf{Date}(m, d)$ for S_2. The inputs, outputs, and post-conditions are then used to compile a conjunctive query for the DL reasoner which is known to be decidable for simple queries, but raises issues in the case of, for example, transitive roles.

3.5.2 Web Services Modelling Ontology

Another major semantic web services effort is the Web Services Modelling Ontology (WSMO) (Fensel et al. 2006). WSMO originally appeared in the short-lived Web Services Modelling Framework (WSMF) (Fensel and Bussler 2002). WSMF was heavily based on the knowledge modelling work done previously by several of the participants, in particular on the UPML framework (Fensel et al. 1999).

The Web Services Modelling Ontology is simultaneously the term for the overarching approach, and the name of a top level ontology which describes the key elements in it. WSMO uses the Web Services Modelling Language (WSML), and has a reference implementation in the Web Service Execution Environment (WSMX). While WSML does have an XML serialisation, it is not based on RDF, nor does it use OWL or description logic. Instead, it is based on F-Logic (Kifer and Lausen 1989; Kifer, Lausen, and Wu 1995). In WSML, as with SWSL, rules are deemed a necessary support for service description.

The approach's philosophy is made explicit in the ontology defined by WSMO: ontologies, goals, mediators, and services.

- 'Ontologies' are domain ontologies, much as in OWL-S, and describe the arguments to goals and services.
- 'Goals' define tasks users may want to achieve. Goals are means of expressing

requester intent, distinct from services—OWL-S does not distinguish the two concepts.

- 'Web services' are concrete software implementations, expressed as web services, along with their semantic descriptions, including choreography and grounding.
- 'Mediators' are software components which attempt to bridge incompatibilities between ontologies, process models, as well as the goals and the services which implement them.

Goals have preconditions, post-conditions, assumptions, and effects. Preconditions and assumptions specify what must be true before invocation, post-conditions and assumptions what should be true afterwards. While pre- and post-conditions can and should be checked by participants at invocation time (by examining the parameters and results of the goal), assumptions and effects are world states which may not be verifiable, especially not directly by the invokers. For instance, in purchasing an item, one can confirm that a credit card number is supplied (precondition), but not that the account has been debited correctly (effect). Web services have choreographies which detail how to interface with them (inputs and outputs), and orchestrations (essentially workflows composing other services). Because WSMO places mediators between goals and services, their effects may need be taken account of when selecting suitable services.

The separation of goals from services and the importance attached to mediation manifest WSMO's notion of 'strong decoupling'. By making the individual services and goals perfectly decoupled, the hope is that their interoperability will be easier to achieve.

WSMX

The Web Services Execution Environment (WSMX) is a reference implementation of the WSMO approach. The WSMX architecture document (Zaremba et al. 2004) states that the architecture itself does not define a matchmaking algorithm. It defines only an interface, whose Java signature is:

```
List <WebService> discover(Goal goal)
```

This returns a list of web services matching the client's `goal`.

A WSMO deliverable (Keller et al. 2004) contains a high-level review of service discovery and matchmaking, and revisits the 'exact'/'plugin'/'subsumes'/'intersects'/'no match' view, this time from a more abstract set-based view of service descriptions.

The built in WSMX matchmaker uses simple string comparison of requested goal and advertised goal achievement. This offers only the most basic level of matchmaking, being unable to deal with approximate matches. For a Semantic Web Services Challenge event a new matcher was developed which could invoke a special 'contracting' method of suitable services to discover more information about them which was used to rank the matches for the original request (Zaremba et al. 2006). A peer to peer model of service discovery in WSMX is presented in (Toma et al. 2005), but offers little technical detail of the approach.

The Internet Reasoning Service

The Internet Reasoning Service (IRS) is a semantic broker in the WSMO camp. The development of the semnatic web services field itself is revealed in the IRS's own history: IRS-I used the UPML model (Fensel et al. 1999) to support knowledge-intensive systems, and CORBA to implement the services. IRS-II replaced CORBA with web services implemented using SOAP, and IRS-III moved from UPML to the explicitly 'semantic web services' model of WSMO.

Orchestration is dealt with by a workflow-like language executed within the IRS, inside notional 'mediators'. Choreography has been accomplished using the Cashew language (Norton and Pedrinaci 2006). Its matchmaking is based on explicit linkage of services to goals, using mediators. Service selection is done by first finding all the services that have attached themselves to the required goal, and then computing, using pre- and post-conditions, which are qualified to fulfil the particular goal. Approximate matchmaking is avoided.

Caching queries

Subsumption reasoning can be expensive, and for large scale matchmaking good performance becomes important. One proposal is to cache results, in a process called 'Semantic Discovery Caching' (SDC) (Stollberg, Hepp, and Hoffmann 2007). Once again, this builds on the notions of matches being *exact*, *subsume*, *plugin*, and *intersect*. SDC builds a graph of goal templates (that is, the uninstantiated goal classes) using a subsumes relation. Each node of the graph is a goal, and a goal G_j has a parent G_i iff $G_i \sqsupset G_j$. For each goal in the graph, the compatible web services W are found and stored in the node. Using this graph, queries can be answered by finding the query goal in the graph, and using the rules below to determine the services compatible with it:

$$
\begin{aligned}
exact(G_i, W) &\Rightarrow plugin(G_j, W) \\
plugin(G_i, W) &\Rightarrow plugin(G_j, W) \\
subsume(G_i, W) &\Rightarrow exact(G_j, W) \vee plugin(G_j, W) \vee subsume(G_j, W) \\
&\quad \vee intersect(G_j, W) \vee disjoint(G_j, W) \\
intersect(G_i, W) &\Rightarrow plugin(G_j, W) \vee intersect(G_j, W) \vee disjoint(G_j, W) \\
disjoint(G_i, W) &\Rightarrow disjoint(G_j, W)
\end{aligned}
$$

This is quicker than comparing the query goal with every available service individually for every query. The paper claims a near constant query response time as number of services increases, compared with a linear increase for the normal, uncached mechanism.

3.5.3 Lightweight semantic annotations

While there is widespread recognition of the insufficiency of purely syntactic definitions like WSDL, there is neither agreement on how the additional semantics should be provided (hence the competing works-in-progress of OWL-S, WSMO, SWSL, and other projects), nor confidence that average service developers will be able to comprehend and

engineer the heavyweight semantics required by the semantic service frameworks. In response, several groups have begun development of standards for annotating syntactic descriptions with the simpler elements of semantic approaches.

The METEOR research group, along with IBM, introduced the WSDL-S (Akkirau et al. 2005) approach. This offers a set of XML attributes with which to annotate WSDL files. This developed into the W3C recommendation SA-WSDL—Semantic Annotations for WSDL and XML Schema (Farrell and Lausen 2007). SA-WSDL offers two extensions to WSDL, in the form of XML attributes which may decorate WSDL or XSD elements. The first attribute, `modelReference`, indicates that the object identified by the XML element is described in some way by the ontological object pointed to by the attribute's URL value. In particular, SOAP operations, messages, and faults can be annotated. The other extension is the attribute pair `liftingSchemaMapping` and `loweringSchemaMapping`. These are intended to link XSD elements constituting the messages to some transformation mechanism that can translate between them and the ontological object suggested by the `modelReference`.

SA-WSDL deliberately does not specify the content of either the `modelReference` or mapping schemas, this being left to particular semantic web service frameworks and implementations to decide. However, the expectation is that both kinds of annotations would be dereferencable links to, say, OWL concepts, and the schema mappings implemented as XSLT.

All the recent semantic web services standards have targeted the WS-* standards. While popular, they face growing competition from the RESTful approach. Section 3.3.3 covered the non-semantic approaches to RESTful description, and two current approaches build on those by adding semantics. MicroWSMO (Kopecký et al. 2008) is a semantic extension to hRESTS (Kopecky, Gomadam, and Vitvar 2008). In essence, it reapplies SA-WSDL in hRESTS, using WSMO-Lite (described below) as its semantic service model. The SA-WSDL attributes of `modelReference`, `loweringSchemaMapping`, and `liftingSchemaMapping` are rendered as HTML `rel` attributes with names `model`,

`lifting`, and `lowering`. SA-REST[14] is similar, but opts to use RDFa instead of a microformat to embed the annotations.

Finally, WSMO-Lite (Vitvar, Kopecký, and Fensel 2008), simplifies the semantic modelling of services. Addressing arguments that current modelling languages are too heavyweight, it strips down services to a simpler set of notions. Rather than an OWL-type ontology, an RDFS vocabulary is provided for service representation. WSMO-Lite is derived from WSMO, but does away with features like choreography, and the notion of goals.

All of these formalisms are recent, and no implemented matchmakers exist for them.

3.6 Our nearest neighbours

In this section, we single out three recent matchmakers which take novel approaches to their task. What differentiates them from others considered in this chapter is either consideration of user feedback or the use of information retrieval techniques even in the presence of semantics, both of which we advocate in this thesis.

Zhang and Zhang

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 (Zhang and Zhang 2002), although that, again, only examines the case of two-party interactions. Because the approach of Zhang and Zhang is so similar to our own, we present a more thorough review and a comparison later, in section 6.4. Briefly, their approach is to record a success metric for each matchmade service invocation. These are recorded and then collated at query time to find those agents with the best history.

[14] `http://knoesis.wright.edu/research/srl/standards/sa-rest/`

Luan

Another attempt to introduce performance-based selection of web services to matchmakers is found in the work of Luan (Luan 2004; Luan 2004). It, too, considers the performance history of a service, based on user feedback. It also attempts to discover areas of expertise and weakness within services' advertised capability. For each service, a record is kept of cumulative user satisfaction, by service request. For instance, a vehicle manufacturer may advertise itself as producing vehicles, but be considerably better at producing Cars than Vans. Luan's matchmaker records the satisfaction ratings for requests of Van purchasing services separately from Car requests, thus allowing it to determine expertise within an advertised capability. It does this with a simple mechanism of averaging customer satisfaction for a service, given the request.

However, it appears that the service classification is treated as a taxonomy. In OWL-S, as in the description logics it is built upon, concepts are not limited to a fixed set of named classes in a taxonomy, but can be dynamically defined through combinations of concrete concepts and logical operations. While a taxonomy may define PetrolCar and ElectricCar, the notion of a hybrid vehicle may be introduced by a user as $\mathsf{PetrolCar} \sqcap \mathsf{ElectricCar}$. Or, alternatively, a car might be defined to have an Engine, in which case a hybrid would be

$$\mathsf{Car} \sqcap \mathsf{hasEngine}\,.\,\mathsf{PetrolEngine} \sqcap \mathsf{hasEngine}\,.\,\mathsf{ElectricEngine}$$

These anonymous concepts, constructed on-the-fly, are common in OWL-S systems (see section 3.5.1), but do not seem to be dealt with in this matchmaker as it stands. Luan also notes that the matchmaker is very memory intensive.

MX

A recent twist has been the return of keyword matching in an ostensibly description-logic based matchmaker: MX. The MX matchmaker is available in both OWL-S (Klusch, Fries, and Khalid 2005) and WSMO (Kaufer and Klusch 2006) flavours. The principle

behind it is to use keyword matching as a fall back when the subsumption matching fails, an approach the authors call 'hybrid'. But they also show experimental results which appear to place description logic-only matchmakers at a disadvantage not only with their hybrid approach, but also to a pure keyboard-based method.

The starting point for MX's DL matching is a tweaked version of the standard 'exact', 'plugin', 'subsumes', 'fail' from (Sycara et al. 2003*b*). In the event of a failure to find an exact, plugin or subsume match, and before reporting outright failure, MX resorts to a keyword vector similarity measure like that found in information retrieval systems. The paper mentions several metrics suitable for computing the inter-service similarities—extended Jacquard, loss of information, Jenson-Shannon information divergence, and plain TF-IDF weighted cosine—which each behave slightly differently, but the simple cosine metric is sufficient to demonstrate the key idea. The cosine metric is defined in (Klusch, Fries, and Khalid 2005) between a request R and a potential service provider S:

$$Sim_{cos}(R,S) = \frac{\vec{R} \cdot \vec{S}}{||\vec{R}||_2^2 \cdot ||\vec{S}||_2^2}$$

with the expected definitions of $\vec{R} \cdot \vec{S} = \sum_{i=1}^{n} w_{i,R} \times w_{i,S}$ and $||\vec{V}||_2 = \sum_{i=1}^{n} w_i^2 x$. The term w_i is the weight of the ith term. A vector $\vec{R}$ represents the requested service, and $\vec{S}$ a service.

The terms in the vectors are the result of 'unfolding' the description logic. We illustrate with an example. Assume a domain ontology as shown in figure 3.8, and a car rental service S with input Driver and output MediumCar. If a user requests a service with input Driver and output SmallCar, the usual subsumption reasoning would fail since the two concepts are disjoint: $\mathsf{SmallCar} \sqcap \mathsf{MediumCar} = \bot$. MX 'unfolds' the terms to their primitive components (denoted C^P):

$$\begin{aligned}
unfold(\mathsf{Driver}) &= and(\mathsf{Driver}^P, \mathsf{Person}^P, \mathsf{Thing}^P) \\
unfold(\mathsf{SmallCar}) &= and(\mathsf{SmallCar}^P, \mathsf{Car}^P, \mathsf{Vehicle}^P, \mathsf{Thing}^P) \\
unfold(\mathsf{MediumCar}) &= and(\mathsf{MediumCar}^P, \mathsf{Car}^P, \mathsf{Vehicle}^P, \mathsf{Thing}^P)
\end{aligned}$$

In this example, we will map the primitive concepts to dimensions in vector space thus:

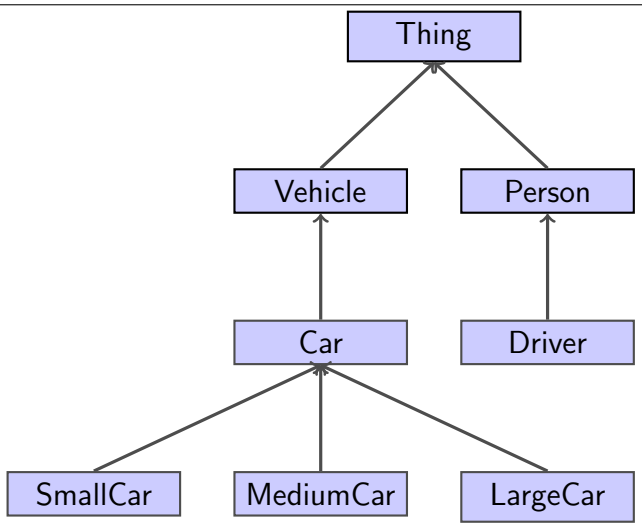
$$\begin{aligned} \mathsf{Thing}^P &= 1 \\ \mathsf{Person}^P &= 2 \\ \mathsf{Driver}^P &= 3 \\ \mathsf{Car}^P &= 4 \\ \mathsf{SmallCar}^P &= 5 \\ \mathsf{MediumCar}^P &= 6 \end{aligned}$$

Which means we can finally consider the vector representations of the classes:

$$\begin{aligned} vec(\mathsf{Driver}) &= (1,1,1,0,0,0)^T \\ vec(\mathsf{SmallCar}) &= (1,0,0,1,1,0)^T \\ vec(\mathsf{MediumCar}) &= (1,0,0,1,0,1)^T \end{aligned}$$

In this scenario, the unfolded MediumCar and SmallCar, when treated as vectors, are judged as sufficiently similar to merit a match, giving the client a service which, while logically invalid, is really quite close to their request and possibly good enough.

Figure 3.8 Domain ontology for MX example



The recall, precision, and time costs reported in (Klusch, Fries, and Khalid 2005) show significant improvements over logic-only approaches:

- The best performing similarity measure, cosine, outperformed the pure logic selection on recall and precision.
- Hybrid matchmaking, using any of the similarity measures, outperformed pure logic selection.
- The extra cost of the similarity computation is around a fifth of the corresponding DL reasoning (1200ms per query for hybrid matching against 1000ms for DL-only, with 350 adverts).

3.7 Summary

We reviewed a number of approaches to service matchmaking, covering both general frameworks and specific matchmakers. If one thing is clear, it is that the best way to find services is still undecided. The matchmakers surveyed used a number of techniques, which together cover a design space which varies in terms of its precision and recall. Some systems require an exact match between request and advert (e.g. bare KQML and the Open Agent Architecture), but essentially guarantee the the correct service is found *if* the various agents have agreed on a shared ontology and precisely implement the same semantics. Other solutions, like COINS's free-text keyword search favour a high recall at the cost of precision. In between, approaches like subsumption try to use extra structure in the service description and query to achieve a balance between recall and precision. Key points are:

- Matchmaking has been a central feature of distributed AI, multi-agent systems, and now web services.
- In most such systems, the mechanism has been the same: services providers lodge service capability descriptions or adverts with a matchmaker, and service requesters query the matchmaker for appropriate services.

- Many schemes for matching query and advert have been tried. Simple string matching, unification of logic formulae, use of nearest neighbour or vector space similarity and term frequency, and most recently, subsumption in description logics.

- Matchmaking has historically been considered as a task of finding one service provider type at a time, not of finding a set of providers that are compatible for the whole workflow.

- Feedback from the client on the success of the matchmaking process is mostly lacking.

- Most of these systems have been small, fairly homogeneous, and closed. Clients and matchmakers trust that service capability adverts are honest and accurate. With web services and semantic web services, there is little in the way of prior agreement or quality control over individual services.

Chapter 4

Critique

The previous chapters have reviewed matchmaking, where the dominant view is that the right service can be selected based on its description in some constrained variant of first order logic. While this works well in toy examples, we think it likely that in practice, no matter how much effort a broker invests in finding a match, messy reality will ensure that, at least some of the time, those matches will be inadequate. Perhaps a particular service has poor network connectivity, or has subtle but important semantic mismatches not evident in the specifications claimed by requester or provider. Some matches that ought to work, will not. There are three broad reasons for this:

1. Service description languages have hard limits to their expressiveness, and it can be intellectually challenging for developers to achieve good descriptions even within those limits. Moreover, in an open web, many providers will have no ability to provide such descriptions, and others will actively set out to provide misleading ones.

2. Individual services vary in their innate ability: the algorithms they use, the quality of data they have access to, and the competence of their implementation all affect the quality of the service's performance.

3. Particular groups of agents will operate better with each other than with other

groups, due to shared ontological viewpoints, their owners' business strategies, or low-level implementation compatibilities.

The third point in particular is novel with this thesis, since matchmaking has previously been considered as a binary relationship between a client's request and a provider's offer. In workflows that are commonly encountered in Grid or business processes, many agents must collaborate. The *interaction* between the agents becomes important, and must be considered. Some previous work (Zhang and Zhang 2002; Luan 2004), covered in the previous chapter, has investigated the use of past agent performance in selecting agents, but none appears to have examined the problem in multi-agent settings. In the rest of this chapter, we briefly reconsidering what users might demand or expect in the way of matchmaking before advancing arguments supporting each of the three points above.

The intent of this critique is not to savage previous work. The problem of correctly representing services in a formal manner is hard, and we have no new direct attack on it. Indeed, many of the problems we note are not technical at all, but arise from human psychology and economy. Instead, we simply want to point out areas where formal ontological modelling currently falls short with respect to real-world implementation, and thereby motivate our introduction in later chapters of selection based on the experience agents actually have with services.

4.1 Expectations for matchmaking

What do we expect of matchmaking? Because sophisticated matchmakers of the kind considered in chapter 3 have not seen any widespread adoption outside the laboratory, the expectations we as researchers have may still be too idealistic. Real users might put up with a great deal less.

It is far from a foregone conclusion that users will want automated service selection at every possible opportunity. Scientists are one group who do not necessarily want

service selection automated fully, or even partially (Lord et al. 2004). In such cases, the wrong choice of service could invalidate the results of scientific experiments. In other scenarios, users might want to retain control because the cost of failure is too high, because they want more control, or because it is too interesting to leave to someone else or a machine. Many people choose not to buy a package holiday from a human travel agent today. And few people go on holiday often enough that they would bother automating it, or learning how to have a broker manage it. We use the web to find cheap flights or hotels, but are curious about the options. And, as is often the case, we do not know what we want until we see it: the exploration itself helps determine our goal.

Today, many such tasks go unautomated, even though programmers are capable of automating them. There is no reason to believe that non-programmers will do so when and if semantic services make it possible. The tasks that will be automated and left in the hands of machines are the same as those that have been put there before: mundane chores that must be repeated by a single agent, where the agent may be an individual user or an organisational entity. Tasks which are done many times in aggregate, but not by individuals, do not get automated because the overhead is simply too much to bother with.

Users will use automatic selection of services only when the benefit exceeds the costs. This may not happen in, for example, scientific experiments: as the myGrid project discovered, scientists often did not trust automated selection (Wroe et al. 2007). Since automatic service selection is an inherently uncertain operation, users will accept it only if the benefit outweighs the cost. The benefit is that they do not have to spend time selecting the services, the cost, what they incur if the automated selection is worse than they would have achieved themselves. So, matchmaking may well be used only when the user does not really mind a somewhat substandard action, thereby making a sloppy, good-enough approach to selection acceptable.

4.2 Limits of logic

The use of logics to describe service functionalities makes many assumptions about the formalisms, descriptions and services, and the people who construct and use them. The first issue is the expressiveness of the service description language. Second is the marked difficulty mortal software engineers have in using formal methods, which is a close parallel to the use of semantics in web service description. Third, that in practice, developers and users will find it difficult to correctly capture the behaviour of their systems or requirements. And finally, the emergence already of a feeling that semantic web services as construed in OWL-S and WSMO are already too complicated, and that we must 'dumb down' further the already constrained languages we use to describe services. We consider each in turn through the rest of this section.

4.2.1 Inappropriate formalisms

The semantic web (Berners-Lee, Hendler, and Lassila 2001) is founded on the notion of the 'ontology'. The study of ontology belongs to philosophy, where it is viewed as the study of the kinds of entities which exist in the world, and the relationships between them. Formal ontology is the "the systematic, formal, axiomatic development of the logic of all forms and modes of being" (Cocchiarella 1991). The word has been co-opted (or, in the view of some philosophers, corrupted) by informaticians to refer to the more limited practice of creating and using formal domain models, usually written in fragments of first order logic. Perhaps the most often cited definition of ontology in computing is Gruber's "an ontology is an explicit specification of an conceptualisation" (Gruber 1993). Borst modified the definition to highlight the role of ontologies in sharing knowledge: "An ontology is a formal specification of a shared conceptualisation" (Borst 1997). This line of developments ends with the definition in (Studer, Benjamins, and Fensel 1998): "An ontology is a formal, explicit specification of a shared conceptualisation". What do these various qualities mean?

- *formal* It is machine readable, with a precise grammar, and known semantics. This explicitly excludes natural language.
- *shared* There is agreement amongst some community on the meaning, or at least, the community believes there is.
- *explicit* The facts of the ontology are recorded precisely and clearly, and are therefore hopefully unambiguous.
- *conceptualisation* An abstraction or model of a domain of discourse.

The hope invested in ontologies is hard to overstate. One prominent ontologist has gone as far as titling a book *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce* (Fensel 2001). The promise of ontologies is that they enable a computer to 'understand' a domain, when of course they merely provide a domain theory in which a theorem prover can make deductions. The semantic web in particular has been aggressively sold as 'machine understandable' and 'machine processable' (Berners-Lee 1998). The latter claim is nothing more than what has been the case since at least the Jacquard loom, and the former relies on the achievement of strong AI. While ontologies certainly help humans build and share knowledge bases, and enable sophisticated inferences to be made in them, they give no guarantee whatsoever that the machine will do the right thing:

> Current intelligent systems are hard to integrate, maintain, and understand because their knowledge bases have not been truly educated on the topics they are supposed to know about.
>
> (Gil 2005)

Ultimately, for ontologies to be useful, the software that deals with them must do something 'real': to send data of a given type, commit to spending money on real product, or perform robotic surgery. The responsibility for correctly grounding symbols to real-world behaviour still lies with the human ontologists and programmers.

This kind of symbol-based world view can be fragile, and has attracted criticism (Shirky 2005; Shirky 2003). Perhaps the best known alternative is Peter Gärdenfors's

'conceptual spaces' (Gärdenfors 2000), which uses geometrical notions to classify objects. Properties are defined as convex regions in a subspace defined by a number of connected dimensions. For example, 'blue' becomes a region in the colour space defined by the hue, saturation and value spaces. Each class is defined as a region in space, bounded by a convex hull around a prototypical member of the class. A principle benefit of such approaches (Gärdenfors 2004) is the ability to deal with similarity between concepts. This notion of similarity in a vector space has already been seen in several matchmakers covered in chapter 3 (Klusch, Fries, and Khalid 2005; Kaufer and Klusch 2006; Subrahmanian et al. 2000).

To understand how fragile descriptions might be, consider an example service and request from (Li and Horrocks 2004), which describe a computer purchasing task:

$$
\begin{aligned}
Advert1 \equiv\ & ServiceProfile \sqcap \\
& \forall\, providedBy\,.(Actor \sqcap \forall\, hasName\,.\{Georgia\}) \sqcap \\
& \forall\, requestedBy\,.(Actor \sqcap \geq_5 hasCreditLevel) \sqcap \\
& \forall\, item\,.(PC \sqcap \geq_{128} memorySize) \sqcap \\
& \geq_{700} hasUnitPrice \sqcap \leq_{200} hasQuantity \sqcap \\
& \forall\, delivery\,.(Delivery \sqcap \leq_{20030501} date \sqcap \forall\, location\,.\, Manchester)
\end{aligned}
$$

$$
\begin{aligned}
Query1 \equiv\ & ServiceProfile \sqcap \\
& \forall\, providedBy\,.(Actor \sqcap \geq_5 hasCreditLevel) \sqcap \\
& \forall\, item\,.(PC \sqcap \forall\, hasProcessor\,.\, Pentium4 \sqcap \\
& \leq_{700} hasUnitPrice
\end{aligned}
$$

Placing information like provider location and price constraints in the service description itself, rather than in a negotiating process, means that service descriptions will need to be changed frequently, and good matches that otherwise could be made would be missed or relegated to low quality matches due to needlessly constraining statements.

4.2.2 Formalism in software engineering

Software development today remains very much a craft, despite efforts to impose order based on the approaches of traditional engineering and mathematics. In the context of semantic web services, the closest point of comparison is with the method and practice of *formal methods*. Formal methods use mathematical techniques to variously specify, reify, and prove the correctness of software (Wing 1990). There are several well-known such formalisms, including Z (Spivey 1992) and the Vienna Development Method (Bjørner and Jones 1978).

The use of formal methods specifically for specification has shown itself to be often effective, but difficult to apply for most engineers (Larsen, Fitzgerald, and Brookes 1996; Meyer 1985). The problem of fully modelling the domain remains difficult, and is not removed by the use of formality:

> For instance it is interesting to note that several contributions to the Sisyphus-II initiative (Schreiber and Birmingham 1996) report 'discoveries' related to the nature of the domain knowledge and the behaviour of the problem solver, which emerged only after the implementation of the end system. And this (fairly typical) phenomenon emerged in the context of an application, the VT elevator design problem (Yost and Rothenfluh 1996), which has been extensively analysed and reconstructed several times!
>
> (Motta 1999)

(Goodenough and Gerhart 1975) defended testing by pointing out errors in published programs which had set out to demonstrate the superiority of formal development of coding over post-hoc testing. Donald Knuth once quipped: "Beware of bugs in the above code; I have only proved it correct, not tried it". Although formal methods have been used with some success in real-world software production, they tend to be either watered-down, or applied to only the most critical sections of code in safety-critical situations in fields like aviation or the military (Clarke and Wing 1996).

Formal verification has gained more favour in hardware (Brock, Kaufmann, and Moore 1996), where the required behaviour is often easier to understand, specify and reason about than is the case in software, due to its regularity and narrowness. Moreover,

the typical production runs of millions of units can both amortise the cost of the proof, and make the potential costs of replacing fielded units after fault discovery worth the investment. Even here, however, formal methods offer no guarantee, as illustrated by the case of the Viper microprocessor.

Viper (Cullyer 1989) was intended for commercial manufacture and use in safety-critical roles in military and civilian applications. The chip relied heavily on formal methods, in specification (Cullyer 1985) and design. Nonetheless, problems came to light. In the wake of this, a series of machine assisted proofs were conducted independently of the design team (Cohn 1987). Cohn raises several issues, including the resources required (six months of effort to verify just the high-level design: the lower levels were expected to be more demanding), and warns against placing too much faith in proofs about what are only models of the artifact. She ends the report with a lament that "the errors we found in Viper's specification and host machine are apparently not present in the actual chip; hence the manufacturers cannot have used the specification which we have started to verify."

4.2.3 In practice

We can expect that the people who will create the semantic services and their descriptions will be drawn from the ranks of today's web designers and software engineers. However, as an industry, we still have difficulty in adhering to the conceptually simpler standards of the Web. A study by browser vendor Opera (Wilson 2008) has found very few web pages are standards compliant. Half do not carry an XML/HTML DOCTYPE declaration, and overall judged only 4% of the 3.5 million sampled URLs to be valid, as defined by the W3C's online markup validator[1]. Tool support was no panacea: of the eleven cited page production systems, only one produced valid pages at a rate over 5%. Finally, the report noted that under half—47% of XHTML and 49% of HTML—of the pages which advertised their validation status through 'validated by' icons were valid. Either content

[1] `http://validator.w3.org/`

generators are lying, or, more likely, other pressures lead to pages losing their validity over time.

Such problems are sure to loom larger when machines become the predominant interpreters of the information. As Peter Norvig, Google's head of research, argued to Tim Berners-Lee[2], the semantic web will be more difficult to construct properly than the current one:

> What I get a lot is: 'Why are you against the Semantic Web?' I am not against the Semantic Web. But from Google's point of view, there are a few things you need to overcome, incompetence being the first... We deal with millions of Web masters who can't configure a server, can't write HTML. It's hard for them to go to the next step. The second problem is competition. Some commercial providers say, 'I'm the leader. Why should I standardise?' The third problem is one of deception. We deal every day with people who try to rank higher in the results and then try to sell someone Viagra when that's not what they are looking for. With less human oversight with the Semantic Web, we are worried about it being easier to be deceptive.

The prospects for semantic services are unlikely to be any better: "Annotation providers are generally not conversant with the use of such technology and are unlikely to make use of the expressive power of OWL" (Lord et al. 2005). The experience of instructors of tutorials in semantic web services suggests most participants are unprepared for the demands (Stollberg et al. 2006). Stollberg et al. cite a lack of ability in conceptualising ontologies and formalising knowledge, on top of poor general knowledge of AI methods. And this is in a self-selected cohort at the upper end of the ability spectrum, interested in semantics at the research level, and motivated enough to participate in a tutorial. The task of describing these systems is complicated, and requires more training than does achieving similar results—at least in the short-term—in a standard programming language.

Specific problems in describing services will include:

- *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

[2] `http://news.zdnet.com/2100-9588_22-6095705.html`

more common, it becomes increasingly likely that users will be unaware of the full implications of their descriptions.

- *User expectation that the information will not be used by clients or matchmakers.* Even if developers fully understand the logics, they may not invest much effort in describing the services in a way that would help the matchmaker. It is not unreasonable to expect service providers will refrain from supplying this kind of data until they observe a significant portion of the service ecosystem using it, creating a bootstrapping problem.

- *Comprehensive constraints may be 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. The problems of 'qualification' and 'ramification'—detailing all the necessary conditions that must hold for a given action to have its intended effect, and infering the results of an action—are key in describing services, and remain difficult to deal with.

- *Privacy or deception* As in real life, the advertisements provided by service providers need not adhere to 'truth in advertising'. 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 honest or open about her service's foibles.

4.2.4 Tools

It is sometimes claimed that tools will make it easy to wrap existing services. Even in creating web services from legacy software, there are important conceptual mismatches. Extant systems exported via web services often have fine-grained interfaces with complex state, or use a factory pattern which requires the creation of a service object which is then invoked (Vinoski 2002*a*; Vinoski 2002*b*). Internal state may be exposed, along

with exceptions and the like, complicating the interface and requiring the client to clean up after they have finished. Exporting these fine-grained interfaces directly through a tool would make services more tightly coupled than they should be, complicating interaction.

On the semantic side, there are tools for creating semantic descriptions from WSDL, such as WSDL2OWLS (Paolucci et al. 2003*a*). However, by definition, the WSDL does not contain sufficient information to create a semantic description: WSDL2OWLS can extract only operation names (which it maps to OWL-S atomic operations), and OWL equivalents of the XML Schema (Fallside and Walmsley 2004). The XSD schemas, however, often make little sense from a knowledge modelling viewpoint, so the OWL translations do not necessarily make ontological sense (Paolucci et al. 2003*b*).

Moreover, our experience of working with WSDL (for example, in the Living Human Digital Library (Viceconti et al. 2007)), is that WSDL does not usually contain enough information to enable even a human to invoke a service. More information must typically be elicited through communication with the service programmers, or by trial and error invocation. Others have found the same: "In our experience, the key difficulty has been poor documentation of the services, requiring experimental invocation of the service with test data" (Lord et al. 2005).

4.2.5 Semantics Lite

To close out this section on the shortcomings of a formal approach to service description, we examine two parallel movements in the web services and semantic services communities. Both place less emphasis on rich formality, vindicating some of the criticisms here. On the services side, the increasing use of services built with the 'REST' philosophy is weakening SOAP's hegemony. In semantics, several attempts are underway to offer users an easy way in to semantics (Semantic Annotations for WSDL), for REST services (Semantic Annotations for REST), as well as lighter-weight formal languages (MicroWSML).

Semantic web services are layered atop standard web services, an important contrast to previous attempts at intelligent services like KQML. The use of existing functionality means that we can take opportunistic advantage of the basic functionality implemented by others with more prosaic aims. It is important to note, then, the evolving attitude to web service provision. SOAP, WSDL, and UDDI have been surrounded by a thriving ecosystem of other standards which have become known as WS-*. The complexity of this software stack has drawn criticism from many programmers[3]. Despite claimed interoperability, in practice there are significant problems integrating different platforms, and in handling the evolving practices not detailed in the standard, such as the emergence of the the `SOAPAction` HTTP header.

The response of many has been to turn to simpler protocols, like REST and XML-RPC. Both are very simple, do not have large companies pushing complex standards, and do not support description systems like SOAP's WSDL. Proposed description languages for REST like WADL have not been greeted with much interest: the many developers who prefer the simplicity of REST do not see a benefit in machine readable API descriptions that still require a human to read documentation.

The debate is by turns frustrating and entertaining, but is instructive for our argument. It seems vital for web service acceptance that the protocols and formats remain relatively simple, inspectable and hackable: this is why web services gained popularity where CORBA did not, and it seems to be a continuing pressure in the current excitement over REST (Richardson and Ruby 2007). As long as programmers do not feel the 'rigour' and tool support of WSDL offers them tangible benefits, semantic services may remain less interesting to them. Given our contention that even strong, well-engineered ontologies will have semantic slack, the prospect of dealing with services which are more lightly specified suggests even more scope for under-specification of interfaces, and thus a greater role for deciding matchmaking on evidence of how they behave, and not just the description.

[3]The more cynical suggest that the complexity was deliberately fostered by the vendors of tools, in order to force programmers to buy into their tool sets just to cope.

The second thread is directly related to the semantic descriptions. Given the complexity of current semantic web services technologies, and the slow take-up compared with other semantic web technologies like RDF and OWL, some researchers have concluded that the demands of semantics must be lessened to lower the barrier. The best known of these is approaches is SA-WSDL, 'Semantic Annotations for WSDL', which is a W3C recommendation (Farrell and Lausen 2007). SA-WSDL enables WSDL definitions to contain pointers to semantics. Similar approaches include SA-REST and MicroWSMO, discussed in section 3.5.3. Should these mechanisms become popular, semantic services will become more available, but with less precise semantics.

4.3 Services themselves

A new paradigm they may be, but web services are still just plain old software, and the least well understood kind of software at that: distributed. Leslie Lamport, a pioneer in distributed computing, gave an unorthodox definition of a distributed system as "one in which the failure of a computer you didn't even know existed can render your own computer unusable" (Lamport 1987). Today's trend away from semi-rigorous requirements gathering and specification and toward agile development and constant evolution of systems runs counter to the notion of formal specification and prior semantic agreement. In this section, we argue against the black-box exchangability of services, and that the descriptions of services will often not be provided by the engineers or even the organisation which produced the service.

4.3.1 Fungibility

The assumption behind the whole enterprise of matchmaking is that services identified by logically equivalent service descriptions are equivalent in the real world. This belief in the black-box fungibility of Web, Grid, and agent services is a useful abstraction, but we must remember that "the map is not the territory" (Korzybski 1931). When the goal

of fully interchangeable blocks of code still eludes conventional programming after decades of research, there is no reason to believe it will suddenly be true for service oriented computing. While we can hope that the greater decoupling that comes with agent or service oriented design will alleviate the trouble to some extent, the greater openness and dynamism will bring other problems to undermine interoperability. Most matchmakers to date have been designed for closed worlds, where it is assumed that agents are fungible.

In (Zhang and Zhang 2002), the problem of variation between agents in their 'intrinsic ability' to complete a given task is highlighted. In myGrid, bioinformaticians expressed strong views about the particular implementation of a service type (Lord et al. 2005). Similarly, the database that a service operates on may be an influence: although this information is sometimes made explicitly available through registry information or word-of-mouth, it may well be deemed a trade secret by the provider.

A related problem is versioning. This is commonly solved by labelling components with a version number. While never foolproof—requirements tend to be liberal and therefore do not solve the problem, or overly tight, which may prevent small problems at the expense of unnecessarily refusing to run at all—in the Internet age, it is even harder with the mantra of 'release early, release often'. Online software services are often simply unversioned, and continual, unannounced release of agent or web services will become more common. It is already impossible in many cases to tell if a service used today is executing the same code it used yesterday. Conducting a search at Google[4] will use both code and a data set that are unversioned, and may return different results today than the same query would have done yesterday.

4.3.2 One service, many descriptions

There are conceptually at least three actors involved in using a semantic web service: the web service creator, the semantic description creator, and the user. Both the leading

[4] `www.google.com`

semantic web services frameworks, OWL-S and WSMO are predicated on the reuse of pre-existing non-semantic services, and that consequently there can be multiple descriptions of the same underlying service. Each non-semantic service may be described and used in different contexts by way of different mediators or descriptions. The people doing the mapping will in many cases not be the people who wrote the services, and may well be acting totally independently, without even the knowledge of the service providers.

Especially in highly technical domains, there may be too much knowledge, explicit and tacit, for these semantics to line up. (Lord et al. 2004) suggests that formal models of the domain and the services will be inferior to the domain experts' experience. The same paper also notes that, especially in areas like bioinformatics, the underlying concepts themselves are subject to disagreement and evolution.

4.4 Interactions

Most matchmaking work has addressed interactions with only two-parties: the service requester and the provider. This bias continues on the semantic web. OWL-S (Martin et al. 2004), for instance, imagines that any interaction will be principally two-party. FIPA-ACL and OWL-S both make notional provision for multi-party interactions, but support for them is weak. Where multi-party interactions do occur, the matchmakings occur serially. Much of the dialogue theory work in agency derives from earlier linguistic or philosophical work, such as the Walton and Krabbe typology (Walton and Krabbe 1995)), which is mostly concerned with two-party dialogue. One can easily 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 (Kavantzas, Burdett, and Ritzinger 2004) reveal the growing realisation that many real-world processes require multiple participants.

A key driver for the success of social web sites, Google, and the Web in general, is the 'network effect', or Metcalfe's Law (Hendler and Golbeck 2008). This is the

hypothesis that the benefit of additional users is more than linear. In the web, the opportunity for links between content grows at a rate considerably greater than the number of documents: in principle, as the square of the number of documents. One of the principle supposed benefits of web services in general, and semantic web service in particular, is the opportunity for the unanticipated, serendipitous reuse of services in ad-hoc orchestrations. In Web 2.0 these have been termed 'mashups', and the semantic equivalent dubbed 'smashups' (Lathem, Gomadam, and Sheth 2007). With many services, there will be many ways to have them interact: too many for individual service providers to test, or even enumerate.

Hardware and software are notorious for having interactions that "shouldn't happen", and such problems are all the more difficult to trace and repair because they cannot be isolated in a single component. In (Wooldridge and Ciancarini 2000) we find the claim that "interaction is probably the most important single characteristic of complex software", and one that certainly has implications for matchmaking in a world of services. Incompatibility between software is a commonplace. Airbus lost billions of dollars and delayed delivery of its A380 aircraft after inaccurate data translation between the different versions of the computer-aided design software used in different teams meant the electrical cabling did not fit the fuselage (Bartholomew 2007). Ironically, a member of another Airbus CAD team predicted just such a problem (Horwood 2005).

It is an everyday fact that, like any interacting systems, some collections of agents will work better together than with others who are individually as capable. Agents provided by the same organisation, written by the same group of engineers, or sharing some view of the world not denoted in their formal specifications, are likely to produce better results when working together than when teamed with other agents that 'think differently'. In the process of creating this dissertation using LaTeX, I had several paths for producing PostScript output: the direct use of `pslatex`; using `latex` then `dvips`; or `pdflatex` then `pdf2ps`. These should produce the same result. They do not. Similarly, in viewing the final result, different viewers (`xdvi`, `gs` `evince`, or Adobe

Acrobat) perform differently, adding their own flourishes with regard to font choice and resolution, crop marks and missing images. Reasonable people reading manuals, or planners looking at formal descriptions of these tools, would be disappointed by the reality of the orchestrations. What underlying causes are there for these problems that are so difficult to encode? In the remainder of this section, we look at some causes which are particularly relevant for semantic services.

4.4.1 Semantic islands

There will be some level of semantic balkanisation. Tim Berners-Lee talks of the semantic web as a 'fractal mess', and James Hendler of 'semantic webs'. Such problems emerge in several ways: there may be differences in the fundamental logic, in the ontologies, in the implementations, and if mediation is used to circumvent the problem, the mediators themselves may suffer from any of these problems. Even when a community agrees on an ontology, a single set of formal symbols, there is no guarantee they actually agree. There can be misunderstandings. The ontology itself may be known to be a compromise, and incapable of expressing differences agreed by many to exist but lacking support for ontology change.

The logics underlying semantics are not yet a settled affair. Despite efforts at sharing knowledge bases, there are still many forms of logic, reasoners for them, and ontologies written in them (Ginsberg 1991). The reasoners which deal with them are, as with any software, correct only in so far as their human implementers can make them.

> In small examples, it may be easy to check the correctness of a system's reasoning. However, for typical real-world examples, manual checking is not feasible. In these instances, the best (perhaps the only) way to check correctness is often by checking for consistency with the reasoning of other existing systems.
>
> (Gardiner, Horrocks, and Tsarkov 2006)

Such an attempt is the subject of (Pan 2005), where an empirical study is made of three brand-name DL reasoners: Fact++, Pellet, and RACER. The study compared the

reasoners on a set of 135 OWL ontologies culled from real-world usage, considering not only how quickly they classified the ontologies but also qualitative issues. Performance wise, there was little to choose: for most of the problems they reasoned in very similar amounts of time[5]. On a number of ontologies, the reasoners did not agree on the consistency of the ontology or bailed out of the attempt. We reproduce the table in figure 4.1. In the study, classifications taking over one hour were recorded as 'Timed out'.

A similar result appears in (Gardiner, Horrocks, and Tsarkov 2006). This paper attempted to quantify the real-world complexity of DL reasoning, also by automated benchmarking of reasoners, adding KAON2 to the three studied in (Pan 2005). 300 OWL ontologies were collected, of which only 162 could be translated to the DIG DL-reasoner interface (Dickinson 2004). DIG is a popular DL interface and language, predating the semantic web. The paper assumes the OWL-DIG translation is correct. They defined 'complex' description logics as those more complex than $\mathcal{ALC}$[6] Those ontologies which proved most computationally expensive were not necessarily very complex. Though not their primary purpose, they also noted the inconsistencies between systems.

Figure 4.1 OWL reasoners disagree

System	Consistent	Inconsistent	Timed Out	Aborted
Racer	101	7	0	27
Pellet	103	0	17	15
Fact++	121	0	3	11

From the review of OWL reasoners in (Pan 2005).

It is wholly understandable that these reasoners differ. That is entirely our point. While OWL-DL is well defined, with precise semantics, it is simply beyond the state-of-the-art in software engineering to ensure that all but the simplest of algorithms perform

[5] http://www.mindswap.org/2003/pellet/performance.shtml

[6] $\mathcal{ALC}$ is a minimal description logic, supporting: (from $\mathcal{AL}$) atomic concepts (including top and bottom concepts), atomic concept negation ($\neg A$), concept intersection ($C \sqcap D$), value restriction ($\forall R.C$) and limited existential quantification ($\exists R.\top$). $\mathcal{C}$ adds complement of arbitrary roles.

identically across implementations. When even OWL reasoners, with their explicit semantics and implementations by description logic experts, do not agree, there can be little hope for the poor software engineer trying to ensure interoperability between services implemented without formal specifications and between many organisations.

4.4.2 Ontology mapping, mediation, and gateways

It is likely that many services will be provided and accessed via gateways and inter-ontology translation: for example, reusing an OWL-S service in a WSMO system. Services performing screen scraping, interfaces to legacy systems, translations between ontologies or web service frameworks (semantic or otherwise) will all introduce their own semantic impedances. While they will make more information and services available, they will also make the possibility for error greater than would be the case for systems designed explicitly to interact with similar systems.

The processes of ontology mapping or merging offer a means to bridge between knowledge bases constructed using different ontologies. Automated ontology mapping systems rely on various inexact mechanisms, such as string similarity between concepts, finding relationships between those concepts through WordNet (Fellbaum 1998), and inferring concepts from shared instances (Kalfoglou and Schorlemmer 2003), and so are going to introduce incompatibilities. Manual translations are not only expensive, even they cannot merge the unmergable: mapping between an ontology which distinguishes newspapers, magazines, and journals with one which lumps them all as periodicals cannot be done without loss of precision. Moreover, the native ontology of a service—the one in which the system's implementers thought and wrote the code around—may not be declared to the outside world, if it formally exists at all. This makes it impossible to tell conclusively if a shared ontology is really being used at a deep level by the participants, rather than merely as a lingua franca.

Beyond simply mapping between ontologies is the heterogeneity in frameworks and implementation platforms. The semantic services landscape at the moment includes not

just OWL-S and WSMO, but SWSF, MicroWSMO, SA-WSDL, as well as agent frameworks being repurposed, and legacy systems exported through web services. Such systems vary in their patterns of communication (direct peer-to-peer, mediated, broadcast), protocols, communication languages, representation languages and implementation platforms. Moreover, the bridges themselves will take many forms: ad-hoc shim implemented on a per task and per agent basis, gateway agents, and reusable mediation toolkits. Sometimes the mediation will be visible to the users and middle agents, and sometimes it will be hidden.

One of the few case studies of inter-system integration in the literature is that done to bridge the significant and independently developed multi-agent systems of RETSINA (Sycara et al. 2003*a*) and Open Agent Architecture (Cheyer and Martin 2001). In (Giampapa, Paolucci, and Sycara 2000), the approach was to build a portal agent (that they call a 'SuperAgent') to bridge the two agent worlds. They cite the following issues in the translation:

- *Different computational/representational paradigms* KQML is functional, Prolog relational. This has implications for mapping multiple return values (since a Prolog clause can succeed several times with different values), and more taxingly, the issue of input and output parameters. In OAA input and output parameters are not explicitly defined. In addition, KQML performatives must be accounted for. These had to be added by hand.

- *Syntactic differences* This might be rather trivial, but could lead to complication and confusion for humans. In this case, KQML's Lisp-inspired `(predicate :arg1 value1 :arg2 value2)` must be translated to the OAA equivalent `predicate(value1, value2)`, but there is nothing in the description to link KQML's keywords with Prolog's positional ones. If automated inference were used, mistakes could be made.

- *Slow translation of service advertisements*, which although not very important in

their small system (just 50 agents), would become more so in larger web service environments.

Another example of the difficulty of inter-agent mediation appears in (Gil 2005), where the authors note several cases where systems had fundamentally different notions of position and movement. For instance, helicopter flight paths generated by a planning agent had a precision of 9 metres, but the agent responsible for flight would not recognise such a short distance, and would assume the goal achieved, without actually moving the helicopter. They remark that "Differences in modelling methodologies make translation an arbitrarily complex task". They make a case for much richer knowledge modelling that formally captures background information and providence information about the ontology, allowing reasoning about the ontology as well as in it.

4.4.3 Workflow construction

Although we have mostly focused on the incompatibilities of clients, the interactions between clients are directed by the workflows. We saw in the last chapter that even a simple distributed problem could result in a broker offering multiple workflows as solutions. Figure 3.2 showed two different sequences of performatives that converted metric and imperial measurements and resulted in computation of a body mass index. In the first sequence, the two input measurements were first converted to metric before being passed to the dietitian. In the latter, the dietitian works with imperial measurements producing a result that is then converted to the metric equivalent. Even in such a trivial example, there may be cause for choosing one ordering over another. For instance, the converter agent's numerical accuracy might be poor, and we would want to avoid using it as much as possible. Or, most simply, we might want to reduce the number of performatives issued. In more complex examples, entirely different sets of agents and operations might be used to achieve the same end, and the complexity of the interactions quickly mount.

4.4.4 Peopleware

Finally, we observe that not all the problems of interactions are purely technical. In orchestrating services drawn from many providers, we inevitably try to cross unmarked, perhaps unrecognised, boundaries between human groups. These groupings may emerge for several reasons:

- *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* Airlines may offer a special deal with other airlines or car-hire companies that would lead to a more satisfied customer. On a deeper level, companies and their suppliers may have invested considerable effort aligning their technologies, without it being obvious to an external entity.

- *Malice* Lastly, and sadly, it is hardly unknown for some software creators to ensure lock-in by making their software deliberately fail to interact correctly with that of other vendors.

4.5 Summary

In this chapter, we made a case that service selection based only on fragments of first order logic will be difficult to do well in an open setting. This was based on problems engineers have in expressing themselves in formal logics, in the variation in performance amongst services with similar advertised properties, and in the many subtle interactions that appear when components are assembled. We also noted that users may actually only use automated service selection when they really do not mind imprecise matching, thus making attempts at precise matching besides the point. We can conclude that:

- The expectations of matchmaking may be unrealistically high. In practical use, good-enough results might not just be good-enough, they might be the only achievable results. If expectations are lowered, there is greater scope for middle agents to perform more flexible but less accurate matchmaking.

- Software engineers find formal descriptions difficult, and formal methods have seen little mainstream take-up. Even those well trained in such formalisms have encountered failures in projects which relied heavily on them. In reality, much software is developed with no specification at all.

- Formal languages, particularly those used on the semantic web, have limited expressiveness, which makes capturing all the assumptions and effects of a service difficult, if not impossible.

- Services themselves are not fungible. Even when their descriptions are identical, it is highly unlikely that their observable behaviour will be. Vendors will offer services that, while comparable at the level of granularity of a service description, will vary in their quality and precise behaviour. Service providers will have little incentive to be especially truthful in their service advertisements.

- As difficult as it is to create high quality ontologies, particularly those with enough richness to model service behaviour, it is even more difficult to combine such ontologies: semantic agreement does not scale well.

- Services will have non-obvious, hard to phrase, and perhaps even hidden differences that will affect not their intrinsic ability, but their ability to function in collaborations with services with different biases.

Chapter 5

Tools

As we stated in the introduction, this thesis introduces an approach to matchmaking that is reminiscent of search engines. We can conceive of a service orchestration as a first-class object, a specific interaction. That interaction names several kinds of service, each of which must be linked by the matchmaker to a specific agent which advertises itself as capable of fulfilling it. If each interaction is scored by the peer which initiated it, the matchmaker can store this ranking, in much the same way search engines record which web pages contain certain words. Over time, a matchmaker can construct a large database of such interactions, and compute the selections of agents most likely to prove successful for new interactions.

We develop the matchmakers themselves in chapters 6 and 7. In this chapter, we review the two formalisms on which we build our matchmakers: the Lightweight Coordination Calculus and the Incidence Calculus. The Lightweight Coordination Calculus (LCC) is a compact language for specifying distributed agent or service dialogues. In our search engine analogy, LCC interactions are our documents. The Incidence Calculus (IC) is a probabilistic calculus based on the explicit representation of events, and their manipulation via set operations to compute probabilities. IC, then, is comparable to the 'inverse indexes' of information retrieval. The fundamentals of our approach to matchmaking do not rely on either of these two formalisms: they are,

however, both clear and convenient vehicles for expressing them.

5.1 Lightweight Coordination Calculus

The Lightweight Coordination Calculus (LCC) (Robertson 2004) developed from work on 'electronic institutions', which are ways of describing inter-agent dialogues. In multi-agent systems, the agents perform actions in the world either through their own internal means or by communicating with others. These inter-agent dialogues, as sets of illocutionary messages (Austin 1976), change the world by having agents commit themselves to actions which they can directly effect. Thus, the content of these messages and the patterns they form crucially determine the behaviour of the system.

Agent interaction patterns typically fall into one of two camps: very strict, and very unstructured. Strict protocols force agents into simple and inflexible communication paths. They are, however, easy to understand and verify. At the other extreme are very flexible protocols in which agents have few constraints on their behaviour, which makes it difficult for agents to reason about the conversations they are participating in, and to determine if other agents are behaving correctly. Electronic Institutions (EIs) (Esteva et al. 2000; Esteva, Padget, and Sierra 2001; Esteva, Cruz, and Sierra 2002) provide a middle path, mimicking traditional human social institutions by providing agents with roles that they fulfil (such as an auctioneer or buyer), scenes that they participate in (signing up to an auction, bidding, and completing the purchase), and by providing institutional over-seer agents that ensure other agents are behaving correctly according to their role and the obligations they have entered into with others. Special agents known as 'governors' enforce the patterns of exchange.

EIs describe agent interactions in terms of labelled transition graphs, called 'dialogic frameworks'. The potential states of the dialogue are represented in the graph's nodes, and the directed nodes indicate the illocutions which move the agents from one state to another. Agents are permitted only to make the illocutions specified in the graph,

and only when in the correct state. They aid in multi-agent systems by making explicit the communication patters permitted and required of agents, thus constraining the amount of reasoning necessary from the union of the mental states (such as those present in the belief-desire-intention model) of the engaged agents to the just those message exchanges explicitly permitted by the institution. The import of this is that institutions, by constraining interactions to the conventionally allowed paths in a managed way, enable agents to better reason about the behaviour of others.

However, EIs have limitations: the role of an over-seer, for instance, can limit the autonomy and concurrency of the agents, an important characteristic of agent systems. LCC extends EIs by providing for a completely distributed mechanism that still ensures agents adhere to a common protocol. This dichotomy is referred to in (Genesereth and Ketchpel 1994) as 'direct communication' (where agents interact directly with one-another) and 'assisted communication' (where they use middle-agents to mediate the transaction). LCC enables us to take either approach, as each task dictates.

5.1.1 Structure and interpretation of LCC

LCC is based on the π-calculus (Milner 1999), and provides a simple, declarative language for defining inter-agent message passing in peer-to-peer systems. The central notion of LCC is the 'interaction'. An interaction I is a triple

$$I = \langle M, C, K \rangle$$

The kind of interaction is defined by M, the model, which is equivalent to the dialogic framework in the electronic institutions. C are clauses currently 'in flight', that is, in a state of partial evaluation by some of the agents. K is the common knowledge, the facts shared between the participating peers for that interaction. In the framework, an agent is denoted as $a(Role, Id)$, where the Id is some atomic identifier unique to the agent, and $Role$ is some (optionally parameterised) role, or type assumed by the agent within the

interaction. An agent can send messages to other named agents:

$$msg_1 \Rightarrow a(Role, Id)$$

and receive messages

$$msg_2 \Leftarrow a(Role, Id)$$

Messages can be unconditional, as above, or conditional on constraints. A constraint C to be satisfied on receiving a message M is a 'reaction' constraint, and is written

$$C \leftarrow M \Leftarrow a(R, I)$$

A constraint on sending a message is a 'proaction' constraint, written as

$$M \Rightarrow a(R, I) \leftarrow C$$

These message-passing operations are the primitive actions of the calculus. They are tied together with operators for sequencing ($then$) and choice (or). Following the Prolog tradition, terms beginning with an uppercase letter are variables, and other terms are constants. A grammar for the framework is provided in figure 5.1. The rewrite rules for expressing the execution are shown in figure 5.2. Since LCC is a language for coordinating services, not programming them, its comparative lack of general programming constructs is a deliberate and useful property.

Along with the interaction model and the partially expanded clauses, an interaction carries 'common knowledge' (Halpern and Moses 1984). This is a set of expressions, specific to an interaction, that every participant in the dialogue can access and extend. Knowledge is expressed as a set of clauses of the form $knows(Agent, Fact)$, where $Agent$ is the name of an agent or the wildcard ?, standing for all agents, and the $Fact$ is a Prolog-like term ($Term$ in the LCC grammar).

Conceptually, an interaction model is a script for communication, defined in the dialogue framework. This script is passed between the agents as they proceed, with each agent marking in the interaction (on the script) what they have done. It can be compared

Figure 5.1 Grammar for the LCC dialogue framework

$$
\begin{aligned}
Model &::= Clause^{*} \\
Clause &::= Agent :: Def \\
Agent &::= a(Role, Id) \\
Def &::= Agent | Message | Def \ then \ Def | Def \ or \ Def \\
Message &::= M \Rightarrow Agent | M \Rightarrow Agent \leftarrow C | M \Leftarrow Agent | C \leftarrow M \Leftarrow Agent \\
C &::= Term | C \wedge C | C \vee C \\
Type &::= Term \\
Id &::= Constant \\
M &::= Term
\end{aligned}
$$

to actors passing around a single script of a play, marking what they have just said, and passing to the actor whose turn is next.

Figure 5.3 shows a model for discovering new music tracks based on current tastes. We see three roles: $listener$, $classifier$, and $recommender$. The first clause is $a(listener, L)$, which will be adopted by the agent L that initiates the interaction. Because L can satisfy the constraint $favourites(Favourites)$, unifying the variable $Favourites$ with a list of its favoured musical tracks, it can use the rewrite rules def and $proaction$ to send a message to the $music\text{-}classifier$. The partially evaluated copy of the $listener$ role definition is added to interaction state I, and sent with the message. The $music\text{-}classifier$ agent C itself is a variable: its value might be determined by L, or by a matchmaker. C receives the message $m(tracks(Tracks))$, after which, through a rewrite $proaction$, it satisfies the constraint $classify(Tracks, Genres)$ and sends the list of $Genres$ to the $recommender$ agent R. R can then satisfy

$$recommend(Genres, Recommendations)$$

and send the final result $Recommendations$ back to L.

There are several ways of deploying LCC. The most common, and the one used for

Figure 5.2 Standard LCC interaction model rewrite rules.

A rewrite rule

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

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 interaction; O is the message produced by the rewrite (if any).

$$\frac{B \xrightarrow{M_i, M_o, \mathcal{P}, O} E}{A :: B \xrightarrow{M_i, M_o, \mathcal{P}, O} A :: E} \; def$$

$$\frac{A_1 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \quad \neg closed(A_2)}{A_1 \; or \; A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E} \; or_{\;1}$$

$$\frac{A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E \quad \neg closed(A_1)}{A_1 \; or \; A_2 \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, O} E} \; or_{\;2}$$

$$\frac{A_1 \xrightarrow{M_i, M_o, \mathcal{P}, O} E}{A_1 \; then \; A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \; then \; A_2} \; then_{\;1}$$

$$\frac{A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \quad closed(A_1)}{A_1 \; then \; A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} A_1 \; then \; E} \; then_{\;2}$$

$$\frac{(M \Leftarrow A) \in M_i \quad satisfy(C)}{C \leftarrow M \Leftarrow A \xrightarrow{M_i, M_i \setminus \{M \Leftarrow A\}, \mathcal{P}, \emptyset} closed(M \Leftarrow A, \mathcal{C})} \; reaction$$

$$\frac{satisfied(C)}{M \Rightarrow A \leftarrow C \xrightarrow{M_i, M_i, \mathcal{P}, \mathcal{C}, \mathcal{C}', \{M \Rightarrow A\}} closed(M \Rightarrow A, \mathcal{C}')} \; proaction$$

$$\frac{satisfied(C)}{null \leftarrow C \xrightarrow{M_i, M_i, \mathcal{P}, \mathcal{C}, \emptyset} closed(null, \mathcal{C})} \; end$$

$$\frac{clause(\mathcal{P}, C, a(R, I) :: B) \quad satisfied(C)}{a(R, I) \leftarrow C \xrightarrow{M_i, M_o, \mathcal{P}, \mathcal{C}, \emptyset} a(R, I) :: B} \; role$$

Figure 5.3 An LCC interaction model

$$
\begin{aligned}
&a(listener, L) :: \\
&\quad m(tracks(Favourites)) \Rightarrow a(music\text{-}classifier, C) \leftarrow \\
&\qquad favourites(Favourites)\ then \\
&\quad play(Recommendations) \leftarrow \\
&\qquad m(recommended(Recommendations)) \Leftarrow a(recommender, R) \\
\\
&a(classifier, C) :: \\
&\quad m(tracks(Tracks)) \Leftarrow a(listener, L)\ then \\
&\quad m(genres(Genres)) \Rightarrow a(recommender, R) \leftarrow classify(Tracks, Genres) \\
\\
&a(recommender, R) :: \\
&\quad m(genres(Genres)) \Leftarrow a(classifier, C)\ then \\
&\quad m(recommended(Recommendations)) \Rightarrow a(listener, L) \leftarrow \\
&\qquad recommend(Genres, Recommendations)
\end{aligned}
$$

our matchmakers, is to treat the interaction model as a data structure passed between agents who hold no conversation state themselves: the interaction contains everything they need to continue the conversation. Another is to distribute, ahead of the interaction, fragments of the model to individual agents, who can then choreograph their own interactions with those agents they directly interact with. This approach has been used to drive web service orchestrations. It can also be used by a single machine to drive an orchestration similar to workflow execution engines languages like BPEL4WS (Andrews et al. 2003).

5.1.2 Uses of LCC

Due to its lightweight nature, LCC has been used as a test-bed for several pieces of research. In particular, a variant of LCC used with web services is the Multi-Agent Protocol language (MAP). MAP was developed in parallel with the Prolog-based LCC, implemented in Java, and designed for coordination of web services implemented in the WSDL/SOAP framework. The language for specifying the protocols bears a passing resemblance to Perl. The key difference with MAP, however, is in how it is distributed. While an LCC model is normally passed from agent to agent as the dialogue progresses, a MAP protocol is sent out to the agents beforehand. This makes it easier to achieve parallelism, but does restrict the behaviour to static protocols, where the dialogue framework is fixed before execution.

As stated earlier, a major benefit of such protocols is the restriction of agent choice: LCC models constrain the search space that agents must reason in to those specified in the model, and the small number of choice points specified in it through the constraints agents must satisfy (the '←' construct). In LCC (Osman, Robertson, and Walton 2005; Osman and Robertson 2007) (Walton 2004), Osman and Walton have applied model checking (Clarke, Grumberg, and Peled 1999) techniques to dialogue frameworks. This makes possible verification, at run-time, of statements about the dialogue, such as that particular deontic commitments are upheld. This can be used to guarantee, for example,

that an agent's trust requirements are not violated when it participates in an interaction.

LCC lacks a direct counterpart to Electronic Institutions' notion of scene. Since the scene provides a useful engineering construct from which dialogues can be composed, work has been done (Joseph et al. 2007) to reintroduce them based on the concept of mobile ambients found in extensions of the π-calculus (Cardelli and Gordon 1998). One use of LCC was to investigate computationally 'dialogue games', which were previously a philosophical enquiry (McGinnis 2006). LCC has been used to directly implement the kind of workflow processes used to tie together web services. In (Guo, Robertson, and Chen-Burger 2005), an LCC protocol is developed which directly interprets the Business Process Execution Language for Web Services (BPEL4WS) (Andrews et al. 2003), enabling such work flows to be executed in a peer-to-peer fashion. It has also been used to study on-demand ontology mapping between peers in a context-sensitive way (Besana and Robertson 2007), and to execute bioinformatics experiments (Quan et al. 2007). Finally, LCC forms the basis for the OpenKnowledge project[1] (Robertson et al. 2007).

5.2 Incidence calculus

The incidence calculus (Bundy 1985; Bundy 1992; Liu 2001) (IC) was invented to provide a logic for probabilistic reasoning that ensured the correct propagation of probabilities through logical formulae. The usual means for adding probabilities to logics is to assign probabilities directly to syntactic features in the logic, which for our purposes will be limited to propositions. Thus, if $p(\phi) = \frac{1}{2}$ (the probability of ϕ being true is 50%) and $p(\psi) = \frac{1}{2}$ then, by definition, $p(\phi \wedge \psi) = \frac{1}{4}$. Such a mechanism cannot handle correlations between statements: if ϕ represents the state of high ambient temperature, and ψ for snow, it is unlikely that the probability of both is $\frac{1}{4}$, yet most logics with probability give that answer.

Logics which maintain correctness in this respect are known as 'truth functional',

[1] `http://www.openk.org`

and the incidence calculus is one such system. The incidence calculus does not directly associate a proposition with a numerical probability. Instead, it assigns an 'incidence' to each proposition. An incidence is a set of possible worlds in which certain formulae are true or false. We will formally introduce the calculus shortly, but we begin with a summary of the syntax and an example.

We begin with sentences of propositional logic, and the propositions themselves such as p,q, and the boolean truth values $true$ and $false$. Propositions can be negated ($\neg p$), and combined in conjunctions ($p \wedge q$) and disjunctions of sentences ($p \vee q$). Sentences may also contain implications ($p \rightarrow q$) and equivalences ($q \leftrightarrow q$). Each sentence has an interpretation, mapping the sentence to $\{true, false\}$. The incidences are sets of worlds where a statement is true, and the incidence of a sentence ϕ is written $i(\phi)$. $\mathcal{W}$ is the set of all possible worlds. The probabilities are calculated by comparing the size of the incidence sets.

As an example, let's examine the dinners eaten by a student over a week. We can declare the worlds to be the days of the week, and the propositions as the various kinds of food and drink consumed. Our student's intake is described as follows:

$$
\begin{aligned}
worlds = \mathcal{W} &= \{mon, tue, wed, thu, fri, sat, sun\} \\
i(haggis) &= \{mon, thu\} \\
i(fish) &= \{fri\} \\
i(lamb) &= \{sun\} \\
i(porridge) &= \{tue, wed, sat\} \\
i(potatoes) &= \{mon, thu, sun\} \\
i(chips) &= \{fri\} \\
i(wine) &= \{fri, sun\} \\
i(whisky) &= \{mon, thu\}
\end{aligned}
$$

The probability of any particular incidence ϕ is the ratio $\frac{|i(\phi)|}{|\mathcal{W}|}$. So we can now see that:

$$\begin{aligned} p(haggis) &= \tfrac{|\{mon,thu\}|}{|\mathcal{W}|} &= \frac{2}{7} \\ p(haggis \wedge wine) &= \tfrac{|\{mon,thu\} \cap \{fri,sun\}|}{|\mathcal{W}|} &= \frac{0}{7} \\ p(haggis \wedge whisky) &= \tfrac{|\{mon,thu\} \cap \{mon,thu\}|}{|\mathcal{W}|} &= \frac{2}{7} \end{aligned}$$

In particular, note that $p(haggis \wedge whisky)$ and $p(haggis \wedge wine)$ have different values, even though $p(whisky)$ and $p(wine)$ are both $\frac{2}{7}$. This property is used later to select multi-service collaborations where individual services may have identical individual performance histories, but their combinations differ.

Conditional probabilities in the incidence calculus are as expected. The probability of ϕ being true given that ψ is true, denoted $p(\phi|\psi)$, is

$$\begin{aligned} p(\phi|\psi) &= p(\phi,\psi)p(\psi) \\ p(\phi|\psi) &= \tfrac{|i(\phi)\wedge i(\psi)|}{|\mathcal{W}|} \cdot \tfrac{|\mathcal{W}|}{|i(\psi)|} \\ p(\phi|\psi) &= \tfrac{|i(\phi)\wedge i(\psi)|}{|p(\psi)|} \end{aligned}$$

This being the case, we can compute the conditional probability of various events:

$$p(whisky \mid haggis) = \frac{|\{mon, thu\} \cap \{mon, thu\}|}{|\{mon, thu\}|} = 1$$

5.2.1 Formal definition

We will now formally define the incidence calculus, following (Liu 2001). We define the set of logical sentences $\mathcal{L}(P)$ we can form from P, a finite set of atomic propositions, and the usual logical connectives.

- $true, false \in \mathcal{L}(P)$
- if $p \in P$ then $p \in \mathcal{L}(P)$
- if $p, q \in \mathcal{L}(P)$ then p, q,$\neg p$,$\neg q$, $p \wedge q$, $p \vee q$, $p \rightarrow q$, and $p \leftrightarrow q \in \mathcal{L}(P)$

Which is to say that our language $\mathcal{L}(P)$ is closed under the logical connectives of negation, conjunction, disjunction, implication, and equivalence. We distinguish a 'basic element set'. If $P = \{p_1, p_2, \ldots, p_n\}$ then a basic element is $q = q_1 \wedge \ldots \wedge q_n$ where each q_i is either p_i or $\neg p_i$. The set of all such elements is the 'basic element set'. Every sentence in $\mathcal{L}(P)$ can be represented by a disjunction of basic elements $\phi \in \mathcal{L}(P) = q_1 \vee \ldots \vee q_n$.

The probability space, (X, χ, μ), has

- a sample space X
- a σ-algebra χ over X
- a probability measure $\mu : \chi \rightarrow [0, 1]$

A σ-algebra Σ on a set X is a subset of the power sets of X such that:

- $\emptyset \in \chi$
- if $X' \in \Sigma$ then $X \backslash X' \in \Sigma$
- the union of countably many subsets of Σ is in Σ

That is, Σ is closed under complement and union. And the probability measure obeys the following properties:

- $\mu(X_i) \leq 1$ for all $X_i \in \chi$
- $\mu(X) = 1$
- $\mu(\bigcup_{j=1}^{\infty} X_j) = \sum_{j=1}^{\infty} \mu(X_j)$ if the X_js are disjoint members of χ.

We can now define incidence calculus theories:

$$\langle \mathcal{W}, \mu, P, \mathcal{A}, i \rangle$$

- $\mathcal{W}$ is the set of possible worlds

- μ assigns a probability for every $w \in \mathcal{W}$, and $\mu(\mathcal{W}) = 1$. $\mu(I) = \sum_{w \in I} \mu(w)$
- P is the (finite) set of atom propositions.
- $\mathcal{A}$ are the axioms, and are a subset of $\mathcal{L}(P)$. Axioms are those elements of $\mathcal{L}(P)$ for which there is a known assignment of worlds.
- i is a function $i : \mathcal{A} \rightarrow 2^{\mathcal{W}}$. $i(\phi)$ is the set of possible worlds in which ϕ is true. $i(\phi) = \{w \in \mathcal{W} | w \models \phi\}$

Since only the sentences in $\mathcal{A}$ are defined directly, we extend i to $\mathcal{L}(A)$ as follows

$$\begin{aligned} i(true) &= \mathcal{W} \\ i(false) &= \emptyset \\ i(\neg\phi) &= \mathcal{W} \backslash i(\phi) \\ i(\phi \wedge \psi) &= i(\phi) \cap i(\psi) \\ i(\phi \vee \psi) &= i(\phi) \cup i(\psi) \\ i(\phi \rightarrow \psi) &= \mathcal{W} \backslash i(\phi) \cup i(\psi) \end{aligned}$$

And the probabilities of elements in $\mathcal{A}$ are defined as

$$\begin{aligned} p(\phi) &= \frac{|i(\phi)|}{|i(true)|} \\ p(\phi|\psi) &= \frac{|i(\phi \wedge \psi)|}{|i(\psi)|} \end{aligned}$$

When considering probabilities of sentences in $\mathcal{L}(P) \backslash \mathcal{A}(P)$, it is not necessarily possible to find an exact incidence set. Upper and lower bound bound incidences are defined as

$$\begin{aligned} i^*(\phi) &= \bigcap_{\psi \in \mathcal{L}(A)} \{i(\psi) | i(\phi \rightarrow \psi) = \mathcal{W}\} \\ i_*(\phi) &= \bigcup_{\psi \in \mathcal{L}(A)} \{i(\psi) | i(\phi \rightarrow \psi) = \mathcal{W}\} \end{aligned}$$

These bounds come in to play when performing inference. For example, in the following application of modus ponens

$$\frac{\phi \rightarrow \psi \quad \phi}{\psi}$$

the incidence of ψ cannot be precisely determined. IC has been used to construct truth maintenance systems (Liu, Bundy, and Robertson 1993) which can propagate the bounds to increase the precision of the resulting probabilities. For our purposes, this does not present a problem, since we do not perform inference.

5.2.2 Justification

The incidence calculus is not frequently applied, since one requires exact records of the incidences to use it. For the application at hand, however, we will have detailed information about each matchmaker invocation, and the calculus provides a simple, intuitive way of dealing with the data and reducing it to probabilities.

5.3 Summary

We introduced two formalisms which underly our approach. The first, the lightweight coordination calculus, provides a simple way of specifying peer-to-peer interactions. The second, incidence calculus, offers a simple means to compute the probabilities of complex expressions, which will come about as our matchmakers try to work out the best combinations of agents for each kind of interaction they are asked to broker.

- LCC provides a lightweight means to specify and deploy service orchestrations. LCC brings with it a world-view in which the workflows are first-class objects with many participating services. When services are no longer the centrepiece, we can begin to consider their *interaction*. Much of the context of these interactions can be captured as in the LCC model, and in the evolving interactions which they shape.

- The incidence calculus is truth functional probability calculus based on sets. In the next chapter, IC will be shown to map intuitively to the notion of interactions and their use of particular services. In chapter 7, we show how it provides a means to predict the effectiveness of multi-service collaborations.

Chapter 6

Monogamy

Our thesis ultimately requires building a matchmaker that can matchmake multi-party interactions, learning how to do so from experience of prior matchmaking events. Before constructing such a creature, we will first see how LCC and the incidence calculus can support a matchmaker which learns how to matchmake single-party interactions. The problem of a client locating a single provider, a case we will call 'monogamy', is the most fundamental matchmaking event, and the easiest to examine.

We begin this chapter by examining the simplest matchmaker, one which selects randomly amongst eligible providers. The intention is to show how we integrate the matchmaking process with LCC, and to establish a baseline behaviour. We then introduce a matchmaker using incidence calculus which learns from user feedback about its selections. Next, we reconstruct in our framework the matchmaker of (Zhang and Zhang 2002), which was the first to use statistical information about agents' prior performance to select providers. We consider the choices taken by Zhang and Zhang, and how they alter the matchmaker's performance.

6.1 Matchmaking and LCC

We begin by showing how we integrate matchmaking with the LCC model. The task is twofold:

1. Describe the services

2. Deal with client requests and the matchmaker's responses in the LCC execution model.

We solve the first very simply, by side stepping the issue of providing a sophisticated service description. Since we argue in this thesis that such descriptions will be both hard to create, and will not guarantee perfect matches anyway, we will adopt the convention that services are identified only through the role name. This means that any service advertising its proficiency in, say, the role of $weather\text{-}forecaster$, will always be capable of precisely that. In a framework like OWL, this step would correspond to the common subsumption matchmaking (Sycara et al. 2003*b*; Li and Horrocks 2004). We would expect such a matching phase to easily integrated into our model, but we proceed without it, assuming that it works to a satisfactory degree (as their proponents themselves claim) to provide us with a set of services for a query which are close enough for us to apply our statistics-based matching.

The second task of inserting the matchmaking process into LCC can be address through several models, classified here along axes related to participation by the agent, and alteration of the interaction model:

1. An agent can inspect the LCC protocol to determine unfulfilled roles, and query the matchmaker by executing a separate interaction. This requires insight on the part of the agent into LCC, something which LCC itself explicitly seeks to avoid since it increases the coupling between agents.

2. The interaction model itself can be rewritten so that the matchmaker is invoked before sending messages to unrecruited agents. Doing this greatly complicates the models, making them less understandable.

3. The LCC model interpreter can invoke the matchmaker out of band when the interaction itself requires a recruiting operation—that is, make a direct call to the matchmaker outwith the structure of the LCC model currently in play.

We will adopt the third approach, which avoids modifying the interaction models themselves, and requires no intelligence from the agent executing the model. It is the most straightforward to implement and means we can cleanly and simply extend the interaction rewrite rules to include the matchmaking operations.

So, how do we change the rewrite rules of figure 5.2 for matchmaking? Since a matchmaker is involved at the point where an agent requires a service from another—as yet unidentified—agent, the natural intercept point in LCC is at the 'send message' stage. This affects the unconditional and conditional send message rules $m \Rightarrow a(Id, Role)$ and $m \Rightarrow a(Id, Role) \leftarrow C$. The augmented version of the rules are shown in figure 6.1. The function $recruit$ is responsible for the invocation of the matchmaker, if necessary, and the modification of the interaction model to reflect that. When $recruit$ needs to find a new service, it delegates to the $matchmake$ function. The $recruit$ function behaves as follows:

1. Check if the term Id in $a(Role, Id)$ in the active clause is instantiated. If it is, the agent is known, and no action need be taken.

2. If Id is not instantiated, try to find a clause matching $collaborator(Role, Id)$ in the interaction's common knowledge, K. If found, unify the Ids.

3. Otherwise, a new agent must be selected, by calling the $matchmake$ function. The algorithms used to implement $matchmake$ may vary, and their character is the topic of much of this thesis. Once an agent is identified, the variable Id is unified with it, and the role/agent combination is recorded in the interaction's common knowledge. Specifically, the fact $knows(matchmaker, collaborator(Role, Id))$ is added to the interaction's common knowledge K.

Remember from chapter 5 that an interaction is a triple $I = \langle M, C, K \rangle$, where M is the model, C are the clauses currently in use, and K is the common knowledge. The reason behind storing the role/agent pairs in the common knowledge as well as simply instantiating them in the live clauses is that the bindings of variables do not persist across

instances of the clauses. As the interaction executes, it is possible for clauses to be reselected from the model, and thus for a previously identified agent to be unidentified in a specific clause freshly copied from the model. Keeping this information in the common knowledge allows the LCC interpreters to retain collaborator identities across clause instantiation. Moreover, a client may add its own collaboration selections to an interaction's common knowledge, if they wish to use particular services for some or all roles.

Our matchmakers require another point of contact with the agent: they need feedback from the client which initiated the interaction on the outcome of the interaction. Since this is done at only one point, and by only one agent, it is easiest to simply alter the model to require the client to send one last message to the matchmaker at the conclusion of the interaction. We do this by adding a term to the end of the client's role:

$$outcome(Outcome) \Rightarrow a(matchmaker, Matchmaker) \leftarrow$$
$$interaction\text{-}outcome(Outcome)$$

This uses a proaction constraint $interaction\text{-}outcome$, defined by the agent, and sends a simple message to the matchmaker notifying it of the result.

To illustrate, we can imagine a periodic communication, such as that made by a user's weather monitoring application to a weather information service. The model M might look like this:

$$a(weather\text{-}watcher(Location, Interval), Watcher) ::$$
$$get\text{-}forecast(Location) \Rightarrow a(weather\text{-}forecaster, Forecaster)\ then$$
$$forecast(Forecast) \Leftarrow a(weather\text{-}forecaster, Forecaster)\ then$$
$$a(weather\text{-}watcher(Location, Interval), Watcher) \leftarrow wait(Interval)$$

$$a(weather\text{-}forecaster(Location), Forecaster) ::$$
$$get\text{-}forecast(Location) \Leftarrow a(weather\text{-}watcher(\_, \_), Watcher)\ then$$
$$forecast(Forecast) \Rightarrow a(weather\text{-}watcher(\_, \_), Watcher) \leftarrow$$
$$forecastFor(Location, Forecast)$$

Figure 6.1 LCC rewrite rules for monogamous matchmaking

These rewrite rules extend those introduced in figure 5.2 to support the selection and communication of the set of collaborating peers. Note the new terms $\mathcal{C}$—the set of collaborators before the rewrite—and $\mathcal{C}'$ (if present) the, possibly extended, set of collaborators after the rewrite. $\mathcal{C}$ is a set of pairs of role and service name, e.g. $col(search\text{-}engine, ferret)$. The same rewrite rules hold regardless of the implementation of the matchmaking function $matchmake$.

$$\frac{B \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{C},O} E}{A :: B \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{C},O} A :: E}\ def$$

$$\frac{A_1 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{C},\mathcal{C}',O} E \quad \neg\, closed\,(A_2)}{A_1\ or\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{C},\mathcal{C}',O} E}\ or_{\ 1}$$

$$\frac{A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{C},\mathcal{C}',O} E \quad \neg\, closed\,(A_1)}{A_1\ or\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{C},\mathcal{C}',O} E}\ or_{\ 2}$$

$$\frac{A_1 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{C},\mathcal{C}',O} E}{A_1\ then\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{C},\mathcal{C}',O} E\ then\ A_2}\ then_{\ 1}$$

$$\frac{A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{C},\mathcal{C}',O} E \quad closed\,(A_1) \quad collaborators(A_1) = \mathcal{C}}{A_1\ then\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{C},\mathcal{C}',O} A_1\ then\ E}\ then_{\ 2}$$

$$\frac{(M \Leftarrow A) \in M_i \quad satisfied(C)}{C \leftarrow M \Leftarrow A \xrightarrow{M_i,M_i\setminus\{M\Leftarrow A\},\mathcal{P},\mathcal{C},\emptyset} closed(M \Leftarrow A, \mathcal{C})}\ reaction$$

$$\frac{satisfied(C) \quad \mathcal{C}' = recruit(\mathcal{P}, \mathcal{C}, role(A))}{M \Rightarrow A \leftarrow C \xrightarrow{M_i,M_i,\mathcal{P},\mathcal{C},\mathcal{C}',\{M\Rightarrow A\}} closed(M \Rightarrow A, \mathcal{C}')}\ proaction$$

$$\frac{satisfied(C)}{null \leftarrow C \xrightarrow{M_i,M_i,\mathcal{P},\mathcal{C},\emptyset} closed(null, \mathcal{C})}\ end$$

$$\frac{clause\,(\mathcal{P}, C, a(R, I) :: B) \quad satisfied(C)}{a(R, I) \leftarrow C \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{C},\emptyset} a(R, I) :: B}\ role$$

The $weather\text{-}watcher$ agent $Watcher$ beings executing the interaction I which requires it to send a message $get\text{-}forecast(Location)$ to the agent $Forecaster$ which can fill the $weather\text{-}forecaster$ role. Since $Forecaster$ is uninstantiated, the $Watcher$ agent's LCC interpreter queries a matchmaker with the interaction I. The matchmaker finds a suitable $weather\text{-}forecaster$ agent (say wf_i), instantiates the $Forecaster$ variable in the active clause of I, and adds the term $col(weather\text{-}forecaster, wf_i)$ to I's common knowledge. With $Forecaster$ known, $Watcher$ can send the message. The $Forecaster$ receives the request and, contingent on it being able to satisfy $forecastFor(Location, Forecast)$, it will return the forecast to the $Watcher$. The $Watcher$ agent then waits for $Interval$ before repeating the interaction.

Since the $weather\text{-}watcher$ role is tail-recursive, when the $Watcher$ agent re-enters it, the LCC expander will create a new copy of the definition clause with only the $Watcher$ and $Location$ variables instantiated (since they are instantiated in the 'call' to the role). Thus, the $Forecaster$ variable is unbound on every fresh invocation of the role. We should not invoke the matchmaker again, however. Instead, the LCC expander checks the common knowledge and instantiates the agent variable itself.

This particular interaction model never terminates, so no success rating is returned to the matchmaker. We can change this by replacing the $weather\text{-}watcher$ role with this version:

$$
\begin{array}{l}
a(weather\text{-}watcher(Location, Interval), Watcher) :: \\
\quad get\text{-}forecast(Location) \Rightarrow a(weather\text{-}forecaster, Forecaster)\ then \\
\quad forecast(Forecast) \Leftarrow a(weather\text{-}forecaster, Forecaster)\ then \\
\left(\begin{array}{l}
a(weather\text{-}watcher(Location, Interval), Watcher) \leftarrow \\
\qquad wait(Interval) \\
or \\
outcome(Outcome) \Rightarrow a(matchmaker, Matchmaker) \leftarrow \\
\qquad interaction\text{-}outcome(Outcome)
\end{array}\right)
\end{array}
$$

Now, if the $wait$ constraint fails, the $weather\text{-}watcher$ concludes the interaction with a final message to the matchmaker. This requires a matching role definition for the matchmaker:

$$a(matchmaker, Matchmaker) ::$$
$$record\text{-}outcome(Outcome) \leftarrow$$
$$outcome(Outcome) \Leftarrow a(weather\text{-}watcher, Watcher)$$

In the interactions models in the rest of this paper, we typically omit this particular role definition.

The various algorithms that we develop for matchmaking in this single-agent case (and, later, in the multi-party scenario) can all be interfaced in the same way to the LCC rewrite rules of figure 6.1, and the $matchmaker$ role definition. By using the same rewrite rules regardless of the matchmaking policy, we can reuse both the interaction interpreters, as well as other tools such as model checkers (Walton 2004; Osman, Robertson, and Walton 2005), on the interactions. Meanwhile, matchmaker implementers can change the $matchmake$ procedure, and clients can use their own choice of matchmaking service.

We still have to provide a mechanism for allowing service providers to inform matchmakers of their availability and capability. We will define a service advertisement from an agent as being a notice that an agent can perform a named role, such as $weather\text{-}forecaster$ in the example above [1]. The advertisements reach the matchmaker through an LCC interaction. A simple interaction model for advertising is shown in figure 6.2. In this case, since the $Matchmaker$ role itself is not specified, the matchmaker to which it is advertising will be selected by its default matchmaker—presumably itself. The client could direct the message to a specific matchmaker by adding common knowledge to the interaction in form $col(matchmaker\ M_1$.

[1] This is not the only way to proceed. Since one of the benefits of LCC is the ability to define ad-hoc protocols using ad-hoc role names, an alternative solution is to define an agent by the constraints it can satisfy. In this thesis, we use define only roles which satisfy one constraint each, and in this case, the two are equivalent in this respect.

Figure 6.2 LCC interaction model for service advertising

$$a(advertiser(Service\text{-}description), Advertiser) ::$$
$$advert(Service\text{-}description) \Rightarrow a(matchmaker, Matchmaker)$$

$$a(matchmaker, Matchmaker) ::$$
$$record(Service\text{-}description) \leftarrow$$
$$advert(Service\text{-}description) \Leftarrow a(advertiser, Advertiser)$$

As an aside, we note here that there is another way for our matchmaker to discover new services: from clients. Since a client can add a $col(role, agent)$ fact to an interaction's common knowledge, a matchmaker could use this information to discover new services. Services discovered in such a manner might well be better than average, since a user would be unlikely to deliberately select services they were not familiar with.

6.2 A random matchmaker

Our first matchmaker will identify suitable service providers by filtering by requested LCC role, and then select randomly from those filtered. We will name this algorithm MATCHMAKEONE-RANDOM.

In the algorithm MATCHMAKEONE-RANDOM, the variable $matchmaker$ holds the state of the matchmaker. For the MATCHMAKEONE-RANDOM matchmaker, this will be very straightforward: the matchmaker is represented as $\langle A \rangle$, where A is a mapping $Role \mapsto \mathcal{P}(Agent)$, from a role to the set of agents capable of fulfilling that role. This relationship is constructed from the advertisements received by the matchmaker from agents.

At this point, we introduce the running example we follow throughout this chapter, and establish a baseline of behaviour with the random selection. Our scenario is familiar to any user of the Web: calling a web search engine. As is well known, there are several

Figure 6.3 MATCHMAKEONE-RANDOM algorithm

MATCHMAKEONE-RANDOM($interaction$, $matchmaker$)

1 $role \leftarrow$ ROLEREQUIRED($interaction$)

2 $candidates \leftarrow A(role) \rhd A(role)$ is the list of agents providing $role$

3 **return** RECOMMENDONE-RANDOM($candidates$)

RECOMMENDONE-RANDOM($candidates$)

1 $a \leftarrow$ RANDOM(1,LENGTH($candidates$))

2 **return** $candidates(a)$

web search engines, all providing the same notional service, and with similar interfaces. The conventional view of matchmaking is that these services would, once identified as matching the client's service request specification, provide essentially identical services. We know from the history of web search that this is simply not the case. The engines use different search algorithms, and cover different subsets of the web. Our hypothetical user, Sergey the searcher, will be more satisfied by some than by others. Sergey executes an interaction model WEB-SEARCH, shown in figure 6.4. In the interaction model's framework, the variable SE represents the search engine which will provide the service, and will be selected by the matchmaker when the interaction is executed.

When Sergey (or his agent) executes the interaction, the LCC interpreter attempts to send the message $search(Query)$ to the search engine SE. Since SE is an unbound variable, the $recruit$ procedure is invoked locally. Sergey has not specified a search engine in the interaction's common knowledge K, so $recruit$ involves a matchmaker to find a suitable candidate. In this case, MATCHMAKEONE-RANDOM simply chooses randomly from those $search\text{-}engine$s that had advertised with it.

Our first experiment simply establishes a baseline of performance. We imagine six search engines 'Apropos', ''bout', 'Comb', 'Discover', 'Expiscator', and 'Ferret', with average performances of 90, 85, 80 75, 70, and 65 respectively. When a search engine

Figure 6.4 LCC interaction model for WEB-SEARCH scenario

$$a(searcher(Query), Searcher) ::$$
$$search(Query) \Rightarrow a(search\text{-}engine, SE)\ then$$
$$use\text{-}result(Results) \leftarrow results(Results) \Leftarrow a(search\text{-}engine, SE)$$

$$a(search\text{-}engine, SE) ::$$
$$search(Query) \Leftarrow a(searcher(Query), Searcher)\ then$$
$$results(Results) \Rightarrow a(searcher(Query), Searcher)$$
$$\leftarrow do\text{-}query(Query, Results)$$

answers a query, its response is computed from its intrinsic performance as perturbed by noise (with the noise drawn from a Gaussian distribution with $\sigma = 10$). For each search engine, we run the WEB-SEARCH interaction 200 times. The client agent judges an interaction's outcome to be 'good' (records $outcome(good)$) if the returned value r is greater than a value t, where t is a per-interaction random value drawn uniformly from $(0, 100)$. We performed a series of 200 runs for each of the six search engines, and another 200 runs using the MATCHMAKEONE-RANDOM matchmaker. Each such trial was done 30 times, and the results averaged.

We can easily predict the experiment's outcome. The average outcome for any interaction involving a given search engine e_i is

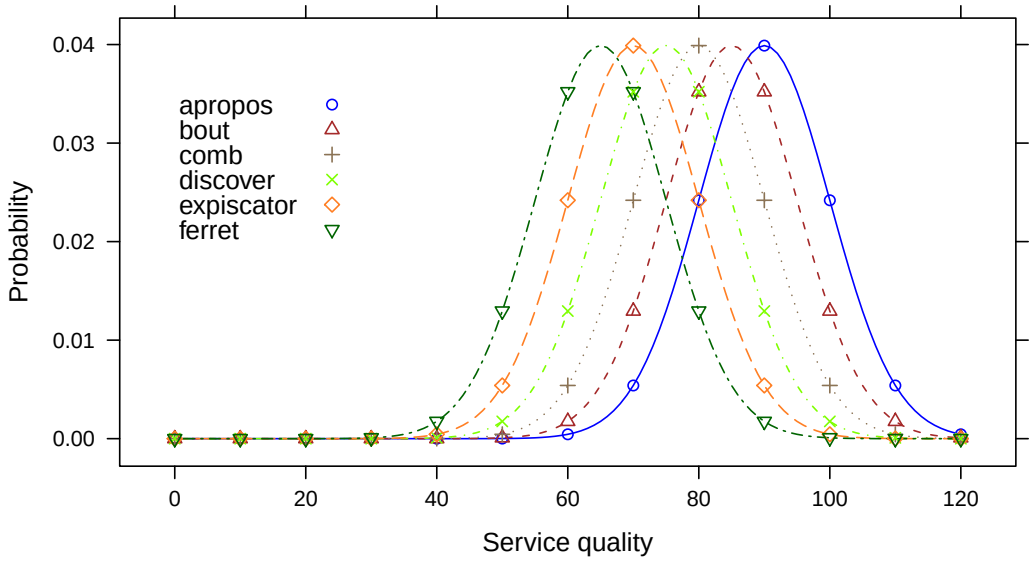
$$\varepsilon[outcome] = Q(e_i)$$

where $Q(e_i)$ is the 'intrinsic quality' of the engine e_i. Since the random matchmaker selects amongst the N engines with equal probability, the expected performance of the random incremental matchmaker is

$$\varepsilon[outcome] = \sum_i P(e_i)Q(e_i) = \frac{\sum_i Q(e_i)}{N}$$

which is just the arithmetic mean of the Q of each engine. This result is borne out in the graph of figure 6.6, which shows the performance of each service, and the random

Figure 6.5 Intrinsic abilities of hypothetical web search engines

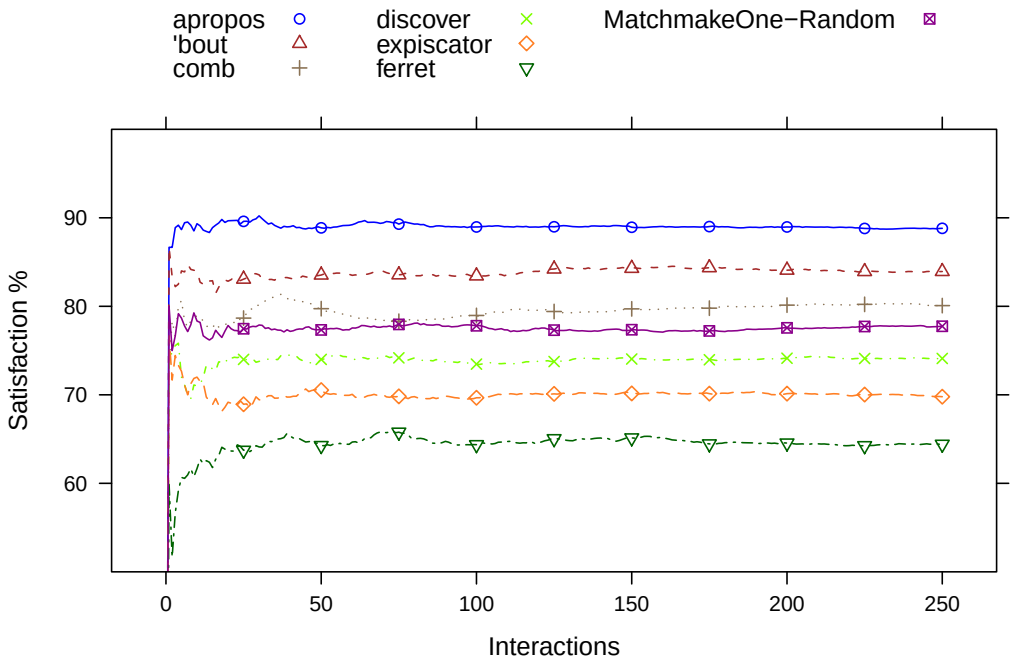


matchmaker, through the series of interactions.

6.3 Adding the incidence calculus

Having established how we integrate LCC with a matchmaking process, we can now add incidence calculus to the matchmaker. A matchmaker associates each LCC interaction it is involved with an incidence calculus world, which we represent with a unique integer. On the completion of a matchmaking session, the client reports to the matchmaker its satisfaction, and the matchmaker records the features of interest from the interaction. For the moment, these features are the name of the model involved and which agents participated in which roles. Satisfaction is measured by a simple binary good/bad decision, and represented as $outcome(good)$ or $outcome(bad)$. This gives us a set of ground predicates, viz:

Figure 6.6 WEB-SEARCH success: individual services and random matchmaking



$$col(search\text{-}engine, ferret)$$

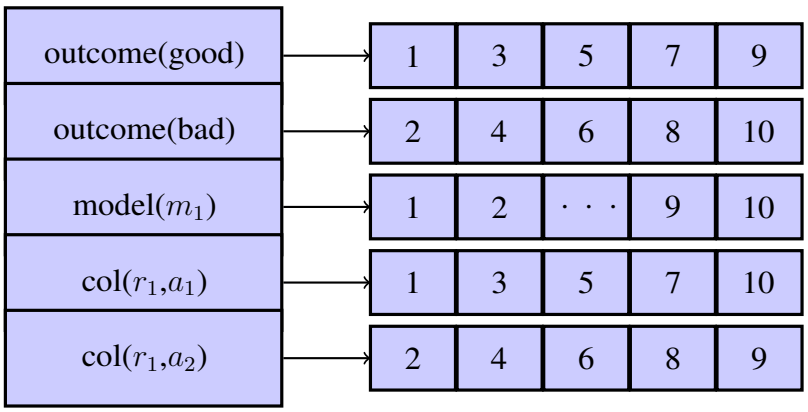
$$model(web\text{-}search)$$

$$outcome(good)$$

We can compare this with a web document, which is indexed as a list of words, and index it just as a web search engine would. Formally, our matchmaker's knowledge base is now the tuple

$$matchmaker = \langle A, I \rangle$$

where A is, as before, the the mapping from a role to the agents capable of fulfilling that role, and I is a mapping from each predicate to the list of incidences in which is true. Below is an illustration of the implementation. On the left is a hash table keyed by incidence calculus axiom, and on the right, lists of worlds in which each axiom is true:



The diagram shows a matchmaker which has been involved with interaction model m_1 ten times, and which has been informed by clients that agent a_1 is significantly more successful than agent a_2 at fulfilling the role r_1.

How do we take advantage of this database in selecting a service? When a client queries the middle-agent for a service to fulfil role r, the matchmaker selects the service which will maximise the probability of satisfying the client:

$$\text{argmax}_a P(outcome(good)|I, \{col(r, a)\} \cup collaborators(I), matchmaker)$$

The algorithm for this operation MATCHMAKEONE-IC is shown in figure 6.7.

The RECOMMENDONE-IC procedure implements an approximate argmax operation. Because the matchmaker must balance exploiting its knowledge with the need for continued exploration of the available agents' behaviour, it sometimes makes a non-optimal choice. For the moment, we will use for RECOMMENDONE-IC a policy known as ϵ-greedy (see section 7.4.2). With ϵ-greedy, we select the best agent with probability $1 - \epsilon$, and randomly choose from the others with probability ϵ. In this chapter, we set $\epsilon = 0.1$. The WEIGHTEDSORT procedure called by RECOMMENDONE-IC is tweaked so that services which have been rarely been tried are moved up the sort order, thus forcing some exploration. For this chapter, services with less than ten interactions are favoured.

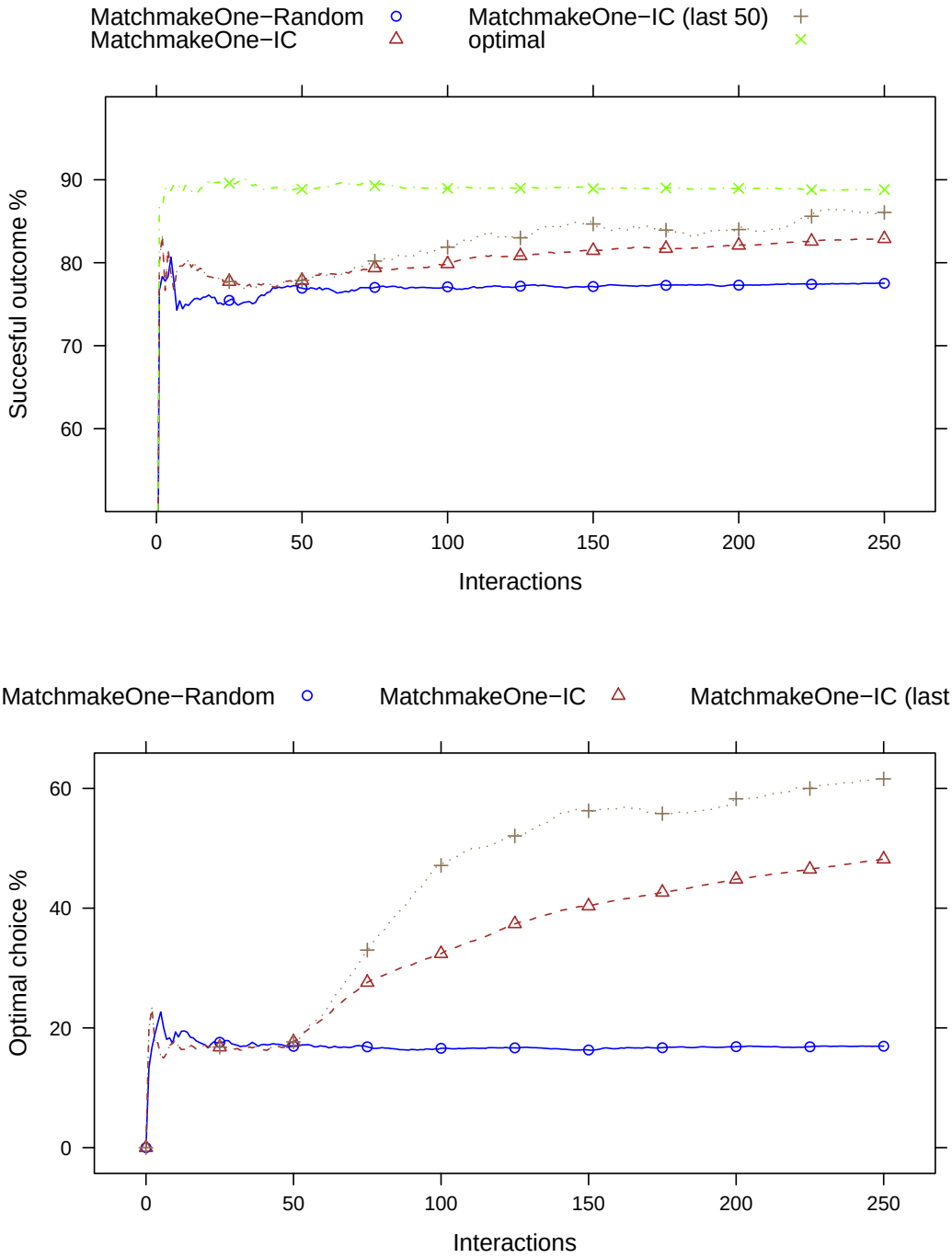
Figure 6.8 shows the results of rerunning our web-search experiment with a matchmaker operating the incremental incidence calculus policy. We see MATCHMAKEONE-IC modestly outperforms MATCHMAKEONE-RANDOM in the proportion of successful

Figure 6.7 MATCHMAKEONE-IC algorithm (ϵ-greedy)

```
MATCHMAKEONE-IC(interaction, role, database)
1  collaborators ← CURRENTCOLLABORATORS(interaction)
2  candidates ← CAPABLEAGENTS(database, role)
3  for c ∈ candidates
4       do quality[c] ← P(outcome(good)| col(role, c) ∪ collaborators)
5  return RECOMMENDONE-IC(candidates, quality)

RECOMMENDONE-IC(candidates, quality)
1  WEIGHTEDSORT(candidates,quality) ▷ Order best-first
2  r ← RANDOM(0.0,1.0)
3  if r > ε
4    then
5         return candidates[1]
6    else
7         c ← RANDOM(2,LENGTH(candidates))
8         return candidates[c]
```

Figure 6.8 WEB-SEARCH selection under different policies



The top plot shows the cumulative satisfaction of clients for the MATCHMAKEONE-RANDOM and MATCHMAKEONE-IC algorithms. The 'optimal' line is the result of always selecting the optimal service, 'apropos'. The bottom plot shows the cumulative percentage of selections of the optimal service, 'apropos'. The 'last 50' lines are based on the moving mean value over the preceding 50 interactions.

interactions, because it is much more likely to select the optimal service for any given interaction. Optimality is a useful metric: the success of an interaction is ultimately dependent on the ability of the available services, so the best we can do is pick the optimal services, even if they are less than perfect (or, indeed, they are little better than the rest). For the MATCHMAKEONE-IC policy, optimal selection is depressed by two factors:

1. The need to keep exploring: deliberately selecting sub-optimal services on occasion, both to expand the services covered, and to track a (probably) changing environment.

2. The algorithm itself settling prematurely on what it believes is the best service, but in fact is not. The data shown in figure 6.8 is drawn from a total of 30 runs, of which typically several will settle on a suboptimal service.

These behaviours are determined by the RECOMMENDONE-IC procedure.

6.4 Comparison to Zhang and Zhang

As noted in chapter 3, the first work to break out of the 'description-only' mind-set is (Zhang and Zhang 2002). They too argued that matchmaking research had ignored the possibility that agents might be selected on the basis of anything other than their own advertisements and self-appraisals of their ability. Zhang and Zhang's belief was that agents would vary in what they call *intrinsic ability* to execute a task. They did not consider that the agent would be negligent or wilfully dishonest in their service capability description, nor that the description formalism would be insufficient. They worked with an example of agents predicting future interest rates, positing several services which offer such a service, with identical interfaces. The services use different algorithms to generate their results: for example, a neural network and a genetic algorithm. Thus, depending on the suitability of the implemented machine-learning algorithm to the application, different performances will result.

In this section, we look at the method adopted by Zhang and Zhang and compare it to our incidence calculus matchmaker. Our aim is not to measure how much better one is that the other, but to draw lessons about respective merits of the approaches in the context of an open services environment.

6.4.1 The Zhang and Zhang matchmaker

The Zhang matchmaker executes a two-stage selection process. In the first phase, filtering appropriate agents based on the service description, they use the nearest neighbour model developed in the IMPACT project (Subrahmanian et al. 2000) (reviewed in section 3.2). In the second stage, Zhang and Zhang's matchmaker used records track records for each service. A client reports satisfaction on a categorical scale: $\mathit{strong\text{-}satisfaction}$, $\mathit{satisfaction}$, $\mathit{weak\text{-}satisfaction}$, $\mathit{neutral}$, $\mathit{weak\text{-}unsatisfaction}$, $\mathit{unsatisfaction}$, and $\mathit{strong\text{-}unsatisfaction}$. Thus, each agent in the database can be denoted $\langle \mathit{agent\text{-}name}, \mathit{services} \rangle$, where $\mathit{services}$ is a set of tuples of the form

$$\langle \mathit{service\text{-}name}, \mathit{inputs}, \mathit{outputs}, \mathit{record} \rangle$$

Thus each agent can offer many services, which are scored independently of one another. The record is a set of pairs

$$\langle \mathit{invocation\text{-}number}, \mathit{satisfaction\text{-}rating} \rangle$$

Once the FINDNEARESTNEIGHBOUR algorithm selects the set A of agents which might satisfy the request, an 'evaluation matrix' is constructed:

$$M = \begin{pmatrix} n_{11} & n_{12} & \cdots & n_{1k} \\ n_{21} & n_{22} & \cdots & n_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ n_{71} & n_{72} & \cdots & n_{7k} \end{pmatrix} \begin{matrix} \mathit{strong\text{-}satisfaction} \\ \mathit{satisfaction} \\ \vdots \\ \mathit{strong\text{-}unsatisfaction} \end{matrix}$$

The columns represents the agents in A, and the rows $1 \ldots k$ are the satisfaction levels $\mathit{strong\text{-}satisfaction}$ through $\mathit{strong\text{-}unsatisfaction}$. Each entry n_{ij} represents number

of agent j's scores registered at satisfaction level i. The authors propose two algorithms for selecting an agent using the evaluation table. The first is a simple set of rules:

MATCHMAKEONE-ZHANG(A)

1 **if** $\forall j \exists l . n_{1l} \geq n_{1j} \wedge n_{5l} = n_{6l} = n_{7l} = 0$

2 **then return** A_l

3 **else if** $\forall j, k . n_{1j} = n_{1k} \wedge \forall j \exists l . n_{2l} > n_{2j} \wedge n_{l5} = n_{l6} = n_{l7} = 0$

4 **then return** A_l

5 **else if** $\forall j, k . n_{jk} = 0, j, k \neq 4$

6 **then return** $ChooseOneRandomly(A)$

MATCHMAKEONE-ZHANG is the algorithm labelled 'SELECTMOSTPROMISING' in (Zhang and Zhang 2002).

The alternative algorithms they proposed reduces the qualitative scores to numbers in the interval $[-1, 1]$ (why this is used, instead of the table shown above used to compute the benchmark scores, is unexplained). The authors give a complicated algorithm for dealing with this, but the essence is simple: sum the numerical scores for each agent together, and select the highest scoring agent. They use an undefined function $ageing\text{-}check$ to make a binary decision on the currency of the agent's track record data.

6.4.2 Initial values

When services are first registered with the matchmaker, their services histories are naturally empty. Zhang and Zhang propose a benchmarking process in which the matchmaker is trained with problems with known solutions, which the matchmaker could repeatedly test its registered agents. Their benchmark problems come with correct answers: the value of the service is computed by calculating the Euclidean distance between an agent's answer and the expected one, and mapping this to the 7-point satisfaction scale. A benchmark score is recorded as $\langle 0, S_{benchmark} \rangle$. A score is computed by taking the Euclidean distance between an agent's result A_i and the

benchmark solution B_i for a task i. How this distance is normalised to or kept within $[0, 1]$ is not specified. The qualitative score is mapped to a number as an equal partition of the interval $[0, 1]$ as follows:

Distance	Satisfaction
$[0, 0.143]$	*strong satisfaction*
$(0.143, 0.286]$	*satisfaction*
$(0.286, 0.429]$	*weak satisfaction*
$(0.429, 0.572]$	*neutral*
$(0.572, 0.715]$	*weak unsatisfaction*
$(0.715, 0.858]$	*unsatisfaction*
$(0.858, 1.0]$	*strong unsatisfaction*

6.4.3 Experiment

We implemented the MATCHMAKEONE-ZHANG algorithm, and ran it on our web search example. Figure 6.9 shows the results. We used two evaluation functions, which are plotted as 'MatchmakeOne-Zhang' and 'MatchmakeOne-Zhang-Optimistic'. In the first, the search score is converted to a satisfaction rating using the following conversion from the search engine's result's 'quality' to the satisfaction level:

Score	Satisfaction
$> 90\%$	*strong satisfaction*
$> 85\%$	*satisfaction*
$> 80\%$	*weak satisfaction*
$> 75\%$	*neutral*
$> 70\%$	*weak unsatisfaction*
$> 65\%$	*unsatisfaction*
$\leq 65\%$	*strong unsatisfaction*

This algorithm under-performs incidence calculus for two reasons. The first is that the simple algorithm MATCHMAKEONE-ZHANG is very sensitive to below-par ratings.

After an initial good start, a few low ratings quickly reduce the algorithm to performing random selection, and indeed the rate at which it selects the optimal search engine converges with that of the random matchmaker. We can tweak the Zhang algorithm by changing the satisfaction function to not report any level of 'unsatisfaction':

Score	Satisfaction
$> 90\%$	*strong satisfaction*
$> 85\%$	*satisfaction*
$> 80\%$	*weak satisfaction*
$\leq 80\%\%$	*neutral*

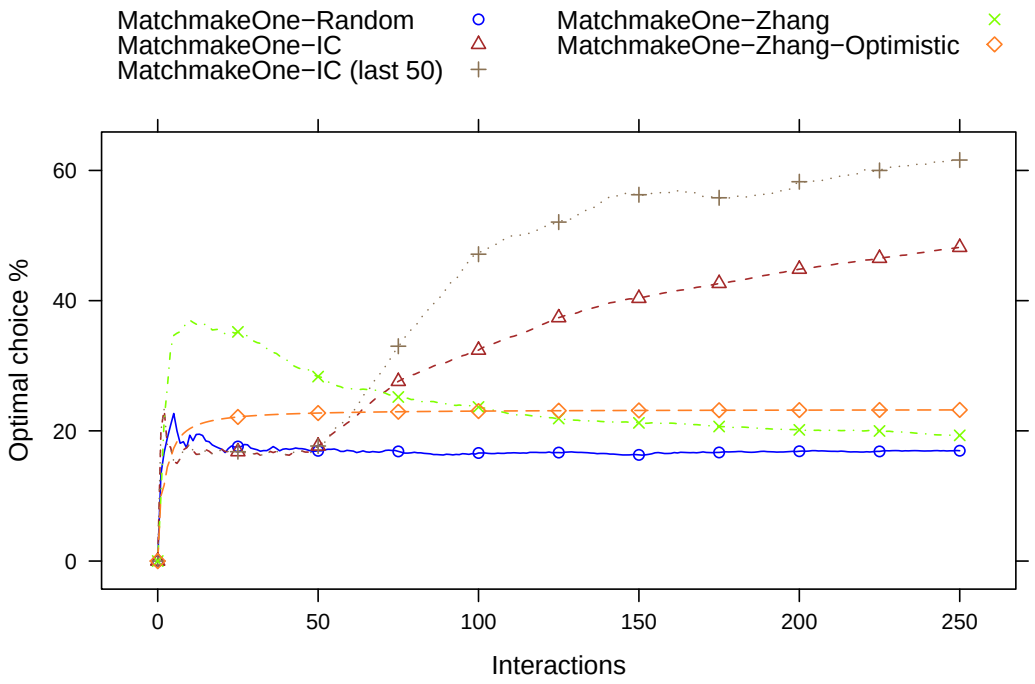
The line 'zhang optimistic' in figure 6.9 shows the improvement of this scheme.

The dependence on good initial values is high. There is no exploration after the initial 'best' service is found, so the initial values must be accurate. The ageing mechanism does not help much here, because once there is a front-runner, that agent will accumulate newer records faster than others. Because the algorithms do not attempt exploration, the accuracy of the initial track records is crucial. There ought to be a mechanism to ensure that the matchmaker continues to explore agents, both new entrants to the 'market', and to check on older agents which may have improved their performance over time.

6.5 Discussion

The comparison between our incidence calculus and the Zhang system bring up several points. First, in an open system, we cannot rely on a supply of high quality initial values. This will be particularly true where money must be spent for a service to operate, whether that is explicit, in the case of buying a flight, or in a more general sense like invoking a computational expensive process or needing generate data. Instead, the matchmaker must be capable of exploring the services itself. Secondly, the system should not be overly sensitive to individual rankings. We cannot know how individual clients will rate services, and we should not be thrown either by the occasionally bad

Figure 6.9 Zhang and Zhang algorithm on the WEB-SEARCH task



The Zhang algorithm initially performs well, but quickly deteriorates due to its over-sensitivity to occasional poor results. Using the 'optimistic' valuation, the Zhang algorithm improves its performance significantly, quickly outperforming MATCHMAKEONE-RANDOM and MATCHMAKEONE-IC, before plateauing. After about 100 runs, however, it is again outperformed by MATCHMAKEONE-IC on the moving average shown by the 'MatchmakeOne-IC (last 50)' line.

performance from a particular service. The Zhang method suffers from both of these problems, in comparison to our incidence calculus scoring.

However, the Zhang algorithm does have in its favour much lower storage requirements for track records. In needs to keep only seven numbers for every service: a count of interactions at each satisfaction level. In contrast, the incidence calculus system keeps a (small) record of every interaction. The MATCHMAKEONE-IC matchmaker as demonstrated can be realised in much simpler and more efficient way, simply by counting the good and bad outcomes. However, we show in the next chapter how the IC approach easily extends to dealing with multi-party interactions.

A final question we raise here, but one for which we do not answer, relates to the satisfaction rating itself. There are three issues:

1. Can clients be persuaded to report satisfaction?

2. How should we measure the satisfaction?

3. Are the clients' ratings trustworthy?

The first question we must ask is whether users will be prepared to invest the effort to report satisfaction. We cannot know this, but we may surmise from the ratings systems used in sites like Amazon that many users would be prepared to do so. Moreover, in many cases, the outcome of an interaction can be rated automatically: a computer can easily rate an interaction as unsatisfactory if a timeout occurs, or a error is raised. Contrarily, if the interaction runs to completion, and the end result is understandable, then the machine can return a satisfactory rating.

The second issue of what metric we use is related to the first. It is easier to elicit a simple yes/no answer than one which requires a user to value the response on a scale of, say, 0 to 100. And, while finer grading levels lead to more precise rankings from the viewpoint of individual clients, that information can be harder to use to reach a consensus. Each client has different grading criteria, and some may mark more extremely than others. A single bad rating from a harsh client might offset several votes

from less extreme happy users. Using a simple good/bad rating gives every client or interaction an equal say in ranking a set of services, and the final incidence calculus score represents a consensus view of a collaboration's adequacy. In an open system like the Internet, broad agreement from many users in different context on which services are good-enough may well prove more useful than more precise ratings which cannot be generalised out of the context that produced them.

One may well question the veracity of the outcomes reported by clients. Most obviously, they could be fraudulent, with malcontents marking down good services and giving higher than justified ratings to their own services. For matchmakers serving specialist communities such as bioinformatics, we might expect most users to be trustworthy. For open matchmakers, and services with no cost or authentication requirements, the problem of this kind of 'spamming' will be more serious. One way around this could be to search the database for malicious patterns of behaviour. Although we have not done so here, we could easily store details of clients' identity with each interaction record, and use this to curb individuals soiling the database.

6.6 Summary

In this chapter we integrated LCC with matchmaking and introduced a simple matchmaker using incidence calculus. The main points were:

- Introduction of a matchmaking model to LCC through an extension of the rewrite rules for model interactions.
- Addition of a feedback mechanism through which clients could inform the matchmaker of the success or otherwise of interactions resulting from the matchmaker's choice of services.
- Demonstration through the example of web search how performance can be improved using this matchmaker.

- Reconstruction of a seminal work in matchmaking (Zhang and Zhang 2002) in our framework. We argued that it was unnecessarily arbitrary, and required high quality initial values to offset its lack of exploration and harshness when dealing with unsatisfactory outcomes.

- Observation that, for the single agent case, our approach using incidence calculus is an expensive way of pursuing simple naive Bayes reasoning to find the optimal service. The next chapter will show how it redeems itself in multi-party interactions.

Chapter 7

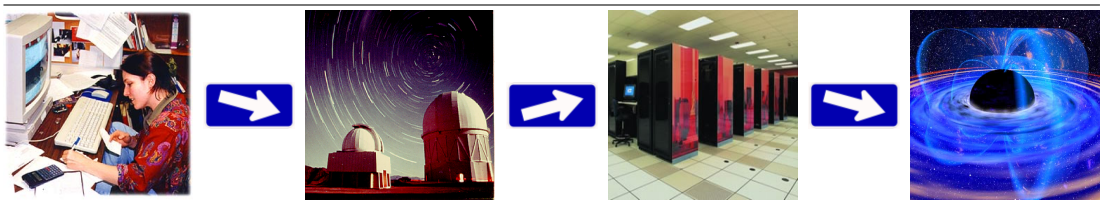
Polygamy

In the last chapter, we examined the use of an incidence calculus database to select individual services. The problem of matchmaking has historically been limited to finding just one service at a time, but in this chapter we address the fact that many tasks require the involvement of several agents. It is our thesis that in addition to agents' individual merits, overall success of an interaction will depend on how the agents interact. We extend our incidence calculus matchmaker to cope with selecting multi-agent teams, and illustrate its application in a Grid computing scenario. We then show the advantage of using a full database of interaction histories compared to simply maintaining a record of individual service performance in the multi-party case. Later in the chapter, we briefly look at using the same approach to selecting role definitions when dynamically constructing protocols. We finish by highlighting the connections between the incidence calculus matchmakers and both information retrieval and reinforcement learning.

7.1 Building a team

Let's look at an example of how our approach would work in a hypothetical scenario on the Grid (figure 7.1). We imagine that Astrid, an astronomer, is attempting to find and visualise a suspected black hole in a region of space around the object Cygnus X-1. The voluminous data about this segment of space is kept in a very large file

Figure 7.1 A Grid workflow in the astronomical domain



'*cygnusx-1*', which is stored at numerous repositories, all of which can fill the role *astronomy-database*[1]. Astrid uses an LCC interaction to copy this raw data from the database to a computationally intensive service called *black-hole-finder* which determines if there is a black hole present. The *black-hole-finder*, if successful, will send the processed data (now refined and significantly smaller) back to Astrid, who can use it to create a visualisation. The interaction model for this BLACK-HOLE-SEARCH workflow is shown in figure 7.2.

The conceit on which this example hangs is that network bandwidth between various pairs of *black-hole-finder* and *astronomy-database* is not uniform, and sometimes insufficient for the interaction to complete in a timely enough manner for Astrid to be satisfied. Since bandwidth information between two ostensibly unrelated services is not a property of the individual services, and in any case is not available to the engineers providing and describing the individual services, it is not declared in the service advertisement. In practice this kind of knowledge is unknowable—since the service provider cannot know apriori who might use their service or where and how they connect to it—and hence could not be declared to the matchmaker by any of the individual services in their advertisements. Figure 7.3 shows the network connectivity available between the various databases and compute servers we provide in this example. Note that LCC is used only to coordinate the interaction: when domain-specific protocols, such as Grid FTP, are available and more appropriate, they are used to perform the heavy

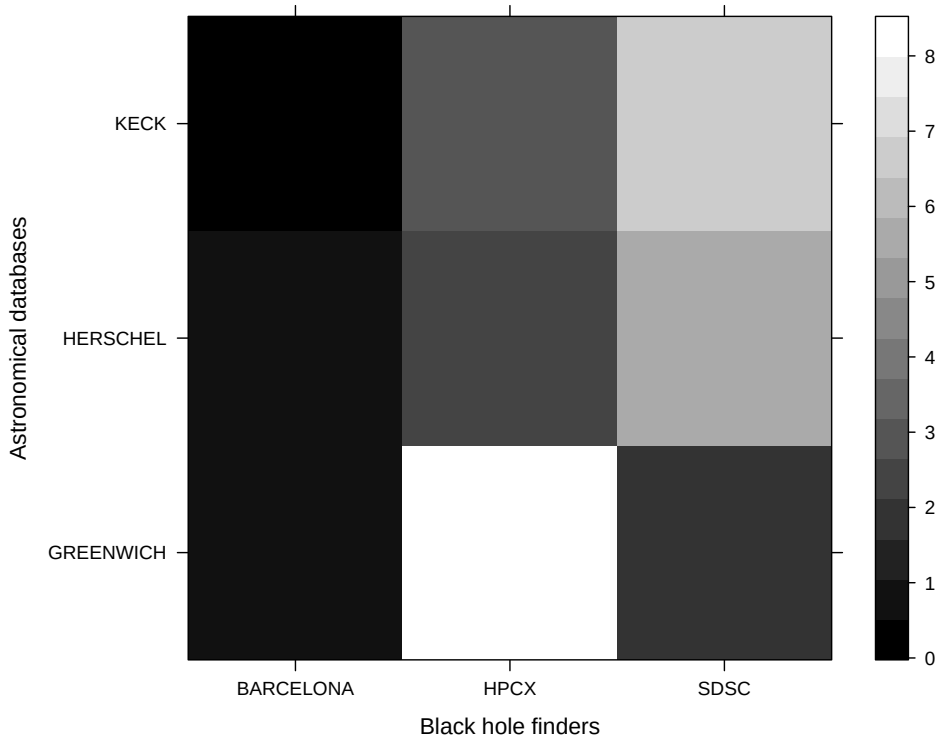
[1] One might object that not all such databases might hold that particular file. We can ignore ignore this for our purposes: the solution is to construct a set of databases which do have the file, and for the matchmaker to consider only those. This could be done by the client 'out of band', by the client after augmenting the interaction model to include a search for appropriate databases, or by an extension to the matchmaker. The end result would be a list of services able to discharge the *astronomy-database* role being added to the common knowledge which the matchmaker could use to narrow the query.

Figure 7.2 LCC interaction model for the Astrogrid scenario

$$
\begin{aligned}
&a(\mathit{astronomer}(\mathit{File}), \mathit{Astronomer}) :: \\
&\quad \mathit{search}(\mathit{File}) \Rightarrow a(\mathit{black\text{-}hole\text{-}finder}, \mathit{BHF})\ \mathit{then} \\
&\quad \mathit{visualise\text{-}black\text{-}hole}(\mathit{Data}) \leftarrow \\
&\qquad \mathit{black\text{-}hole\text{-}data}(\mathit{Data}) \Leftarrow a(\mathit{black\text{-}hole\text{-}finder}, \mathit{BHF}) \\
\\
&a(\mathit{black\text{-}hole\text{-}finder}, \mathit{BHF}) :: \\
&\quad \mathit{search}(\mathit{File}) \Leftarrow a(\mathit{astronomer}(\mathit{File}), \mathit{Astronomer})\ \mathit{then} \\
&\quad \mathit{grid\text{-}ftp\text{-}get}(\mathit{File}) \Rightarrow a(\mathit{astronomy\text{-}database}, \mathit{AD})\ \mathit{then} \\
&\quad \mathit{grid\text{-}ftp\text{-}sent}(\mathit{File}) \Leftarrow a(\mathit{astronomy\text{-}database}, \mathit{AD})\ \mathit{then} \\
&\quad \mathit{black\text{-}hole\text{-}data}(\mathit{Data}) \Rightarrow a(\mathit{astronomer}, \mathit{Astronomer}) \\
&\qquad \leftarrow \mathit{black\text{-}hole\text{-}search}(\mathit{File}, \mathit{Data}) \\
\\
&a(\mathit{astronomy\text{-}database}, \mathit{AD}) :: \\
&\quad \mathit{grid\text{-}ftp\text{-}get}(\mathit{File}) \Leftarrow a(\mathit{black\text{-}hole\text{-}finder}, \mathit{BHF}) \\
&\quad \mathit{grid\text{-}ftp\text{-}sent}(\mathit{File}) \Rightarrow a(\mathit{black\text{-}hole\text{-}finder}, \mathit{BHF}) \leftarrow \\
&\qquad \mathit{grid\text{-}ftp\text{-}send}(\mathit{File}, \mathit{AD})
\end{aligned}
$$

lifting by satisfying the appropriate constraint $grid\text{-}ftp\text{-}send$ in the $astronomy\text{-}database$ role.

Figure 7.3 Astrogrid services' network bandwidth



Megabytes of network bandwidth available for sending data from the astronomy database services to the computation servers running the black-hole finding code.

This example is different from the WEB-SEARCH model in that we must recruit multiple kinds of services in one interaction. We can solve this in two ways. The first is repeated application of our MATCHMAKEONE-IC or MATCHMAKEONE-RANDOM procedures which select a single service at a time, calling the matchmaker as the services are required by the interaction. The second is to make a selection for every role in the model at once, in a single matchmaking operation. For this, we introduce the procedure MATCHMAKEALL-IC.

MATCHMAKEALL-IC calculates the joint distribution for all possible permutations of agents in their respective roles, selecting the grouping with the largest probability of a good outcome. Formally, we compute

$$\text{argmax}_{\mathcal{C}} P(outcome(good)|I, \mathcal{C} \cup collaborators(I), matchmaker)$$

where $\mathcal{C}$ is the set of collaborators $\{col(r_i, a_j)\}$ for the currently unfilled roles in the interaction I. $collaborators(I)$ is the set of collaborators already determined for the interaction. In the Astrogrid model, it is more concretely:

$$\text{argmax}_{\{s_1,s_2\}} P \left(\begin{array}{l} outcome(good)|I, matchmaker, \\ \{col(astronomy\text{-}database, s_1), col(black\text{-}hole\text{-}finder, s_2)\} \end{array} \right)$$

The already selected collaborators $collaborators(I)$ may have been chosen through early matchmaking events, or because one of the clients explicitly selected a collaborator by inserting into the interaction's common knowledge a statement such as:

$$known(matchmaker, collaborator(black\text{-}hole\text{-}finder, hpcx))$$

As an aside—we do not implement or use it in this thesis—a simple extension would allow a client to bar specific service from participation by adding statements of the form

$$known(matchmaker, not(collaborator(Role, Agent)))$$

to the common knowledge. The pseudocode for MATCHMAKEALL-IC is shown in figure 7.4.

We could introduce MATCHMAKEALL-RANDOM but refrain from doing so, since its behaviour in terms of selecting services is not interestingly different from repeated applications of MATCHMAKEONE-RANDOM. In our naming convention, we are deviating from the traditional meaning of the terms 'matchmake one' and 'matchmake all' as used in, for example, the KQML/FIPA-ACL terminology. There, the 'one' refers to a request for the matchmaker to select a single agent matching the requirement, and 'all'

Figure 7.4 MATCHMAKEALL-IC algorithm

MATCHMAKEALL-IC($interaction$, $database$)

1 $roles \leftarrow$ UNFILLEDROLES($interaction$)

2 $collaborators \leftarrow$ CURRENTCOLLABORATORS($interaction$)

3 $candidates \leftarrow$ ALLCOLLABORATIONS($database$, $roles$, $collaborators$)

4 **for** $\mathcal{C} \in candidates$

5 **do** $quality[\mathcal{C}] \leftarrow P(outcome(good)|\mathcal{C} \cup collaborators)$

6 **return** RECOMMENDALL-IC($candidates$, $quality$)

RECOMMENDALL-IC($candidates$, $quality$)

1 WEIGHTEDSORT($candidates$,$quality$) ▷ Order best-first

2 $r \leftarrow$ RANDOM(0.0,1.0)

3 **if** $r > \epsilon$

4 **then return** $candidates$[1]

5 **else** $c \leftarrow$ RANDOM(2,LENGTH($candidates$))

6 **return** $candidates$[c]

RECOMMENDALL-IC is essentially identical to RECOMMENDONE-IC, except that it selects sets of collaborators instead of individuals. UNFILLEDROLES returns a set of those roles defined in the model but currently without a matching $col(r, a)$ in the interaction's common knowledge.

directs the matchmaker to forward all suitable advertisements to the client, so the client may perform the final selection itself. Here, we use 'one' to mean the selection of a single service for the current unfilled role, but 'all' to mean selecting one service for each of the unfilled roles in an interaction.

Filling more than one role at a time requires a modification to the the rewrite rules of figure 6.1. The modification is limited to the $recruit$ function used in the $proaction$ rule. In the monogamous case, $recruit$ knows the role which needs to be filled, since it is available as a variable in the rewrite rule. To extend it to polygamy, we can remove the explicit mention of the role immediately at hand, and have $recruit$ inspect the interaction to determine the unfilled roles (UNFILLEDROLES in the MATCHMAKEALL-IC algorithm in figure 7.4). Since the $recruit$ function's behaviour is (in principle) selectable by the agent executing the LCC interaction, the agent can retain control over the policy, at least to the extent that its chosen matchmaker will cooperate in such a policy.

Returning to our BLACK-HOLE-SEARCH example, we can compare the performance of our two incidence calculus based algorithms against random selection. The model itself was introduced back in figure 7.2. For this experiment, we ran each matchmaker policy with 250 interactions, and for statistical stability in the results, each such trial was run 30 times and averaged. Satisfaction was determined by randomly selecting a file size for the astronomical dataset: if the file size exceeds the network bandwidth between the two selected service, Astrid is unhappy. The file size is drawn from a uniform distribution between 0 and 12Mb, compared with the 0-8Mb of the available network bandwidths[2].

The plots in figure 7.5 show the comparative rates of satisfaction of the client, and of selecting the optimal service pairing. In this case, although MATCHMAKEONE-IC and MATCHMAKEALL-IC reach comparable levels of satisfaction, their optimality levels are significantly different. This is accounted for by the fact that MATCHMAKEONE-IC

[2]For simplicity, we assume that the transfer must happen within a second: we could trivially scale the numbers to more realistic values.

is selecting the first requested service ($astronomy\text{-}database$) greedily, and that for two such databases ($keck$ and $herschel$), there is a good but not optimal pairing with $black\text{-}hole\text{-}finder$service $sdsc$.

After a few interactions, the matchmaker's database might look like this:

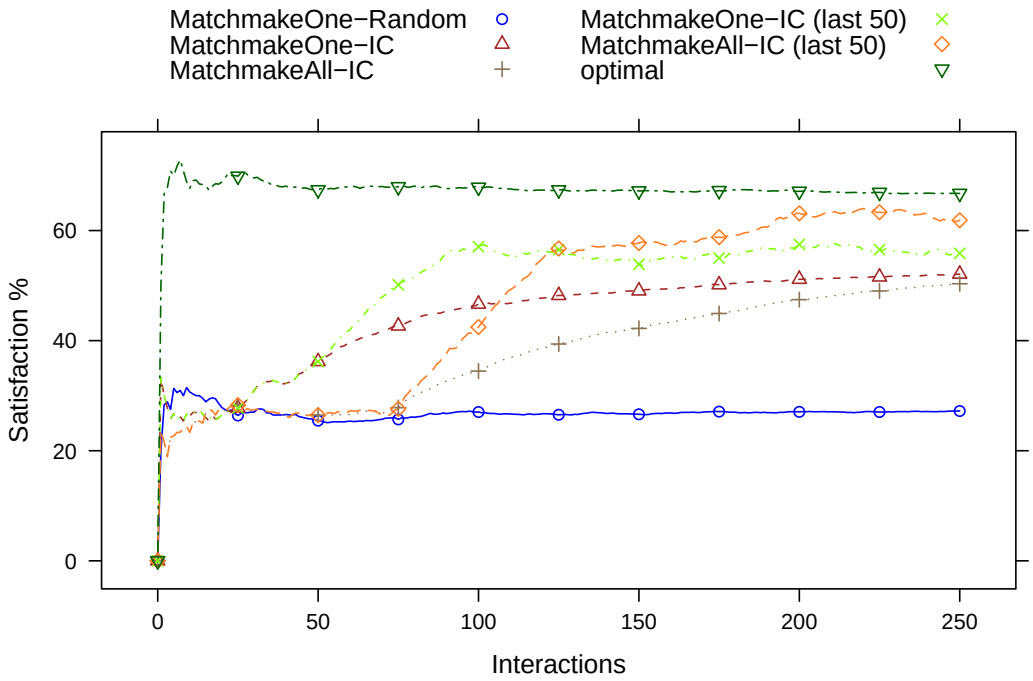
$$i(model(\text{BLACK-HOLE-SEARCH}), \{1, 2, \ldots, 24\})$$
$$i(outcome(good), \{2, 5, 6, 7, 9, 10, 11, 13, 14, 16, 17, 18, 20, 21, 24\})$$
$$i(col(astronomy\text{-}database, greenwich), \{1, 6, 11, 13, 17, 21, 22\})$$
$$i(col(astronomy\text{-}database, herschel), \{3, 7, 8, 9, 12, 16, 18, 20\})$$
$$i(col(astronomy\text{-}database, keck), \{2, 4, 5, 10, 14, 15, 19, 23, 24\})$$
$$i(col(black\text{-}hole\text{-}finder, barcelona), \{3, 4, 8, 11, 19, 22\})$$
$$i(col(black\text{-}hole\text{-}finder, hpcx), \{1, 7, 9, 10, 12, 13, 16, 21, 24\})$$
$$i(col(black\text{-}hole\text{-}finder, sdsc), \{2, 5, 6, 14, 15, 17, 18, 20, 23\})$$

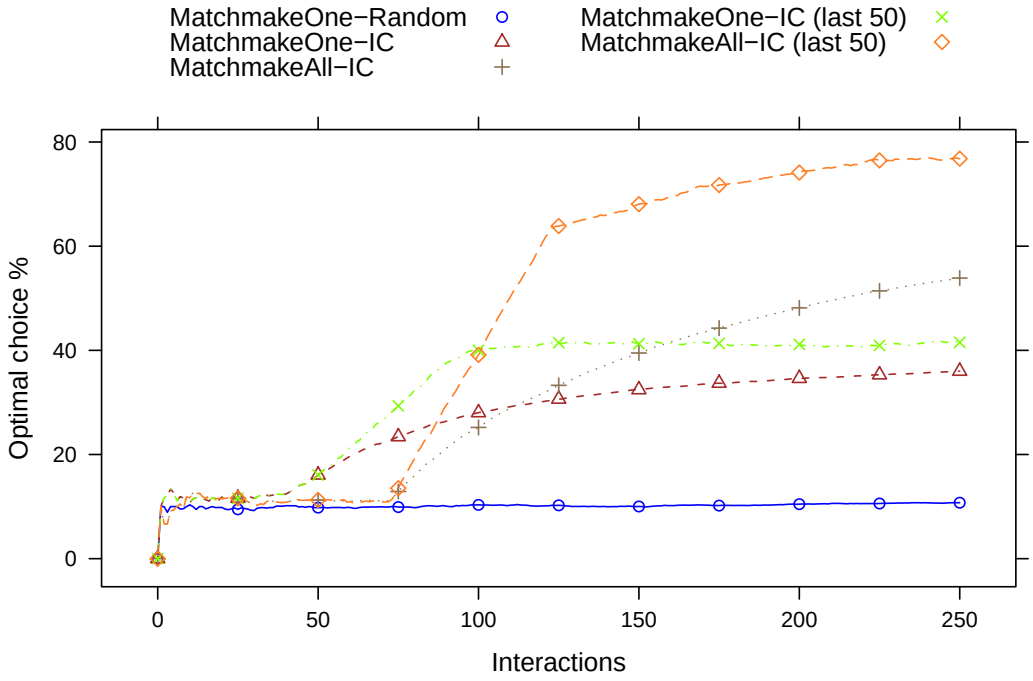
We can see the importance of the *interaction* of the services by noting that the $barcelona$ supercomputer is relatively unsuccessful compared to its rivals:

$$\begin{aligned}
i(col(black\text{-}hole\text{-}finder, barcelona) \wedge outcome(good)) &= \{11\} \\
p(outcome(good) |\, col(black\text{-}hole\text{-}finder, barcelona)) &= \frac{1}{6} \\
i(col(black\text{-}hole\text{-}finder, hpcx) \wedge outcome(good)) &= \{7, 9, 10, 13, 16, 21, 24\} \\
p(outcome(good) |\, col(black\text{-}hole\text{-}finder, barcelona)) &= \frac{7}{9} \\
i(col(black\text{-}hole\text{-}finder, sdsc) \wedge outcome(good)) &= \{2, 5, 6, 14, 17, 18, 20\} \\
p(outcome(good) |\, col(black\text{-}hole\text{-}finder, barcelona)) &= \frac{7}{9}
\end{aligned}$$

This is not because it is a worse supercomputer than $sdsc$ or $hpcx$, but because its connection to the available databases is limited.

Figure 7.5 Astrogrid selection





7.2 Better than (naive) Bayes

As noted in chapter 6, maintaining the entire database of interactions is much more memory intensive than a simpler mechanism of keeping a count of successful and unsuccessful interactions per service. The computational cost in using it is also considerable. In this section, we contrast the incidence calculus matchmakers with comparable ones which use a naive Bayes model of the services' abilities which keep track only of individual services' success rates.

Our incidence calculus matchmaker is effectively computing the Bayesian 'maximum likelihood' hypothesis, where the 'hypothesis' is the agent or agents that can be selected at the current point in time. This hypothesis h_{ML} is computed by finding the hypothesis resulting in the highest probability given the data D

$$h_{ML} = \text{argmax}_h P(h|D)$$

For our matchmakers, this is

$$h_{ML} = \text{argmax}_h P(h|\mathcal{P}, collaborators, database)$$

where the hypothesis has the form

$$h = outcome(good), newcollaborators$$

and the $newcollaborators$ being the services we are to matchmake. Computing this h_{ML} is known as 'brute force' Bayes classification, and the incidence calculus procedures do precisely this.

Because of its expense, the exact Bayesian computation is often approximated by a 'naive Bayes' classifier (Mitchell 1997). A naive Bayes classifier makes the assumption of independence between the random variables, that in general $P(x, y) = P(x)P(y)$. Where the Bayesian (and incidence calculus) calculation would know the exact probability of a successful outcome for a given set of collaborators, a naive Bayesian calculation would simply take the product of the individual services' probabilities. For

example, where IC would calculate the probability

$$P(outcome(good) \wedge col(r_1, a_1) \wedge col(r_2, a_2) \wedge \cdots)$$

the naive Bayes classifier settles for

$$\prod_i P(outcome(good), col(r_i, a_i))$$

This allows complex statements of the form found in our matchmaker queries to be computed without succumbing to the 'curse of dimensionality', either in computational expense (calculating the exact sets which apply under all the conditions) or in sparsity of data. On the flip side, naive Bayes becomes increasingly inaccurate as its independence assumption is violated. As we argued in chapter 4, and ensured in the Astrogrid scenario, assuming agents' ability to be independent of that of other agents is optimistic.

7.2.1 Naive Bayes matchmakers

We now introduce two algorithms using Naive Bayes, MATCHMAKEONE-BAYES and MATCHMAKEALL-BAYES. Figure 7.6 shows the pseudocode. The implementation of MATCHMAKEONE-BAYES is obvious: we evaluate the numerical probability of success for each individual service, and select the one with the greatest likelihood (subject to some randomness to drive exploration). MATCHMAKEALL-BAYES functions simply by repeatedly invoking MATCHMAKEONE-BAYES for each of the roles returned by UNFILLEDROLES. This is, after all, simply the application of the naive Bayes assumption that each choice is independent of the others.

7.2.2 Experiment

We rerun the experiment of the previous section, comparing naive Bayes selection with the results from the incidence calculus matchmakers. Figure 7.7 compares the satisfaction and optimality outcomes between the incidence calculus algorithms and the Bayes matchmaker. As can be seen from the graph, Bayes certainly improves over

Figure 7.6 MATCHMAKEONE-BAYES and MATCHMAKEALL-BAYES algorithms

```
MATCHMAKEONE-BAYES(interaction, role, database)
1  candidates ← CAPABLEAGENTS(database, role)
2  for c ∈ candidates
3       do quality[c] ← P(outcome(good)| col(role, c))
4  return RECOMMEND-BAYES(candidates,quality)

MATCHMAKEALL-BAYES(interaction, database)
1  roles ← UNFILLEDROLES(interaction)
2  for r ∈ roles
3       do
4            candidates ← CAPABLEAGENTS(database, role)
5            for c ∈ candidates
6                 do quality[c] ← P(outcome(good)| col(r, c))
7            collaborators[r] ← RECOMMEND-BAYES(candidates,quality)
8  return collaborators

RECOMMEND-BAYES(candidates, quality)
1  WEIGHTEDSORT(candidates,quality) ▷ Order best-first
2  r ← RANDOM(0.0,1.0)
3  if r > ϵ
4     then
5          return candidates[1]
6     else
7          c ← RANDOM(2,LENGTH(candidates))
8          return candidates[c]
```

random selection, but significantly less than incidence calculus. In figure 7.8, we can see the number of times each pairing of $astronomy\text{-}database$ and $black\text{-}hole\text{-}finder$ services is selected by each matchmaker policy. Both the IC policies, and Bayes, have discovered that the Barcelona supercomputer is a bad choice, but Bayes displays less discrimination on the other choices, lowering its overall performance.

7.2.3 Service preselection

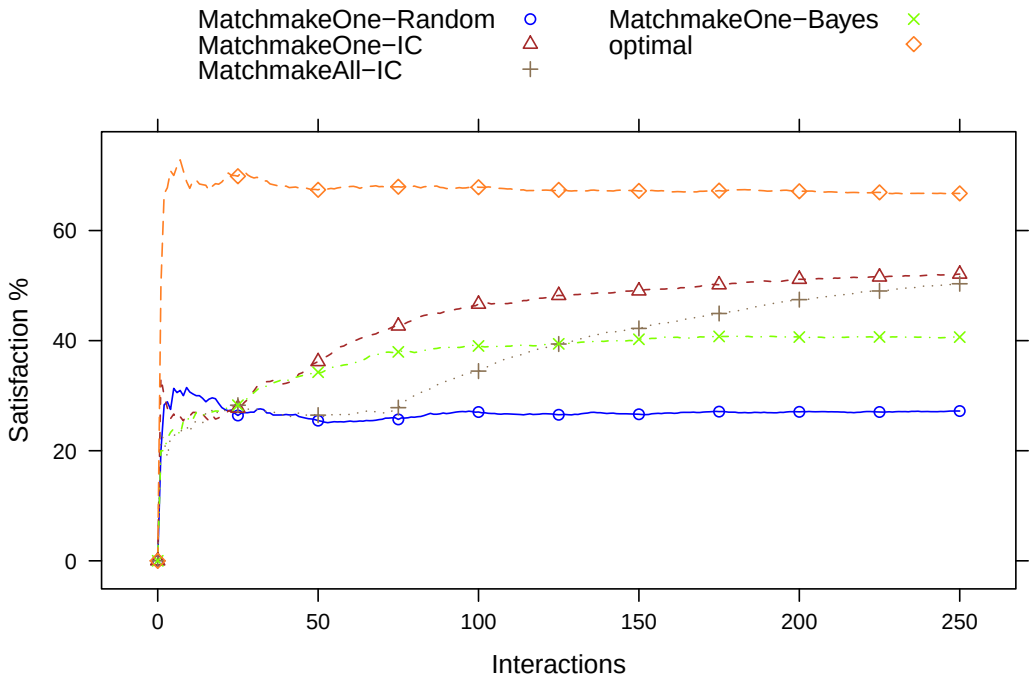
We now consider a variation on the Astrogrid scenario that shows more clearly the benefit to the incidence calculus approach. As we noted in the previous chapter, a client can preselect collaborators by modifying the interaction's common knowledge. An example of this might be a client booking a trip involving flight: if the client were accumulating frequent fly 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. For example, in a peer-to-peer search, by selecting an agent the client suspects will be helpful in a particular enquiry, the broker can find further agents that are closely 'socially' related to that first one.

Returning to our astronomer, suppose Astrid must use a particular compute server, perhaps because the operation is expensive and her grant will only stretch to paying for computer time at an affiliated institution. She would add a fact like the following to the common knowledge:

$$known(matchmaker, collaborator(black\text{-}hole\text{-}finder, hpcx))$$

informing the matchmaker, and any other collaborating agents, that she intends to use the UK's HPCX supercomputer. For our experiment, every interaction is started with a preselected $black\text{-}hole\text{-}finder$, randomly chosen from $\{hpcx, barcelona, sdsc\}$. The matchmaker should then select a service which works well with that one. Figure 7.9

Figure 7.7 Astrogrid selection with Naive Bayes



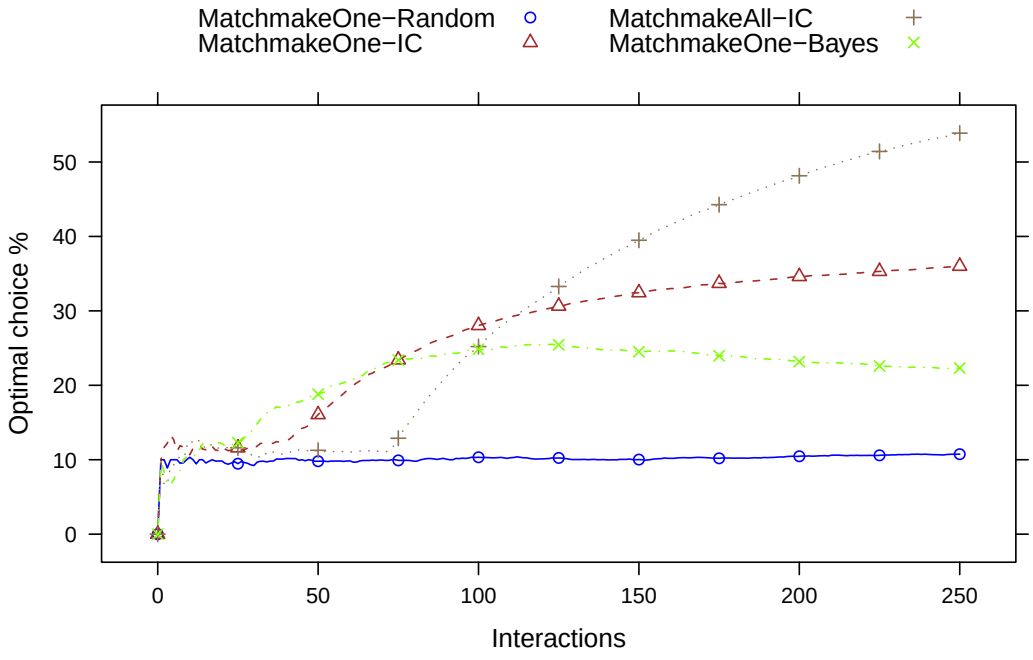
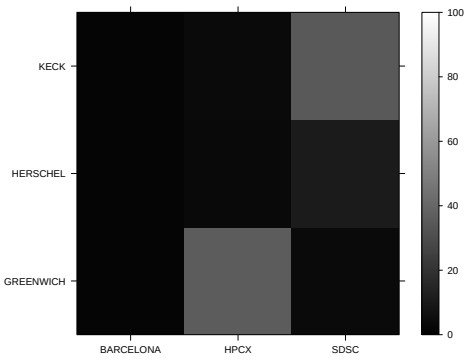
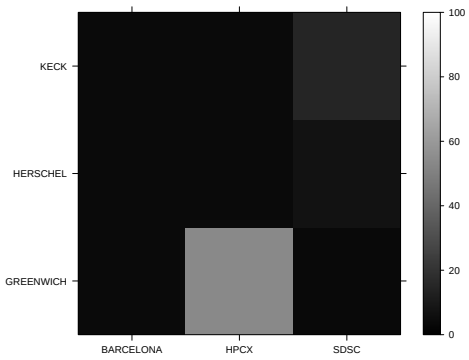


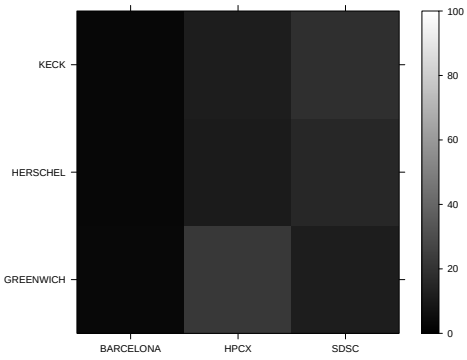
Figure 7.8 Astrogrid service selection patterns



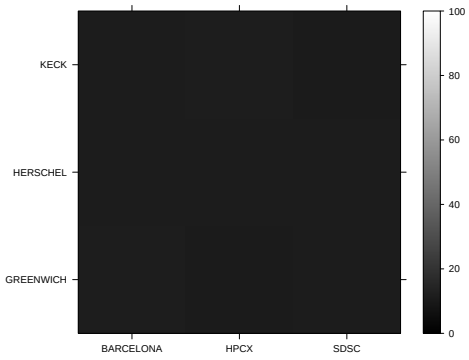
ONE-IC



ALL-IC



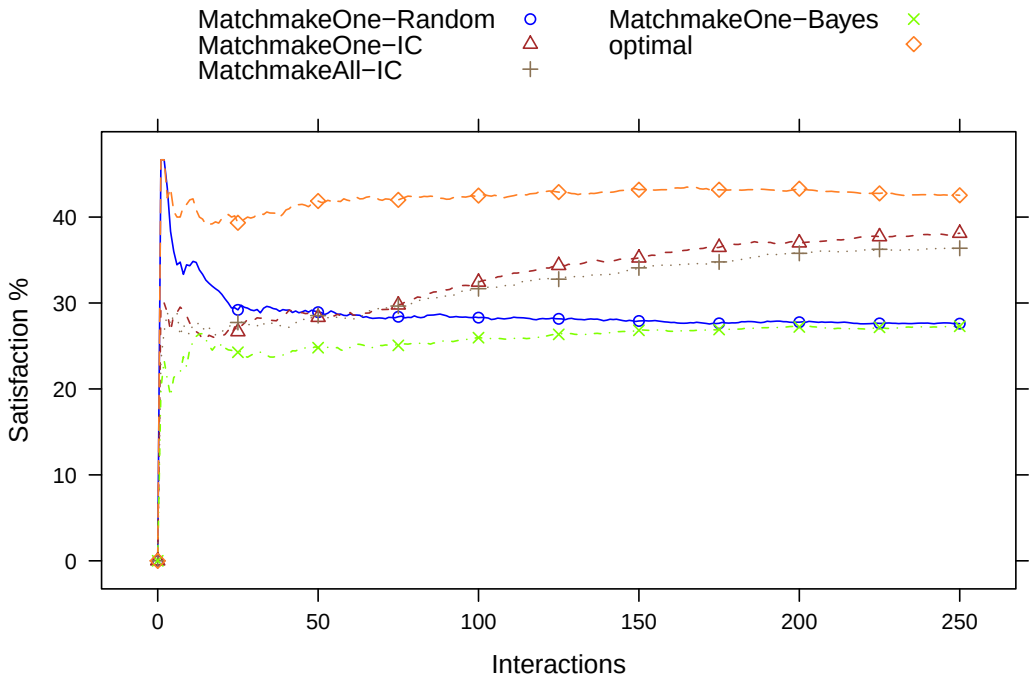
ONE-BAYES



ONE-RANDOM

shows the results for satisfaction and optimality. Now, naive Bayes performs at a level little better than random selection. The relative selection rates plotted in figure 7.10 show that the Bayes matchmaker, due to the naive Bayes assumption of independence, does not deal well with the user pre-selecting some of the services.

Figure 7.9 Astrogrid with preselected computation server



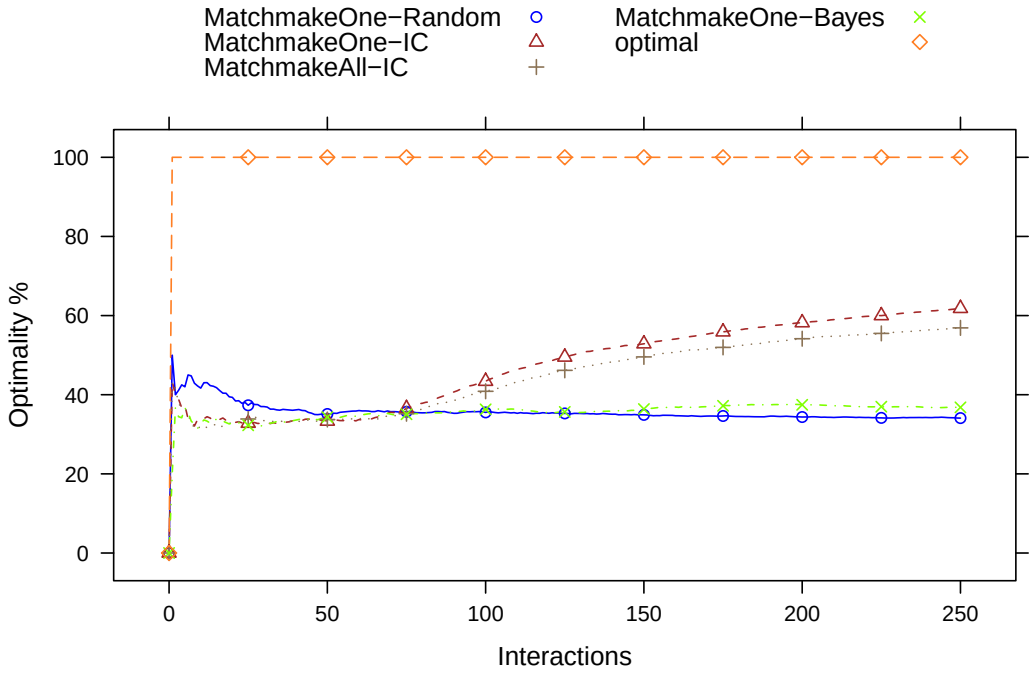
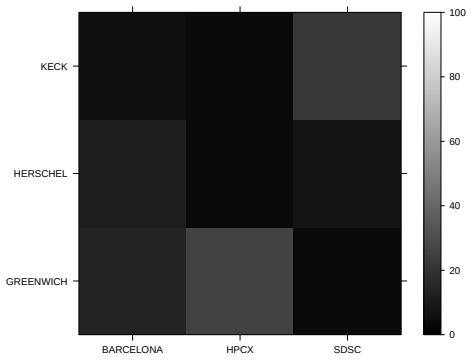
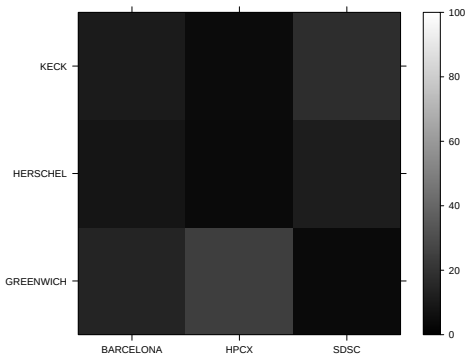


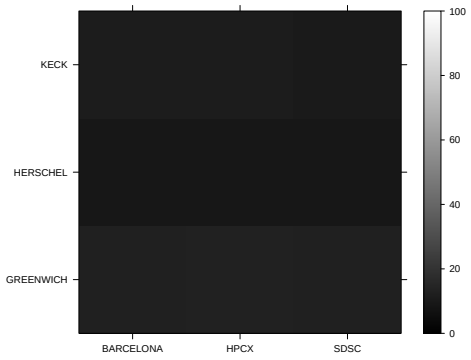
Figure 7.10 Astrogrid service selection with preselected compute server



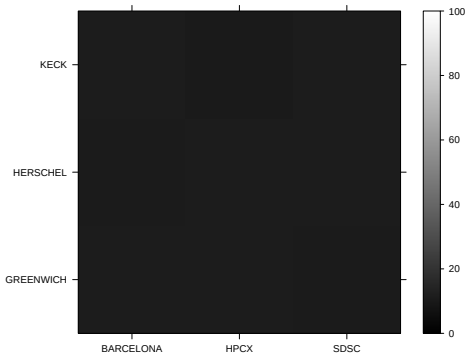
ONE-IC



ALL-IC



ONE-BAYES



RANDOM-ONE

7.3 Selecting roles

We have seen how to use performance histories to improve the selection of teams of agents. We now sketch how we can apply the same technique to select the interaction patterns themselves. Just as there are many agents capable of performing a given task (that is, fulfilling a role), so there are many sequences of actions which can lead to the achievement of a task. An interaction model is composed of several roles, which interact when messages are sent between agents. For some workflows where roles can be considered as 'subroutines' in a larger workflow, with clearly defined inputs and outputs and no interaction with other parts of the workflow, we can interchange role definitions. For instance, in a scientific setting, the preprocessing of a dataset prior to its use in the main workflow could be seen in this way.

We can structure a task in multiple ways, and we can expect that some of these will be more likely than others to produce pleasing outcomes. 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 usually preferable to obtain event tickets first then organise transport, since ticket availability is usually more constrained.

In the same way we added collaborations to the rewrite rules, we also modify the rules to account for the dynamic addition of roles to the interaction model. The rules in full are shown in figure 7.11. The principle addition is the $enrole$ function, in the def rule, which selects a role definition, if necessary, and adds it to the model P, producing an extended model $\mathcal{P}'$. Other rules are modified as necessary to ensure the modified protocols $\mathcal{P}'$ are propagated through the evaluations.

The role selecting algorithms are straightforward counterparts to their service selecting brethren. EMBELLISHONE works similarly to MATCHMAKEONE, adding role definitions to the protocol as those roles are required at run-time. We have not provided equivalent to RECRUIT-ALL, since this can inflate protocols with many roles that will

remain unused.

7.3.1 Experiment

In our example, we take the problem of booking a trip involving obtaining flight and hotel room reservations. The LCC model is shown in figure 7.12, which makes reference to a $travel\text{-}agent$ role, but contains no matching role definition. Figure 7.13 shows two alternative definitions for $travel\text{-}agent$: $\mathit{flight}\text{-}then\text{-}hotel$ and $hotel\text{-}then\text{-}\mathit{flight}$. These definitions are denoted in a form like:

$$role(\mathit{flight}\text{-}then\text{-}hotel) \equiv a(travel\text{-}agent, Agent) ::$$

where $\mathit{flight}\text{-}then\text{-}hotel$ is the name of a role definition that can be used place of the role $travel\text{-}agent$. For the purpose of this experiment, we suppose that it is a preferable course of action to book the flight then the hotel room, reasoning that hotel room costs are more flexible than flight costs.

Specifically in this experiment, we begin with a budget of £400, and assume that a flight always costs £250. The room booking service manages to find rooms at a cost of r times the proffered budget, where r is drawn from a uniform random distribution of $[0.3, 0.6]$. Thus, if the room is booked first, it will cost on average £200, and the flight will then be unaffordable. Booking the flight and then the room will result in a total cost of £300 on average (and presumably correspondingly less salubrious accommodation). As usual, we ran a total of 30 trials, each with 250 interactions. Figure 7.14 shows the results, which again show the potential benefit of a matchmaker using experience to improve its performance in selecting role definitions.

Figure 7.11 LCC rewrite rules for role selection

This set of rewrite rules extend those introduced in figure 6.1 to enable role selection at runtime. We incorporate the change to the role parameter to *recruit* mentioned in section 7.1. The rule *def* houses the enrolement machinery, in the form of the *enrole* function. *enrole* produces a new protocol $\mathcal{P}^+$ containing the new role definition (if necessary), and the evaluation of the role body occurs in the context of this new model, eventually producing a $\mathcal{P}'$. Other rules have been modified as necessary to propagate the modified models $\mathcal{P}'$ as they may arise.

$$\frac{B \xrightarrow{M_i,M_o,\mathcal{P}^+,\mathcal{P}',\mathcal{C},O} E \quad \mathcal{P}^+ = enrole(\mathcal{P},R)}{a(R,I) :: B \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{P}',\mathcal{C},O} a(R,I) :: E}\ def$$

$$\frac{A_1 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{P}',\mathcal{C},\mathcal{C}',O} E \quad \neg\, closed\,(A_2)}{A_1\ or\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{P}',\mathcal{C},\mathcal{C}',O} E}\ or_{\ 1}$$

$$\frac{A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{P}',\mathcal{C},\mathcal{C}',O} E \quad \neg\, closed\,(A_1)}{A_1\ or\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{P}',\mathcal{C},\mathcal{C}',O} E}\ or_{\ 2}$$

$$\frac{A_1 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{P}',\mathcal{C},\mathcal{C}',O} E}{A_1\ then\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{P}',\mathcal{C},\mathcal{C}',O} E\ then\ A_2}\ then_{\ 1}$$

$$\frac{A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{P}',\mathcal{C},\mathcal{C}',O} E \quad closed\,(A_1) \quad collaborators(A_1) = \mathcal{C}}{A_1\ then\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{P}',\mathcal{C},\mathcal{C}',O} A_1\ then\ E}\ then_{\ 2}$$

$$\frac{(M \Leftarrow A) \in M_i \quad satisfied(C)}{C \leftarrow M \Leftarrow A \xrightarrow{M_i,M_i\setminus\{M\Leftarrow A\},\mathcal{P},\mathcal{C},\emptyset} closed(M \Leftarrow A,\mathcal{C})}\ reaction$$

$$\frac{satisfied(C) \quad \mathcal{C}' = recruit(\mathcal{P},\mathcal{C},unfilled\text{-}roles(\mathcal{P}))}{M \Rightarrow A \leftarrow C \xrightarrow{M_i,M_i,\mathcal{P},\mathcal{C},\mathcal{C}',\{M\Rightarrow A\}} closed(M \Rightarrow A,\mathcal{C}')}\ proaction$$

$$\frac{satisfied(C)}{null \leftarrow C \xrightarrow{M_i,M_i,\mathcal{P},\mathcal{C},\emptyset} closed(null,\mathcal{C})}\ end$$

$$\frac{clause\,(\mathcal{P},C,a(R,I) :: B) \quad satisfied(C)}{a(R,I) \leftarrow C \xrightarrow{M_i,M_o,\mathcal{P},\mathcal{P}',\mathcal{C},\emptyset} a(R,I) :: B}\ role$$

Figure 7.12 Booking a trip with LCC

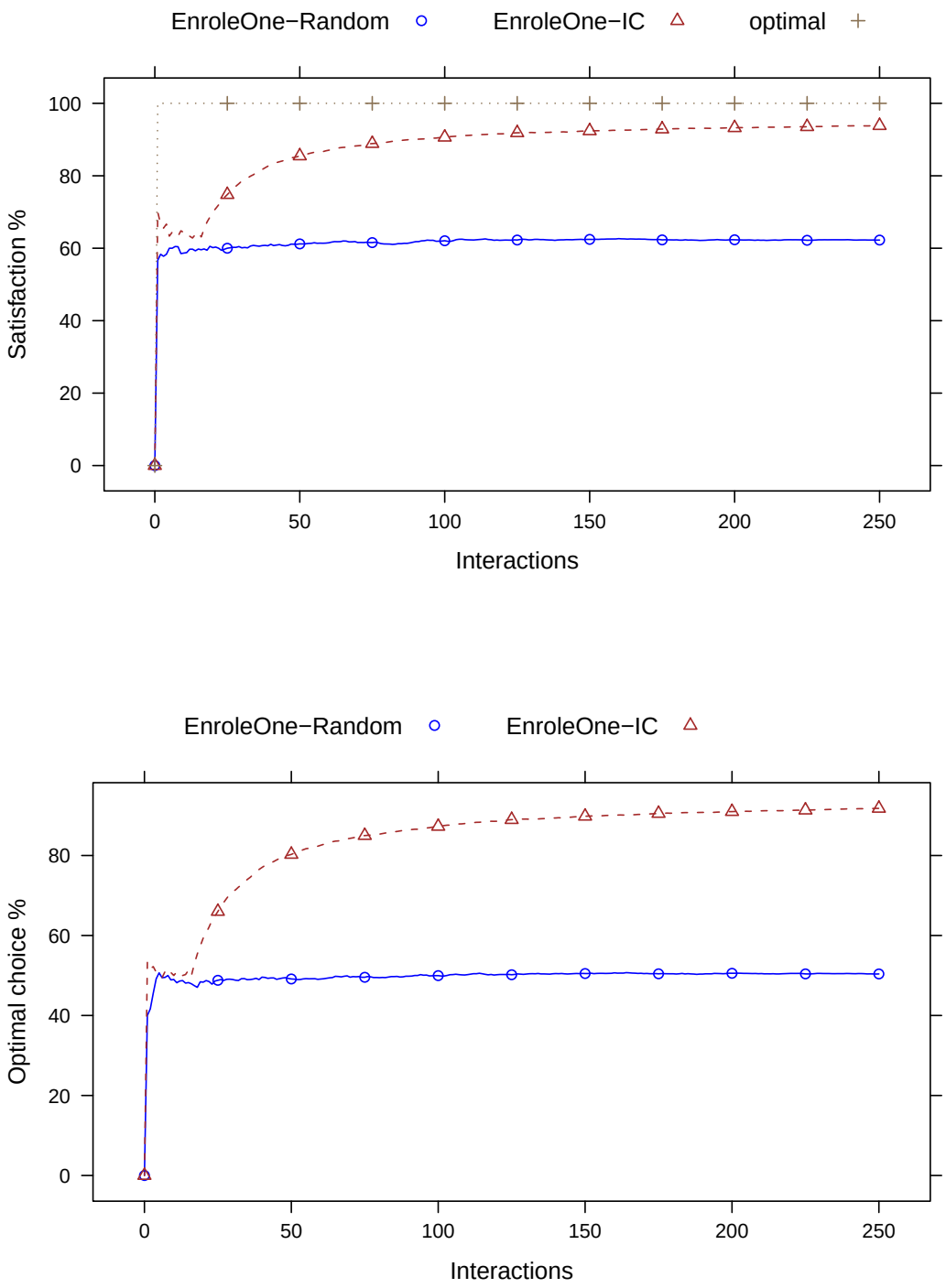
$$
\begin{array}{l}
a(\mathit{traveller}, \mathit{Traveller}) :: \\
\quad \mathit{book\text{-}holiday}(\mathit{Src}, \mathit{Dst}, \mathit{Start}, \mathit{End}, \mathit{Money}) \Rightarrow a(\mathit{travel\text{-}agent}, \mathit{Agent}) \\
\qquad \leftarrow \mathit{travel\text{-}details}(\mathit{Src}, \mathit{Dst}, \mathit{Start}, \mathit{End}, \mathit{Money})\ \mathit{then} \\
\left(\begin{array}{l} \mathit{booking}(\mathit{Start}, \mathit{End}, \mathit{Cost}) \Leftarrow a(\mathit{travel\text{-}agent}, \mathit{Agent})\ \mathit{then} \\ \mathit{matchmaking}(\mathit{good}) \Rightarrow a(\mathit{matchmaker}, \mathit{matchmaker}) \end{array} \right) \mathit{or} \\
\left(\begin{array}{l} \mathit{failure} \Leftarrow a(\mathit{travel\text{-}agent}, \mathit{Agent})\ \mathit{then} \\ \mathit{matchmaking}(\mathit{bad}) \Rightarrow a(\mathit{matchmaker}, \mathit{matchmaker}) \end{array} \right) \\
\\
a(\mathit{hotel}, \mathit{Hotel}) :: \\
\quad \mathit{book\text{-}hotel}(\mathit{Location}, \mathit{Start}, \mathit{End}, \mathit{Money}) \Leftarrow a(\mathit{Role}, \mathit{Agent})\ \mathit{then} \\
\quad \mathit{room\text{-}available}(\mathit{Location}, \mathit{Start}, \mathit{End}, \mathit{Money}, \mathit{Cost}) \Rightarrow a(\mathit{Role}, \mathit{Agent}) \\
\qquad \leftarrow \mathit{room\text{-}available}(\mathit{Location}, \mathit{Start}, \mathit{End}, \mathit{Money}, \mathit{Cost})\ \mathit{or} \\
\quad \mathit{no\text{-}vacancy} \Rightarrow a(\mathit{Role}, \mathit{Agent}) \\
\\
a(\mathit{airline}, \mathit{Airline}) :: \\
\quad \mathit{book\text{-}flight}(\mathit{Src}, \mathit{Dst}, \mathit{Start}, \mathit{End}, \mathit{Money}) \Leftarrow a(\mathit{Role}, \mathit{Agent})\ \mathit{then} \\
\quad \mathit{flight\text{-}available}(\mathit{Src}, \mathit{Dst}, \mathit{Start}, \mathit{End}, \mathit{Money}) \Rightarrow a(\mathit{Role}, \mathit{Agent}) \\
\qquad \leftarrow \mathit{flight\text{-}available}(\mathit{Src}, \mathit{Dst}, \mathit{Start}, \mathit{End}, \mathit{Money})\ \mathit{or} \\
\quad \mathit{no\text{-}flights} \Rightarrow a(\mathit{Role}, \mathit{Agent}) \\
\\
a(\mathit{matchmaker}, \mathit{matchmaker}) :: \\
\quad \mathit{record\text{-}matchmaking\text{-}outcome}(\mathit{Outcome}) \\
\qquad \leftarrow \mathit{matchmaking}(\mathit{Outcome}) \Leftarrow a(\mathit{Role}, \mathit{Agent})
\end{array}
$$

Figure 7.13 Alternative travel agent role definitions

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

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

Figure 7.14 Role selection



The top plot shows client satisfaction for EnroleOne-Random and EnroleOne-IC. The 'optimal' line shows satisfaction for clients which always preselect the optimal role definition (*flight-then-hotel*). The bottom plot shows how often the EnroleOne-Random and EnroleOne-IC select the optimal role. Results are averaged over 30runs.

7.4 Connections

In this section, we highlight the connections between our approach and work in other fields. We said earlier that our approach was comparable with the information retrieval model used for decades to find documents matching keyword queries. Having laid out the details of our approach, we can now make this analogy more precise. We also touch base with reinforcement learning, which informs how we balance exploitation of the database with the need to continue exploring new service combinations and revisiting others which may have changed their competence over time.

7.4.1 Information retrieval

Information retrieval (IR) (Baeza-Yates and Ribeiro-Neto 1999; Witten, Moffat, and Bell 1999) deals with the searching of a collection of documents for those which match a given query. The documents may be images, sounds, videos, or other media. Most commonly, however, they are text. Likewise, the queries may take different forms, but again the usual case is a short list of words. The most commonly used information retrieval systems today are web search engines, and they form both an inspiration for our approach and the touchstone for the comparison in this section.

In an information retrieval system, the user issues a query q, from a the set of all possible queries Q. The IR system locates an 'answer set' A_q of documents from the complete set of documents in its database, D. The query is a list of keywords which the user believes models the set of documents she is interested in. In the simplest case, the answer set A_q is the set of documents which contain all the keywords. A slight extension is to allow the user to specify keywords which should not be present in the answer set. A query is then a composite of the required terms q_+ and the tabu terms q_-. Such queries are known as 'boolean', since the terms must be either be present or not present.

The implementation of such a retrieval system is straightforward, at least at the abstract level, although optimisation is always a large and complicating factor in real-

world systems. Each document d in the set D is dealt with as a set of keywords, usually after some preprocessing to remove the most common terms like 'the', 'a' etc, and often to stem or conflate similar words e.g. transforming 'running' and 'ran' to 'run'. The system defines a 'lexicon' T of words or terms which will be indexed. Each document d is allocated a unique integer identifier, and for each unique term $t \in T$ in d, the identifier for d is added to the list of documents for t. The result is an 'inverted index'[3] for the keywords with pointers to the containing documents.

The similarity to our matchmaker is clear. If we substitute the document set for an interaction set and individual collaborations for keyword terms, we can see that the two indexes are essentially the same. Query construction is somewhat different: instead of requesting an answer set of documents for a single query, we build a set of queries where we consider a fixed core consisting of the current interaction's prior commitments to collaborators, and a variable component identifying each of the possible new collaborators for the unfilled role or roles under consideration. Further, the answer set itself is not of interest, but the relative size of the sets.

Although simple to understand, and the basis for our matchmaker, the boolean model is not much used in current IR systems, since it is too blunt a tool. Keyword matches are all or nothing, and do not account for multiple occurrences of terms in a single document, which would usually indicate a stronger match. Most modern IR systems, such as Google (Brin and Page 1998) are based on a vector space model, which measures the 'similarity' between query and documents $sim(\vec{q}, \vec{d_j})$. In boolean searches, documents (and queries) are represented as sets of terms, but they can equivalently be represented by $|T|$ dimensional vectors, where each dimension is associated with a term $t \in T$, and the value of a given dimension is 1 if the term is present and 0 otherwise. In a vector space model, this is generalised so that vectors are measured as real values, not as booleans. This enables multiple occurrences of a term to be recorded and used to measure relevance, and for matches to be less than 100%. A match is defined via some

[3]This being the conventional term, although it is not actually an inverted index, but an inverted file, or an index.

similarity metric between the vector standing for the query, and the vectors representing each document. The simplest measure is Euclidean distance:

$$sim(\vec{q}, \vec{d}) = (\sum_{t \in q} \vec{q_t} - \vec{d_t})^{(1/|T|)}$$

A more common measure is the angle between the query and the query and document (actually the cosine of that angle):

$$sim(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| \times |\vec{d}|}$$

This can be further refined by accounting for the relative frequencies of terms in the documents and the document collection. A vector representing d a document comprises weights $w_{d,t}$, that is, a measure of the prominence of term t in d. This weight is influenced by the intra-document frequency of t in d, or 'term-frequency' and denoted $tf_{d,t}$. $tf_{d,t}$ is usually the number of times t occurs in d ($n(d,t)$), and scaled. One might scale over all words in the document $n(d,t)/\sum_{t \in T} n(d,t)$, or by the term occuring most often $n(d,t)/max_{t \in T}(d,t)$

Analysis of the whole document collection D reveals which terms are most able to differentiate between clusters, and are thus of greater weight in the query. The 'inverse document frequency' (IDF) of a term idf_t is

$$idf_t = log\frac{|D|}{n_t}$$

where n_t is the number of documents in which t occurs. Finally, the term frequency and inverse document frequency are often combined to the TF-IDF measure, making the final weights for a document vector d:

$$w_{d,t} = tf_{d,t} \times idf_t$$

Information retrieval systems are most commonly compared by their 'precision' and 'recall' performance. Both of these are measured by comparing the document set A_q against a 'correct' answer D_q for that query. This ideal answer D_q is determined

manually when a benchmark problem is constructed. Recall is defined as the proportion of documents in the answer set that are relevant compared with the ideal answer:

$$recall = \frac{|A_q \cap D_q|}{|D_q|}$$

Similarly, precision is the fraction of answer set that is deemed relevant:

$$precision = \frac{|A_q \cap D_q|}{|A_q|}$$

Neither of these measures are applicable for us, since our answer set is, by definition, the correct set as determined by previous users. However, these metrics are sometimes used to evaluate the response of matchmakers, particularly in the service 'selection' phase. In particular, the IR approach of ((Klusch, Fries, and Khalid 2005)) uses such a scheme (see section 3.6), and it also appears in the semantic service selection (S3) challenge which tries to benchmark matchmakers[4].

In information retrieval, 'relevance feedback' is sometimes used to allow a user to refine their query. After an initial query q is submitted, the user reviews the resulting answer set, informing the system of which documents are more relevant. The system can then perform 'query expansion', adding new terms to the query q, or reweight the terms in q. An early method was 'Rocchio's method'. The revised query is a sum of the original query, plus a sum of the terms from the relevant documents R_+, minus a sum of the irrelevant documents R_-:

$$\vec{q'} = \alpha\vec{q} + \beta \sum_{\vec{d} \in R_+} \vec{d} - \gamma \sum_{\vec{d} \in R_-} \vec{d}$$

Such techniques are not seen on web search engines for two reasons. Web users are generally unwilling to spend much effort refining their searches, being generally happy with initial results. Secondly, query expansion can significantly slow the speed at which document similarity is computed, because it increases greatly the number of terms which must be compared.

[4] `http://www-ags.dfki.uni-sb.de/~klusch/s3/`

In our matchmaker, we rely on user feedback between queries, not during them. This is different from traditional IR, where a search engine's raw material is not affected by its own actions[5] We next look at this from the perspective of reinforcement learning, which deals exactly with the issue of an agent learning as it participates in an environment.

7.4.2 Reinforcement learning

Reinforcement learning (Barto and Sutton 1998) is one of the three principle sorts of learning, the others being supervised and unsupervised. In reinforcement learning, an agent participates in an environment, choosing actions which lead to rewards. The agent 'wants' greater reward, and attempts to learn to select those actions which provide higher rewards over the long term. The two key features which distinguish reinforcement learning (RL) from other kinds of learning are also present in our matchmakers:

1. The learner is an active participant in its environment, and must therefore strike a balance between exploiting its current knowledge to improve immediate performance, and exploring alternatives, which may lead to improved long term performance.

2. Correct decisions are characterised by an evaluation function, not by instruction. That is, the learner's choice of actions result in a value related to its performance. It is rewarded based on the value of its action sequence: it is not told explicitly what actions sequence it should have produced.

The first is clearly true since our matchmakers are learning the best selections as they participate with clients and services in ongoing interactions. The second property is satisfied because the matchmaker is told how well its selections worked for the client, not which selections it ought to have made. Indeed, we cannot know which services

[5] At least to a first approximation. It's certainly possible, and probably the case, that users choose to create links to other pages based in part on the search results they see. This biasing caused by search engines has been dubbed the 'Googlarchy', but at least one study has shown search engines actually lead to less prominent sites receiving *more* traffic than they would by pure browsing (Fortunato et al. 2005).

should have been selected, since that information is not available to any agent in the environment.

A reinforcement learner receives a *reward* for each *action* it takes from each of a series of states. The interaction occurs at discrete time intervals t. At each point in time t, the learner is given a representation of the current state s_t, and must choose an action a_t. The agent receives a numerical reward r_{t+1} based on the new state which was caused, to a greater or lesser extent, by its action a_t. Using these values, a learner constructs a policy π, a probabilistic mapping from states to actions. The value function for a policy $V^\pi(s)$ gives the expected long term reward (or result) from pursuing policy π from state s. The action-value function $Q^\pi(s, a)$ gives the expected return from taking action a in state s, and following continuing to follow policy π. For our matchmakers, the states are LCC interactions with unassigned roles[6], and the actions are selecting services to meet those roles. In our case, a real reward is issued only after the interaction has completed, but this is solved by simply making all but the final reward 0. Our matchmaking task is of a kind known as 'episodic', because the sequence of states can be broken into sub-sequences, each marking the end of a sub-task. The episodes correspond to the matchmaking activities related to a single interaction.

A learner attempts to improve its policy by progressively approximating the optimal policy π^*, and does so by updating its value and value-action functions based on experience. The simplest way to approximate Q^* is to take the mean reward for a given action:

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_a}}{k_a}$$

Keeping and computing this can be computationally very expensive and thus is usually avoided in RL, but it is exactly what we do in our matchmakers. For us, the computational cost is not so great: most of the rewards are zero and can be ignored. Moreover, by keeping a database of interactions, we keep open the possibility of performing other analysis offline (possibilities are discussed in chapter 9), and allow clients direct access

[6]More precisely, just the mappings from roles to services, assigned or not.

to the data in the event they wish to use their own policy. Another shared interest with RL is the importance of the trade-off between exploiting our current knowledge and exploring the consequences of actions we have not tried before. Our solution so far has been our so-called 'argmax' algorithm. In RL this approach—called 'ϵ-greedy'—is just one of many, and we will take a look at them in the next chapter.

7.5 Summary

In this chapter we extended our approach from the matchmaking for a single agent to matchmaking several collaborations in a single interaction. We demonstrated the applicability of a simple, comprehensible technique to the problem of achieving satisfactory coherence between the participants in an interaction. This is the core contribution of the thesis.

- Illustrated through the Astrogrid scenario the importance of inter-agent performance.
- Demonstrated an improvement in user satisfaction over random selection by using historical performance data.
- Showed how full interaction histories and the incidence calculus improve over the simpler and algorithmically cheaper naive Bayes approach.
- Applied the same basic technique to automating the construction of interaction models at run time, based on the previous success or otherwise of components of an overall workflow.
- Discussed connections between our approach and those of information retrieval and reinforcement learning.

Chapter 8

Scaling

Over the last three chapters we introduced a simple mechanism for tracking and predicting the performance of agents as they participated in multi-party interactions. We looked at illustrative, small-scale examples involving a single interaction model and a handful of agents. In this chapter, we investigate what happens on a larger stage, with many agents, roles, and interaction models. We pursue this through simulations, rather than by empirical study of real semantic web services. We are unhappy with this, but see little alternative, given the lack of deployed semantic web services systems, and the difficulty likely to be encountered in augmenting a semantic services platform with such a scheme. Moreover, the number of user ratings to be captured is not insignificant, and would be difficult to achieve in a small scale study. Since our thesis states that formal models of services cannot reasonably capture all aspects real world use, we can hardly now claim to be able to evaluate in a precise way how our matchmaker would fare in practice. We do show that it will scale effectively, and that, for reasonable and conservative parameters of likely interaction problems, it offers worthwhile improvements in finding agents which work well together.

8.1 Simulation

There are very few semantic web or agent services openly available. Those that do exist have been carefully constructed so as to work together, as we argued in chapter 4. Consequently, we create a synthetic environment to evaluate our matchmaking approach. Our simulation model is not intended as a comprehensive model of web services now, or in the future. Although we construct our simulation based on the values of some easily tracked parameters (ontology for instance), we do not believe that all such values can be easily encapsulated.

8.1.1 Evaluation model

For the purposes of our scaling experiments, we adopt four notions:

- *Interoperability*, representing an agent's choice of ontology and implementation platform and the resulting incompatibility with agents that make other choices
- *Intrinsic ability*, representing an agent's core competence at its task
- *Random noise*, a random value between 0 and 1, which is added to an interaction score to account for the variance in users' opinions, network outages, and so on
- *Good/bad threshold*, a real value between 0 and 1, which determines whether the computed score of an interaction should be recorded as good or bad

These values are strong simplifications of the real environment where there would be many other such factors, including geographical locality (for maps, weather, restaurant ratings), and network limitations (bandwidth, line of sight or atmospheric issues in sensor networks). Since our matchmaking scheme is intended to work without regard to the causes of the incompatibilities, we believe such a coarse model is acceptable. Adding more variables would not increase the fidelity to an (unknown) reality, but would remove clarity from the experiment and results.

We define an agent A as a tuple $\langle \mathcal{I}, \mathcal{A} \rangle$ where $\mathcal{I}$ is the implementation and $\mathcal{A}$ the intrinsic ability. The value of $\mathcal{I}$ is a discrete value standing for the agent's choice of ontology and platform, while $\mathcal{A}$ is a real value between 0 and 1. These properties of the agents are known only to an oracle, and not to the matchmakers. An interaction I is defined as a tuple $\langle M, R, C \rangle$ with interaction model $\mathcal{P}$, requesting agent R, and collaborating agents C.

In an experiment, the matchmaker must take an interaction and recruit appropriate services to each role. When all roles are filled, the experimental harness asks the oracle for a verdict on the quality of the collaboration. The score is defined as

$$\Delta_{\mathcal{I}}^{|\mathcal{I}(I)|-1} \times \min_{s \in collaborators(\mathcal{I})}(\mathcal{A}(s))$$

The $\Delta_{\mathcal{I}}$ is a simulation parameter which determines the quality of interaction between services with different 'implementations'. When $\Delta_{\mathcal{I}}$ is 1, we assume full compatibility, and at 0, total incompatibility between agents with different values for their implementation value. The set of implementations used by all collaborators in an interaction I is denoted $\mathcal{I}(I)$, and so the number of distinct implementations used in an interaction is $|\mathcal{I}(I)|$. To get to the final good/bad determination, we add some noise to this oracular score, and compare to the threshold:

$$outcome = \begin{cases} good \text{ if } \left(\Delta_{\mathcal{I}}^{|\mathcal{I}(I)|-1} \times \prod_{s \in collaborators(\mathcal{I})} \mathcal{A}(s) + noise\right) \geq threshold \\ bad \text{ otherwise} \end{cases}$$

The $threshold$ itself is defined as 90% of the score of the optimal set of available agents for the interaction.

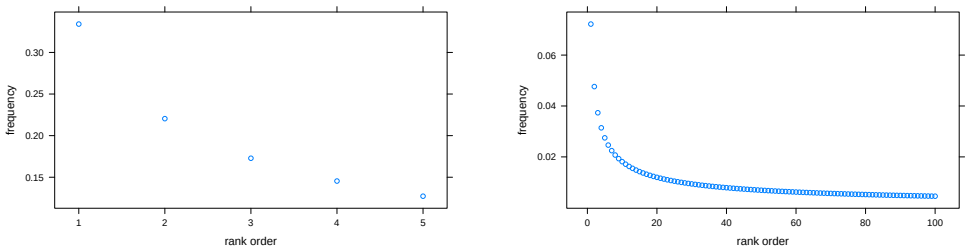
How should the values representing implementations be distributed? It is reasonable to expect that a few ontologies will claim most attention, while there will be many others less popular but which still find happy users in their niches. These kinds of patterns are known as 'power-law' distributions, and are often seen on the Web (Baldi, Frasconi, and Smyth 2003). We choose to use the Zipfian distribution, a particular class of power-law,

which is defined

$$f(k; s, n) = \frac{1}{k^s} H_{n,s}$$

where $H_{n,s}$ is the nth harmonic number. The parameter s controls how flat the distribution is, while n is the total number of terms being considered. Figure 8.1 shows Zipfian distributions with $s = 0.6$ for both plots, and $n = 5$ and $n = 100$ respectively.

Figure 8.1 Zipfian probability distributions



8.1.2 Experimental setup

Until now, our simulations were performed with a bona fide LCC interpreter which used a Prolog engine implemented in Common Lisp to fully expand the interactions according to the rewrite rules mentioned in chapters 6 and 7. Executing workflow languages is unnecessarily expensive in time when our primary aim is to score our matchmakers' interactions—especially since our own interpreter emphasises clarity over quickness. While we could optimise the interpreter to some degree, in this chapter we choose to discard workflow interpretation entirely in order to evaluate much larger simulations of many thousands of interactions. Instead of explicit models, our interactions become simple sets of roles and collaborators.

This simplification enables us to study larger simulations. How large should they be? One is tempted to look at the web to gauge an appropriate scale. When Google first

went public, it indexed 26 million pages, and now covers a substantial fraction of the trillion URLs Google has found[1]. But the HTML web is not a legitimate comparison, not least because pages are not services. The start-up company seekda[2] crawls the web for WSDL, and as of early 2009, they claim 28000 services from 7000 providers. Rough numbers from bioinformaticians claim around 3000 services in that domain (Hull et al. 2006), but not all of those are web services.

In related research discussing matchmakers based on subsumption, we usually see evaluations where there are perhaps a few hundred services. As in our case, most or all of these services are synthetic, although they are often based on a smaller set of 'real' descriptions which are then cloned and randomly manipulated by programs to provide a larger set for testing purposes. Numbers of services across those papers detailing their matchmaking experiments are

- 3 to 20 providers in the WARREN system (Decker, Sycara, and Williamson 1997)
- 350 in the MX matchmaker (Klusch, Fries, and Khalid 2005; Kaufer and Klusch 2006)
- 100 to 1500 in another OWL-S matchmaker (Li and Horrocks 2003)
- up to 2000 in a WSMO matchmaker (Stollberg, Hepp, and Hoffmann 2007)

This does not translate directly into our own simulation, for there is an important difference: in those systems, each query to find a suitable services was executed only once[3], since the system as a whole is stateless. The query is made, a match is found, and an evaluation made. For our experiments, the same basic query—'find me a service of this type'—must be made many times. The evaluation of that match will happen many times, giving slightly different results because of the random noise we add, but more importantly, the matchmaker's selection will vary, because it is learning to improve its

[1] `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`
[2] `seekda.com`
[3] The actual tests are usually run several times to obtain statistical stability in the timing, but in principle, each is run only once.

response. While individual queries are very fast, running them hundreds of thousands of times is not.

How long does a matchmaking operation take? All the systems referenced above can matchmake a single role in under one second. According to (Miles et al. 2003), normal UDDI queries took 190ms, while using a UDDI directory enriched with RDF increased the time to about 210ms. The numbers in (Stollberg, Hepp, and Hoffmann 2007) are similar: "average time for a single matchmaking operation has been 115 ms, and 15 ms for the discovery-by-lookup procedure". We will give precise timings for our scheme in section 8.3, but we also answer individual queries in under one second. Individual queries in the neighbourhood of one second do limit the size of our simulations, even though they would be perfectly adequate in a real-world deployment where the matchmaking database would be built and used over days, months, and years.

8.2 Tuning

Before looking at the final macro performance of our system, we briefly look at how we can improve the micro performance of the matchmaking operations. We look at the basic cost of the incidence calculus operations, which lie at the core of our method, and show how they can be made faster. We then briefly present other optimisations that could be made in a deployed system.

8.2.1 Incidence calculus operations

At the core of our approach is the computation of the incidence sets, so it is important that they be efficiently computable. We store the incidence sets as ordered lists of integers, and intersecting multiple lists can be straightforwardly implemented as a sequence of pairwise intersections. Doing this is $O(n)$, for sets of n worlds. It helps that, as more sets are intersected, the candidate sets to be processed usually become significantly smaller. It is a standard optimisation for IR systems to keep the list of sets

yet to be processed sorted in increasing size (Witten, Moffat, and Bell 1999). Figure 8.2 illustrates performance for sizable sets with and without this optimisation.

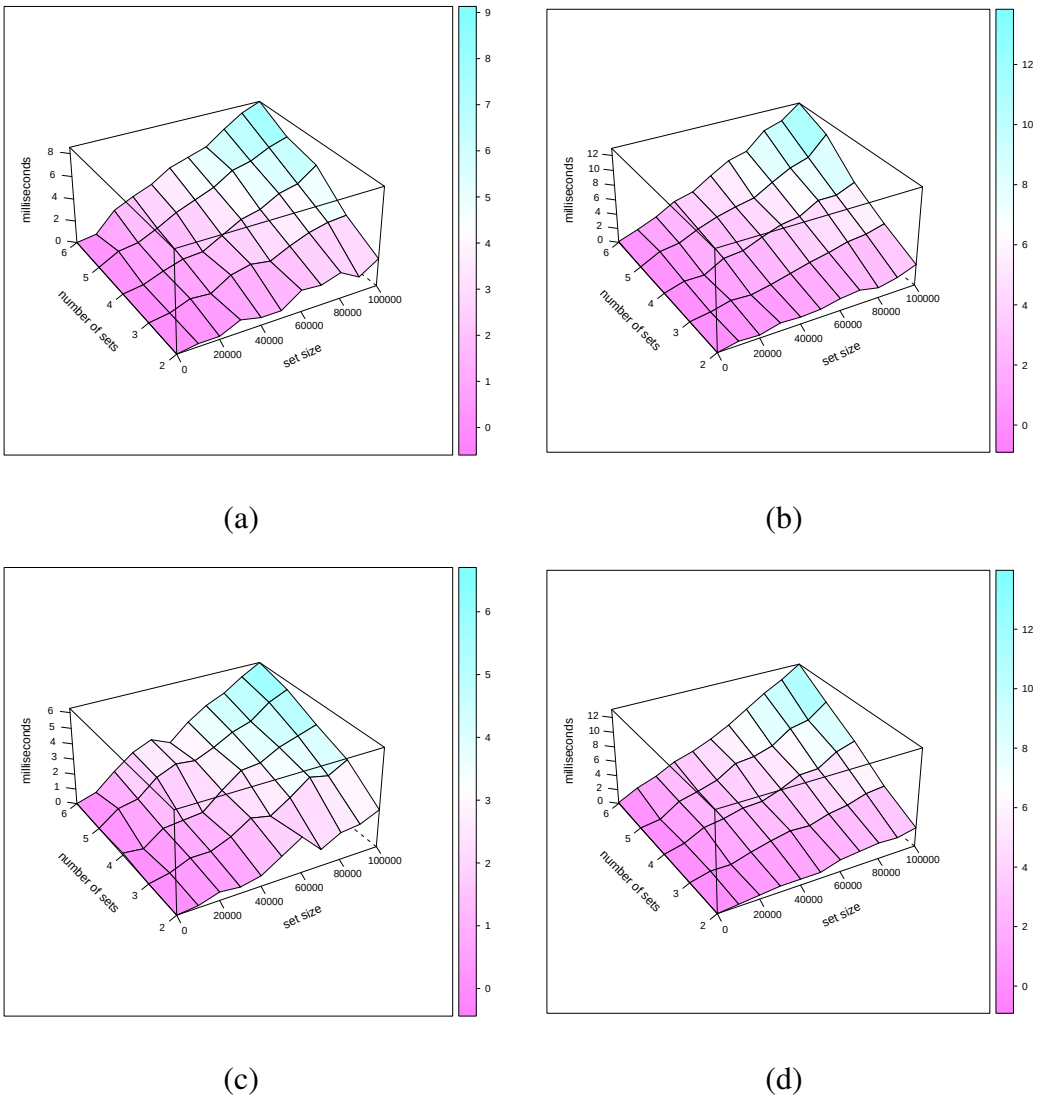
A bigger impact can be made by reconsidering how we compute the probabilities. Up to now, we did this using the standard incidence calculus definition, which requires that we intersect the full list of good and bad outcomes with the lists of the various role/service pairings:

$$p(outcome(good)|\ collaborators) = \frac{|i(outcome(good)|\ collaborators)|}{|i(collaborators)|}$$

Since *every* interaction is recorded in either the $outcome(good)$ or $outcome(bad)$ lists, intersecting with them is slow. If we instead determine only the set $i(collaborators)$, we can go through this set and individually count how many of the interactions contain $outcome(good)$. Remember from 6.3 that we store each interaction individually, as well as indexing them for the incidence calculus operations, so going through the set of interactions is straight-forward. Moreover, this approach allows us to make a further important optimisation. We can now limit our search to only the most recent interactions for which $collaborators$ is true, imposing a 'horizon' on our search. Again, this is a frequently used optimisation in web-scale information retrieval systems, where on the order of ten thousand pages are found in a batch and then ranked (Brin and Page 1998). Similarly, we can look for the last 50 or so interactions using the set of collaborators we are considering, and see what proportion of those led to a good income. An added benefit is that we can now account naturally for changes in services' behaviour, since we are interested in how the service performed recently, rather than considering its entire performance history. Although not exploited here, the use of a horizon instead of immediate intersection with the $outcome$ allows the use of other metrics for output. For instance, a real number score can now be assigned to each interaction, and the sum taken of those interactions returned within the horizon.

We implemented the horizon operations at the incidence calculus query level. We added MATCHMAKEHORIZON-ONE and MATCHMAKEHORIZON-ALL matchmakers which use the same algorithm as MATCHMAKEIC-ONE and MATCHMAKEIC-ALL

Figure 8.2 Set intersection operations



Plots (a) and (b) show intersection performance respectively for sets of randomly selected incidences and full sets. A full set of n instances is the set of instances $\{1, 2, 3, 4, \ldots, n\}$), while the random sets are constructed by choosing the first n integers each with $p = \frac{1}{2}$. Plot (c) shows the moderate increase in performance achieved by keeping the incidence sets sorted in increasing set size. Plot (d) shows no improvement since the sets are always the same size. All measurements are averaged over 30 runs. Intersections of 6 sets of 100000 random incidences typically return 700 to 800 incidences. As discussed in section 8.4, it becomes difficult to use more than about four terms in one query, precisely because the resulting datasets become very sparse.

respectively, but which use the modified incidence calculus functions.

8.2.2 Possibilities

Our implementation, using normal Common Lisp lists, is bordering on the naive, but still performs reasonably quickly. We are confident that using the implementation techniques of Web scale IR systems would offer a considerable constant factor improvement through both high and low level optimisations. In the remainder of this section, we sketch some of those techniques, and discuss the use of other algorithms from reinforcement learning that might perform better than the ϵ-greedy algorithm used here.

Caching An obvious optimisation is to cache results. The most common queries will likely be made many times by different clients, and by caching the most frequent queries, we can save substantial computational effort. This is commonly used by Google and other web search engines for the most common searches.

Parallelisation The wall time required for a matchmaking operation can be reduced by parallelising the queries. There are at least two ways we can exploit parallelism. First, when we query the database to find the best collaborator:

$$\mathrm{argmax}_{\mathcal{C}} P(outcome(good)|I, \mathcal{C} \cup collaborators(I), matchmaker)$$

we create several queries for each set of collaborators implied by the choices available in the new collaborators $\mathcal{C}$. We can execute all of those queries in parallel. Secondly, every interaction has an associated 'possible world' in the incidence sets, and because we never consider predicates across worlds, we can split incidence sets up according to the time frame, or epoch, in which they occurred. That is, the first n interactions would be considered the first epoch, the second n interactions would be the second epoch, and so on. Each epoch might cover a billion incidences, and could be assigned to a separate machine. A query for a particular set of services can be dispatched to each machine, and the final numerical counts collated easily and quickly. This stratification also lends itself

to progressive deepening, in conjunction with the horizon search, so that only the most recent epochs need be examined in most cases, with the search going further back in time when insufficient recent data is available.

Fairness and exploration After the incidence calculus-based algorithms have computed the historical performance of the various combinations of agents, our matchmakers must decide whether to select the combination that has the highest score, or to try one of the others. This balance between exploiting current knowledge and continuing to explore new options is important for a matchmaker like ours, and the field of reinforcement learning has developed several alternative 'action selection' functions. The optimal action selection function can be decided only with detailed knowledge of the problem, and in our case, we do not have this to hand (Barto and Sutton 1998).

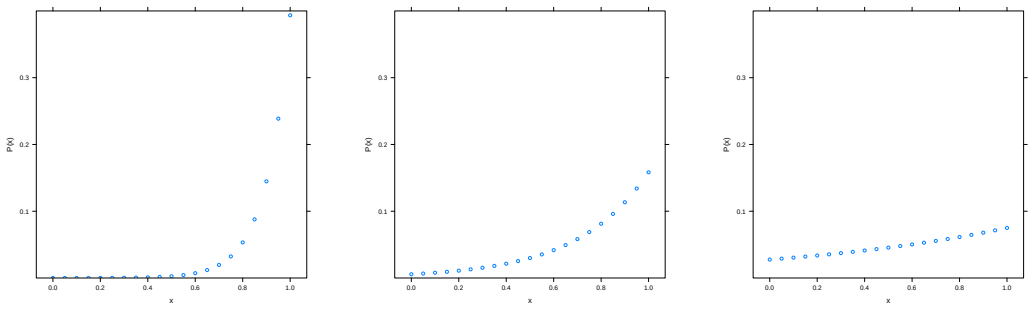
A primary alternative to the ϵ-greedy selection used in this this thesis is 'softmax'. ϵ-greedy usually selects the highest-scoring action, and with probability ϵ randomly selects amongst all available actions with equal probability. Softmax also usually selects the the highest-scoring action, but its choice of other actions is weighted by their score, so that the second-highest scoring action is selected more frequently than third, and so on. One common way to compute the weighting is a Boltzmann distribution parametrised by a 'temperature' which can be used to tune the weighting. The probability of selecting the action a amongst the A available at a temperature t in a Boltzmann distribution is

$$P(a) = \frac{e^{Q(a)/t}}{\sum_{b \in A} e^{Q(b)/t}}$$

Figure 8.3 shows Boltzmann distributions for several temperature values.

Another common technique is to provide optimistic initial values. Instead of services beginning with an empty history or zero score, they begin with a high score that is gradually reduced by poor performance. This would transparently favour exploration of newly advertised services.

Figure 8.3 Boltzmann probability distributions



From left to right, Boltzmann probability distributions for $t = 0.1$, $t = 0.3$, and $t = 1$. The lower the temperature t, the stronger the preference (higher $P(x)$) for better performing actions (represented here by higher x values).

8.3 Experiments

In this section we present the results of our larger-scale experiments. We considered a total of 16 scenarios, running each one with 30000 interactions, and each over 30 runs for statistical stability. The sixteen scenarios explore three dimensions of the problem space: number of roles per model, the interoperability between agents, and the intrinsic ability of individual agents. The scenarios used the following values:

parameter	values
roles per model	2, 3
interoperability $\Delta_{\mathcal{I}}$	1, 0.95, 0.9, 0.8
intrinsic ability $\mathcal{A}$	1, $U(0.8, 1)$

The notation $U(x, y)$ denotes the uniform probability distribution between x and y, inclusive. For each scenario we create 300 services, assigning each service one of 30 'implementation platforms' according to the Zipf distribution mentioned earlier. In each scenario we create 100 models using either 2 or 3 roles depending on scenario. Each service is assigned one role, and we evenly apportion them, so that each role has 3 services which can fulfil it.

Figure 8.4 shows the plots for the simulations with models having two roles requiring matchmaking, and figure 8.5 for models with three roles to be filled. The lines plots the proportion of good outcomes (as defined in section 8.1.1), with a sliding average covering the last 500 interactions.

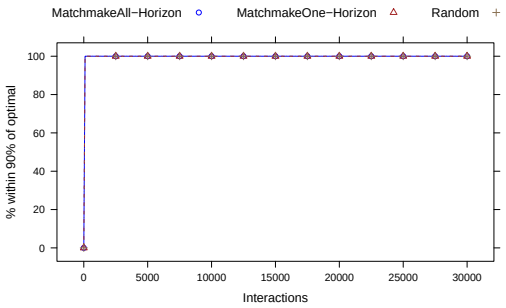
As before, not too much is to be read into the exact values obtained, but we can see that for even for small deviations from perfect intrinsic ability and interoperability, our simple matchmaking algorithms offer significant gains over random choice between ostensibly suitable services. Again, MATCHMAKEONE-HORIZON improves more rapidly than MATCHMAKEALL-HORIZON, but reaches a lower final level of performance.

The time required for an individual matchmaking operation is usually below the measurable threshold of 4 millisecond on our platform[4]. Some matchmaking operations register above 0ms, but never above 50ms, and these longer times can be attributed to garbage collection in the system. Measuring the run-time for entire simulations, we can infer an average of 7ms per operation over the 30000 interaction experiments, but this includes the time taken by the simulation framework and garbage collection in addition to the core matchmaking operation.

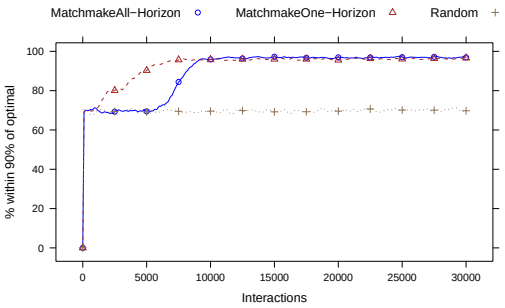
[4]A laptop with an Intel Core 2 Duo 2.40GHz CPU and 4GB RAM, running Linux 2.6.30 and SBCL 1.0.31.

Figure 8.4 Simulations with two-role models

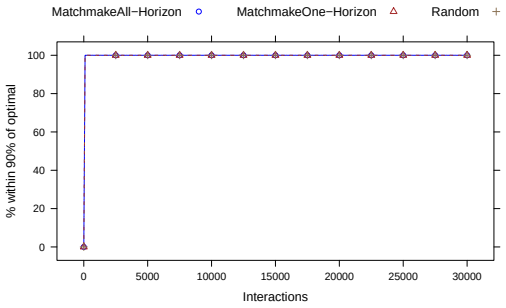
$\mathcal{A} = 1, \Delta_{\mathcal{I}} = 1$



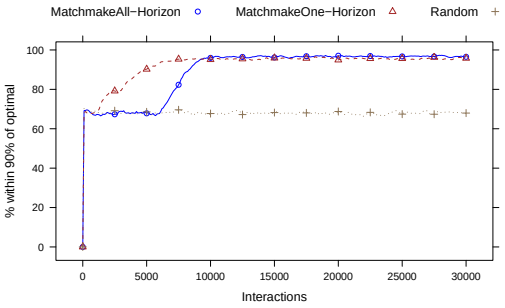
$\mathcal{A} = \mathrm{U}(0.8,1), \Delta_{\mathcal{I}} = 1$



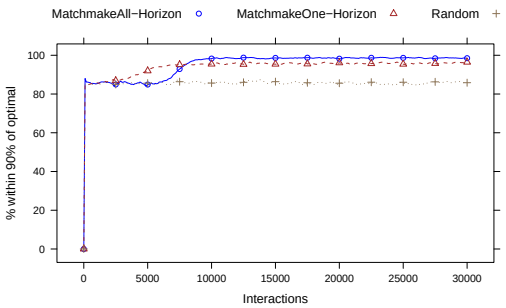
$\mathcal{A} = 1, \Delta_{\mathcal{I}} = 0.95$



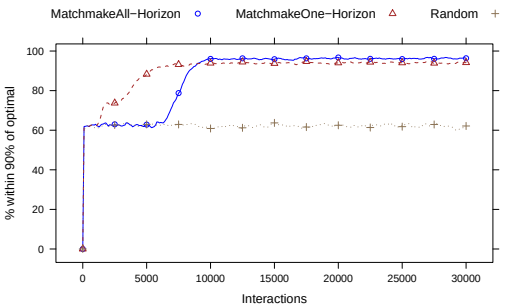
$\mathcal{A} = \mathrm{U}(0.8,1), \Delta_{\mathcal{I}} = 0.95$



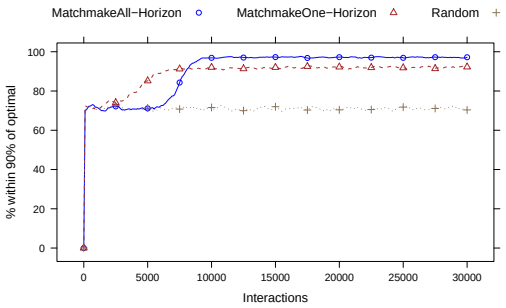
$\mathcal{A} = 1, \Delta_{\mathcal{I}} = 0.9$



$\mathcal{A} = \mathrm{U}(0.8,1), \Delta_{\mathcal{I}} = 0.9$



$\mathcal{A} = 1, \Delta_{\mathcal{I}} = 0.8$



$\mathcal{A} = \mathrm{U}(0.8,1), \Delta_{\mathcal{I}} = 0.8$

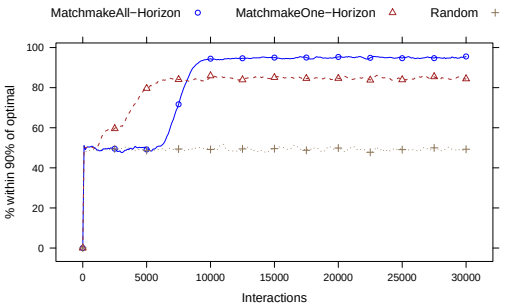
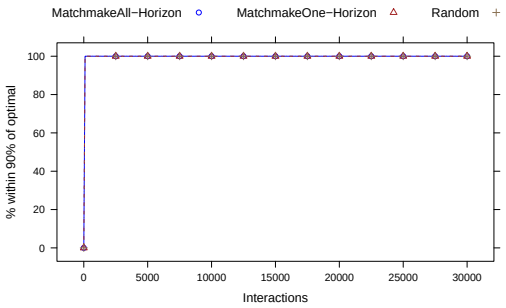
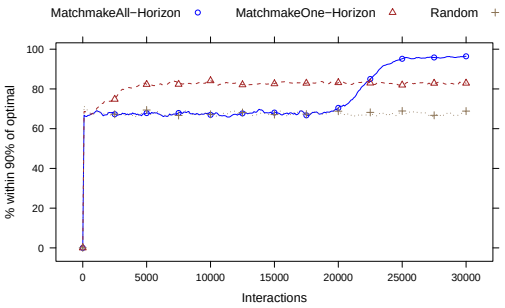


Figure 8.5 Simulations with three-role models

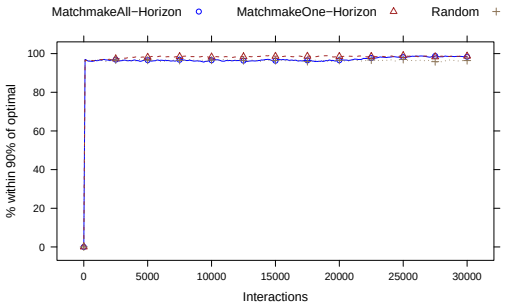
$\mathcal{A} = 1, \Delta_{\mathcal{I}} = 1$



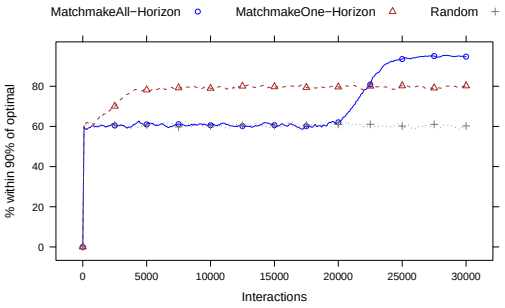
$\mathcal{A} =$U(0.8,1)$, \Delta_{\mathcal{I}} = 1$



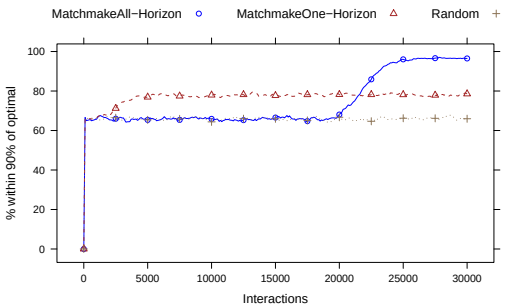
$\mathcal{A} = 1, \Delta_{\mathcal{I}} = 0.95$



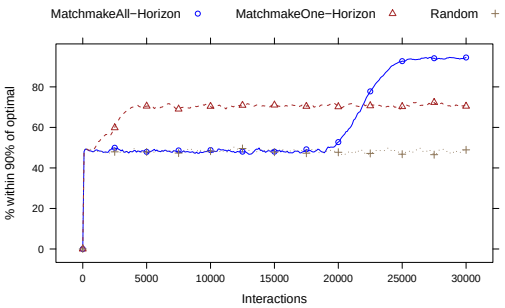
$\mathcal{A} =$U(0.8,1)$, \Delta_{\mathcal{I}} = 0.95$



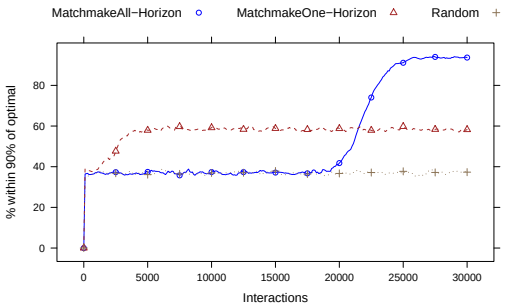
$\mathcal{A} = 1, \Delta_{\mathcal{I}} = 0.9$



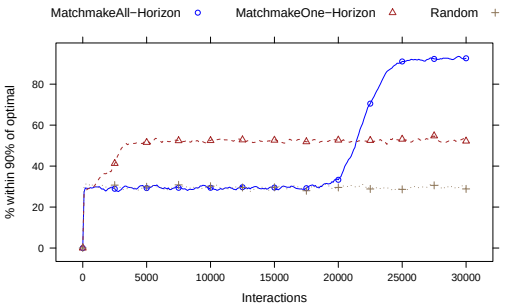
$\mathcal{A} =$U(0.8,1)$, \Delta_{\mathcal{I}} = 0.9$



$\mathcal{A} = 1, \Delta_{\mathcal{I}} = 0.8$



$\mathcal{A} =$U(0.8,1)$, \Delta_{\mathcal{I}} = 0.8$



8.4 Discussion

What can we conclude about the scalability of our algorithms? The experiments in the last section showed that individual queries comparable to those in the matchmaking literature can execute very quickly, and that we can build databases of large numbers of interactions. Furthermore, we have an existence proof of our approach in the web search engines that daily handle millions of such queries.

There is one genuine issue with the scalability of this approach, and it is caused by the interaction of lots of services to consider for each role, and the number of roles in an interaction. For an interaction with a set of roles R, and with each role nominally being fulfillable by a service $s \in S_r$, the number of collaborator sets to consider is

$$\prod_{r \in R} |S_r|$$

Each matchmaking request from a client expands to a set of queries to the incidence calculus database of this size, making scaling in this way $O(|R|^{|S|})$. The real problem lies not in our approach but in the domain: not only can we not build a system to deal with it efficiently, but there can never be enough interactions to fully explore such a space. This is the common 'curse of dimensionality' (Bellman 1961).

We can dodge the problem, at least for the role count, by using a technique from statistical natural language processing: 'n-grams'. In the task of predicting the next word in a sequence, the preceding words in the phrase are used as a key into a corpus, and the most likely following words in the corpus can be found by finding the maximum likelihood estimator

$$\text{argmax}_{w_n} P(w_n | w_{n-1}, w_{n-2}, \ldots)$$

Just as in our case, considering every word in the sequence rapidly leads to very sparse data sets. By making a 'Markov assumption' that words at a particular distance are independent, the length of phrase to be considered is greatly reduced. In a bigram model, only pairs of words are learned, leading to predictions of the next word w_n based on just the current word, w_{n-1}. In trigrams, w_n is predicated based on the preceding

two words w_{n-2}, w_{n-1}, and so on. Typically, the n in the n-gram does not exceed four, and even this requires a training corpus of tens of millions of words (Manning and Schütze 1999, p201). Equating words with collaborations, we can see that the same approach can be taken with the matchmaker. That is, we can assume that the effects of interactions between services 'fade out' as the data passes from service to service, so we need only consider services which interact closely with other services in a given model. Less obvious is the mapping from the linear sequence of words to the set of collaborations. In the simplest case, we can use the order in which roles were filled. In MATCHMAKEALL-* algorithms, we can use the arbitrary order of role definitions in the model. Neither of these correspond well to any sensible notion of proximity. In a system with an explicit dataflow accessible to the matchmaker, such as LCC, we can determine the flow of data between services, and use that to determine the nearest sources of any given data, and use those as the priors.

On policy choice Although we would ideally like to leave policy selection to each client, the interaction of the policies and the database can bring limitations. In particular, the MATCHMAKEALL algorithm cannot straightforwardly reuse a database built with MATCHMAKEONE. For a model with roles R, MATCHMAKEALL will always select an agent for every role in R, while an interaction using MATCHMAKEONE may matchmake only some $R' \subset R$, so that RECRUIT-ALL cannot form a joint probability distribution over R, since there are no entries for the roles $R \backslash R'$.

As we have seen in the various experiments in chapters 7 and 8, MATCHMAKEALL will outperform MATCHMAKEONE, but in real-world cases, MATCHMAKEONE may well be preferable as the more robust: if roles frequently go unfilled, total work on the broker would be reduced, and the matchmaker would not need to record the assignments of agents that were never actually used. In any event, because we have chosen to record a database of interactions, we can expose some or all of this to the client at selection time, so that they can fine tune their own selection.

8.5 Summary

In this chapter, we explored the performance of our matchmaker at large scale, using simulated environments. We:

- Presented a simple model of a large scale environment, representing individual agents through a small number of parameters.
- Introduced some optimisations to the basic algorithms which offer significant improvements in speed.
- Outlined potential optimisations for both the set intersection operations through parallelisation, and to the algorithms determining final service choice.
- Provided experimental data from simulations which show the method scales acceptably for the sizes of problems typically suggested in the matchmaking literature.
- Discussed the experimental results, concluding that the scaling properties of the algorithms mean they could be a valid means of augmenting matchmaking systems.

Chapter 9

Conclusion

In this final chapter, we highlight the scientific contributions of the work, and finish with a look at a open questions and suggestions for future research.

9.1 Contributions

In this thesis we critiqued the view of service matchmaking as simply a matter of matching input and output types, asserting instead the importance of actual outcomes in agent selection. We began our argument by surveying the background to the service composition and selection problem, highlighting the emphasis on formal methods for describing the capabilities of services. We then looked at several matchmaking frameworks and matchmakers, from distributed computing, through multi-agent systems, ending with today's semantic web services, noting that they all focus on the service advertisements and requests. This prepared the ground for our own contributions in the following chapters, where we:

- Outlined reasons to question the efficacy of purely logic-based selection of services based on service capability advertisements.
- Introduced the existing formalisms of LCC (Robertson 2004) and incidence calculus (Bundy 1985; Bundy 1992), and tied them together to produce a

framework for multi-party matchmaking.

- Showed how statistical information about service invocation histories can be used to select agents in two-party LCC interactions, and reviewed earlier history-based matchmaking work in the context of this formalism.
- Extended the method and matchmakers to support the selection of several agents in a single interaction.
- Showed how the same technique can be select roles in order to extend a model at run time.
- Evaluated the technique in simulations of large numbers of interactions, using synthetic services and interactions with plausible interaction impedances.

We believe the key contributions of this thesis to be

1. The argument against seamless interoperability between services based only on the limited formal service descriptions provided by by service builders.
2. The view of interactions themselves as significant entities worthy of study. In particular, we showed that interactions can be usefully modelled using information retrieval techniques, in the same way search engines record and process Web pages.
3. The introduction of simple matchmaking algorithms which can exploit interaction databases to improve over time their selection of services which are individually better performing, and together more interoperable.

Our proposed approach is not technically complex, but we believe that it provides a useful starting point for exploring and exploiting the information about service composition that is available to the users of those services, rather than their advertisers. We implemented our prototype matchmakers using LCC, but the approach would have value in any setting where service workflows are specified in a flexible, semantic

encoding. Consequently, we can see applications in multi-agent systems, web services, and on the Grid. It is these areas, particularly web services, where we can place our contributions in perspective. Recent trends in service provision point in the same direction as our own work:

- *Looser semantics* Both in the move from SOAP to REST services, and in the promotion of lightweight semantics for service descriptions through SA-WSDL (Farrell and Lausen 2007), MicroWSMO (Kopecký et al. 2008), and WSMO-Lite (Vitvar, Kopecký, and Fensel 2008), it is clear that whatever services and descriptions we see in the near future will have less exactness in typing than matchmaking research has hitherto presupposed.

- *Explicitly multi-party workflows* Matchmaking has focused on simple client-server interactions, where only one service is required. In practice, many scientific workflows require numerous participants, while 'mashups' of multiple services are becoming common way of providing new Web applications. Moreover, there is a growing understanding that the workflows themselves are important artifacts to be treated as first-class objects (Wroe et al. 2007).

- *User feedback* Interaction almost defines the Web 2.0 phenomenon, and users are now accustomed to providing ratings for almost everything, from blog articles and books, to restaurants and professors. Google's Android Market prominently features users' ratings of programs. The myExperiment project created a website where scientific workflows users can share the workflows, and comment on them. This suggests that users are not only interested in workflows per se, but are eager to share them and contextual information about them.

Taken together, these trends would seem to support the case for matchmaking techniques like ours. As Google researchers recently argued:

> . . . invariably, simple models and a lot of data trump more elaborate models based on less data.
>
> (Halevy, Norvig, and Pereira 2009)

If it is the case that users will trade certainty for deftness, the occasional wrong answer for the chance to make one-off choreographies that usually end up working, then matchmakers like ours would be a useful addition to the middleware landscape.

9.2 Future work

The work presented in this thesis has been based on plausible simulations of agent systems. While valid in itself, it is not possible to fully explore the possibilities of the presented approach, let alone tune our algorithms and implementations, in the absence of real-life experience of large numbers of users of such systems. Future work, therefore, would hopefully exploit empirical data regarding the behaviour of services, users, and the resulting interactions.

9.2.1 Exploiting structure

Our major criticism of conventional matchmaking is that it takes account only of the single service requested and the advertisements of available services. We have introduced matchmakers which take account of the service records of providers, in the context of multi-party interactions. Our model is essentially an application of information retrieval techniques as used on the early web. A key advance in web search engines was the move from treating web pages as flat sequences of text, as in traditional information retrieval, and exploring the extra structure presented through HTML tags and links (Pitkow 1997; Kleinberg 1999). Google's success is famously based on just such an exploitation of link structure through their 'PageRank' algorithm (Page et al. 1998).

We can reasonably expect similar properties to emerge in service interactions, too. What other forms of information in the LCC models and interactions could we exploit, and what other information could be added to those interactions to further improve service selection?

The kind of information that could be used includes:

- Services directly exchanging data with one another have greater dependence on one another than with another service in the same interaction which is not directly involved with that data exchange. Tracking the data flows within the protocol would enable these to be teased out.

- Variants of LCC have explored deontic constraints (Osman and Robertson 2007), dynamic transformation of the interaction model (McGinnis 2006), and ontology mapping (Besana and Robertson 2007). Each of these could provide hooks for a matchmaker which recorded them and their consequences.

Some of the structure of an interaction may be known only to the client requesting the matchmaking operation. A client may want to use that knowledge to influence the matchmaking, in a way that is too idiosyncratic or private to embed directly in a matchmaker's behaviour. Because our technique is based on structurally simple queries to the database, a client could construct its own queries for matchmaker, without requiring specialised support from it.

9.2.2 Mining matchmaking databases

The matchmakers presented in this thesis construct databases of service invocation information. We made use of these to automatically select services as required by interacting agents. If this kind of matchmaker became popular, it would become a powerful resource for its community of users and service providers. What more might be done with the kind of information it collects? As we sketched in section 7.3, we can use interaction databases to help construct the workflow itself. It is conceivable that interaction models could carry instructions to the matchmaker to store particular variables' values as clues to the interaction's behaviour. These instructions could be determined automatically, but at least initially they could be more easily added by those constructing the workflow, who would have some insight into important variables, and could add annotations to record and make use of them.

The matchmaking here has been fully automatic: an agent leaves service selection entirely in the hands of the matchmaker. There are several types of 'matchmaker' which can be differentiated by the information available to the matchmaker and client (Decker, Sycara, and Williamson 1997). Where a client has knowledge that would influence its choice of service, but that it does not wish to disclose, it can ask the matchmaker for a list of appropriate services, and make the final selection itself. Such a client can use matchmakers as we use web search engines, issuing queries and receiving lists of appropriate services, and details about the prior invocations.

Offline analysis, including clustering, could reveal explanations for the preferential performance of certain groups, in a way that could be used more generally than the very specific associations we currently work with. When the client wants to make a decision about which of set of recommended services to use, such a matchmaker can provide additional information about the services: the context in which others have used the service. In (Belhajjame et al. 2008), data flows within workflows are analysed to determine upper and lower bounds for variable types. This static analysis could benefit from the run-time information made available from our matchmaking databases.

Service providers would surely be interested to discover the patterns in which their services were being used. Moreover, they could use the interaction databases to debug their services' interactions with others. The interactions would also be of interest to the emerging field of web science (Hendler et al. 2008), which aims to understand the web and the emergent properties create through these kind of interactions.

9.2.3 Deployment

A large question remains over the likely scalability and mechanism of deployment of general-purpose matchmakers, and there seems to be little research in that direction. In this work, we too ducked the issue and considered only the single matchmaker. There are two principle issues: the technical problem of building a system capable of fielding many concurrent queries, and the organisational one of finding a sustainable way to pay

for the physical infrastructure required, and ordering the social implications.

Matchmakers must be provided by someone: Who will do that? The contemporary analogues for this kind of service include mail servers, DNS, and web search engines. The Domain Name Service (DNS) is a distributed mechanism, which is provided at the edge by users' Internet service providers (ISP). DNS provision is an inseparable part of the ISP's task, so they can be expected to provide it, and in turn can expect to charge their customers for it.

Search engines, on the other hand, pay for their existence through advertising. One early model was for search engines to alter their page rankings by placing paying customers' pages higher in the query result's order than their page intrinsically deserved. As pointed out in (Page et al. 1998), this undermines user trust, and such search engines have fallen into disuse. Today's approach places appropriate and obvious advertisements against 'honest' search results. Central to this business model is that the search engine does not alter its search results, merely adds paid-for adverts in a sidebar, or otherwise separately from the text. Thus, users still place trust in the rankings. It is hard to see how this might translate to matchmaking: agents would be impervious to eye-catching adverts. Providers might return to altering the ranking itself, lowering to perhaps intolerable levels users' trust in their results.

Domain specific matchmakers may be curated by and offered as a service to the community: in areas like bioinformatics this kind of behaviour is already providing hubs for services, documentation and the like. We might see matchmakers operated close to the user, by their employer, say, in the same manner as mail servers or intranet search engines today. Such a system would reduce the potential for sharing recommendation information, but also make it harder for service providers to disseminate their capability advertisements.

Another route would be hierarchical and peer-to-peer systems, which have been explored for information retrieval (Gravano, García-Molina, and Tomasic 1999; Siebes and Kotoulas 2007), and achieved significant real-world success in file sharing

applications. One approach taken by the *GlOSS* (Glossary of Search Servers) family of systems (Gravano, García-Molina, and Tomasic 1999) is to create a second-level information retrieval engine, which indexes the areas of expertise of the IR systems that directly index documents. In the matchmaking field, that would mean each direct matchmaking database might cover a smaller set of services (perhaps several thousand) in a particular domain, and the meta-level system would pass on queries having determined the most appropriate matchmaker to deal with it.

9.2.4 Getting real

It is hard to see how semantic web services will progress until a more hands-on approach is taken to service descriptions in realistic settings. The ubiquitous use of the travel agent scenario, for example, in semantic web services and agent papers (Berners-Lee, Hendler, and Lassila 2001; McIlraith, Son, and Zeng 2001; Fensel et al. 2006; Galizia and Gugliotta 2008; Kazhamiakin et al. 2008) is tired. The field must place more emphasis on the engineering and empirical components, and there are some encouraging signs of progress in this direction, with several 'challenge' events emerging.

The Semantic Web Services Challenge[1], and its attached workshops, is an important step in this direction (Petrie et al. 2009). Although the focus is still on the engineering of small-scale solutions, the problems are posed by the challenge organisers rather than by the contestants, so we get a better idea of how well the solutions handle problems they were not explicitly designed to solve. The semantic service selection (S3) challenge[2] is an event to test matchmakers. However, selection is done based on the use of the OWL-S and SA-WSDL test collections[3] and subjective criteria of relevance. As this thesis argued, the very model of complete and correct descriptions is questionable. Probably the most important current activity is OPOSSum (Online Portal for Semantic Services)[4]. This is

[1] http://sws-challenge.org/wiki/index.php/Main_Page
[2] http://www-ags.dfki.uni-sb.de/~klusch/s3/
[3] http://projects.semwebcentral.org/projects/owls-tc/
[4] http://fusion.cs.uni-jena.de/opossum/

an online repository for collecting and searching semantic web service descriptions from multiple formalisms, and aims to be a clearing house for such descriptions, and their comparative evaluation.

These efforts point in the right direction: we must grapple with real problems at a large-scale. Many years ago, a similar plea was made for knowledge representation in (Lenat and Guha 1990):

> The majority of work in knowledge representation has been concerned with the technicalities of relating predicate calculus to other formalisms, and with the details of various schemes for default reasoning. There has been almost an aversion to addressing the problems that arise in actually representing large bodies of knowledge with content. The typical AI researcher seems to consider that task to be 'just applications work'. But there are deep, important issues that must be addressed if we are to ever have a large intelligent knowledge-based program... In short, we must bite the bullet.

The semantic services community must bite its own bullet of building sizable, functioning collections of semantic web services, and evolve them in realistic settings with real users. There are large costs involved in this, including the monetary cost of the required engineering work, and there may be little in the way of academic prestige to be obtained in the effort. Worse yet, we may discover that real users do not want some of the alleged benefits of matchmaking at all (Lord et al. 2004).

Bibliography

Advanced Knowledge Technologies (2007). *Advanced Knowledge Technologies: Selected Papers 2007*. ISBN: 085432 873 4. See p. 12.

Akkirau, Rama, Joel Farrell, John Miller, Meenkshi Nagarajan, Marc-Thomas Schmidt, Amit Sheth, and Kunal Verma (2005). *Web Services Semantics—WSDL-S*. W3C Technial Note. Apr. 2005. See p. 68.

Altintas, I., C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock (2004). *Kepler: an extensible system for design and execution of scientific workflows*. In: *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, pp. 423–424. See p. 52.

Andrews, Tony, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana (2003). *Business Process Execution Language for Web Services Version 1.1*. URL: `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf`. See pp. 6, 51, 105, 106.

Arnold, Ken, ed. (2000). *The Jini Specifications*. second. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201726173. See p. 2.

Austin, John L. (1976). *How to Do Things With Words*. 2nd Revised. Oxford Paperbacks. See pp. 37, 99.

Baader, Franz, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, eds. (2003). *The description logic handbook: theory,*

implementation, and applications. Cambridge University Press. ISBN: 0-521-78176-0. See pp. 26, 56.

Baeza-Yates, Ricardo and Berthier Ribeiro-Neto (1999). *Modern Information Retrieval*. Essex, England: Addison Wesley/ACM Press. ISBN: 0-201-39829-X. See p. 161.

Baldi, Pierre, Paolo Frasconi, and Pahdraic Smyth (2003). *Modelling the Internet and the Web: Probabilistic Methods and Algorithms*. Wiley. See p. 170.

Bartholomew, Doug (2007). *PLM: Boeing's Dream, Airbus' Nightmare*. In: *Baseline* (Feb. 2007). Online at `http://www.baselinemag.com/c/a/Projects-Processes/PLM-Boeings-Dream-Airbus-Nightmare/`. See p. 90.

Barto, Andrew G. and Richard S. Sutton (1998). *Reinforcement Learning: An Introduction*. MIT Press. See pp. 165, 177.

Battle, Steve, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin, Sheila McIlraith, Deborah McGuinness, Jianwen Su, and Said Tabet (2005). *Semantic Web Services Framework (SWSF) Overview*. Tech. rep. World Wide Web Consortium (W3C). URL: `http://www.w3.org/Submission/SWSF/`. See pp. 8, 58.

Bechhofer, Sean, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein (2004). *OWL Web Ontology Language Reference*. Ed. by Mike Dean and Guus Schreiber. W3C Recommendation 10 February 2004. URL: `http://www.w3.org/TR/owl-ref/`. See p. 56.

Beged-Dov, Gabe, Dan Brickley, Rael Dornfest, Ian Davis, Leigh Dodds, Jonathan Eisenzopf, David Galbraith, R.V. Guha, Ken MacLeod, Eric Miller, Aaron Swartz, and Eric van der Vlist (2000). *RDF Site Summary (RSS) 1.0*. URL: `http://purl.org/rss/1.0/spec`. See p. 7.

Belhajjame, Khalid, Suzanne M. Embury, Norman W. Paton, Robert Stevens, and Carole A. Goble (2008). *Automatic Annotation of Web Services Based on Workflow Definitions*. In: *ACM Transactions on the Web* 2.2 (Apr. 2008). See p. 190.

Bellman, Richard (1961). *Adaptive Control Processes: A Guided Tour*. Princeton University Press. See p. 182.

Benjamins, V. Richard, Stefan Decker, Dieter Fensel, Enrico Motta, Guus Schreiber, Rudi Studer, Bob Wielinga, and Enric Plaza (1998). *IBROW3—An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web*. In: *Workshop on Applications of Ontologies and Problem Solving Methods of the 13th European Conference on Artificial Intelligence (ECAI98)*. Brighton, United Kingdom. See p. 31.

Berners-Lee, T., R. Fielding, and L. Masinter (2005). *Uniform Resource Identifier (URI): Generic Syntax*. Tech. rep. URL: `http://www.ietf.org/rfc/rfc3986.txt`. See p. 44.

Berners-Lee, T., J. Hendler, and O. Lassila (2001). *The Semantic Web*. In: *Scientific American* (May 2001), pp. 34–43. URL: `http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21`. See pp. 7, 57, 78, 192.

Berners-Lee, Tim (1992). *Where should one store links?* Online at `http://www.w3.org/History/19921103-hypertext/hypertext/WWW/StoringLinks.html`. URL: `http://www.w3.org/History/19921103-hypertext/hypertext/WWW/StoringLinks.html`. See p. 7.

— (1998). *What the Semantic Web can represent*. Sept. 1998. URL: `http://www.w3.org/DesignIssues/RDFnot.html`. See p. 79.

Bernstein, Philip A. (1996). *Middleware: A Model for Distributed System Services*. In: *Commun. ACM* 39.2, pp. 86–98. See p. 4.

Bernstein, Steve Battle (Hewlett Packard) Abraham, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin, Sheila McIlraith, Deborah McGuinness, Jianwen Su, and Said Tabet (2005). *Semantic Web Services*

Language (SWSL). Tech. rep. World Wide Web Consortium (W3C). URL: http://www.w3.org/Submission/SWSF-SWSL/. See p. 8.

Besana, P. and D. Robertson (2007). *How Service Choreography Statistics Reduce the Ontology Mapping Problem*. In: *Proceedings of the International Semantic Web Conference 2007*. Busan, Korea: Springer. See pp. 11, 106, 189.

Bizer, Christian, Tom Heath, and Tim Berners-Lee (2008). *Linked Data: Principles and State of the Art*. In: *17th World Wide Web Conference (WWW2008)*. Beijing, China. See p. 8.

Bjørner, Dines and Cliff B. Jones (1978). *The Vienna Development Method: The Meta-Language*. Vol. 61. Lecture Notes in Computer Science. Springer. ISBN: 3-540-08766-4. See p. 81.

Booth, David and Canyang Kevin Liu (2007). *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. Tech. rep. World Wide Web Consortium (W3C). URL: http://www.w3.org/TR/wsdl20-primer. See pp. 47, 50.

Booth, David, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard (2004). *Web Services Architecture*. W3C Working Group Note. World Wide Web Consortium (W3C). URL: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. See p. 2.

Borst, W. N. (1997). "Construction of Engineering Ontologies". PhD thesis. Enschede: University of Twente. See p. 78.

Brachman, Ronald J., Deborah L. McGuiness, Peter F. Patel-Schneider, and Lori A. Resnick (1990). *Living with CLASSIC: when and how to use a KL-ONE-like language*. In: *Principles of semantic networks*. Ed. by John Sowa. San Mateo, US: Morgan Kaufmann. URL: citeseer.ist.psu.edu/brachman91living.html. See p. 41.

Bray, Tim, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Tech. rep. World

Wide Web Consortium (W3C). URL: http://www.w3.org/TR/2008/REC-xml-20081126/. See pp. 3, 44.

Brickley, Dan and Libby Miller (2007). *FOAF Vocabulary Specification 0.91*. Online at http://xmlns.com/foaf/spec/. URL: http://xmlns.com/foaf/spec/. See p. 7.

Brin, Sergey and Lawrence Page (1998). *The anatomy of a large-scale hypertextual Web search engine*. In: *Computer Networks and ISDN Systems* 30.1–7, pp. 107–117. URL: citeseer.ist.psu.edu/brin98anatomy.html. See pp. 9, 162, 174.

Brock, Bishop, Matt Kaufmann, and J Moore (1996). *ACL2 Theorems about Commercial Microprocessors*. In: *Proceedings of Formal Methods in Computer-Aided Design (FMCAD1996)*. Ed. by M. Srivas and A. Camilleri. Springer-Verlag, pp. 275–293. See p. 81.

Bruijn, Jos de, Dieter Fensel, Uwe Keller, Michael Kifer, Holger Lausen, Reto Krummenacher, Axel Polleres, and Livia Predoiu (2005). *Web Service Modeling Language (WSML) W3C Member Submission*. June 2005. URL: http://www.w3.org/Submission/WSML/. See p. 8.

Bundy, Alan (1985). *Incidence calculus: A mechanism for probabilistic reasoning*. In: *Journal of Automated Reasoning* 1.3, pp. 263–284. See pp. 11, 106, 185.

— (1992). *Incidence calculus*. In: *Encyclopedia of Artificial Intelligence*. Wiley, pp. 663–668. See pp. 106, 185.

Burstein, Mark, Christoph Bussler, Michal Zaremba, Tim Finin, Michael N. Huhns, Massimo Paolucci, Amit P. Sheth, and Stuart Williams (2005). *A Semantic Web Services Architecture*. In: *IEEE Internet Computing* 9.5 (September/October 2005), pp. 72–81. See pp. 8, 21.

Cardelli, Luca and Andrew D. Gordon (1998). *Mobile Ambients*. In: *Proceedings of the first international conference on the Foundations of Software Science and*

Computational Structures. Ed. by Maurice Nivat. Vol. 1378. Lecture Notes in Computer Science. Springer, pp. 140–155. See p. 106.

Carriero, Nicholas and David Gelernter (1989). *Linda in context*. In: *Communications of the ACM* 32.4, pp. 444–458. ISSN: 0001-0782. DOI: http://doi.acm.org/10.1145/63334.63337. See p. 35.

Cheyer, Adam and David Martin (2001). *The Open Agent Architecture*. In: *Autonomous Agents and Multi-Agent Systems* 4 (Mar. 2001), pp. 143–148. See p. 94.

Christensen, Erik, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana (2001). *Web Services Description Language (WSDL) 1.1*. URL: http://www.w3.org/TR/wsdl. See pp. 44, 47.

Churches, David, Gabor Gombas, Andrew Harrison, Jason Maassen, Craig Robinson, Matthew Shields, Ian Taylor, and Ian Wang (2006). *Programming scientific and distributed workflow with Triana services: Research Articles*. In: *Concurrency and Computation: Practice & Experience* 18.10, pp. 1021–1037. ISSN: 1532-0626. DOI: http://dx.doi.org/10.1002/cpe.v18:10. See p. 52.

Clancey, William J. (1985). *Heuristic Classification*. Tech. rep. Department of Computer Science, Stanford University. See p. 30.

Clarke, Edmund M. and Jeannette M. Wing (1996). *Formal methods: state of the art and future directions*. In: *ACM Computing Surveys* 28.4, pp. 626–643. ISSN: 0360-0300. DOI: http://doi.acm.org/10.1145/242223.242257. See p. 81.

Clarke Jr, Edmund M., Orna Grumberg, and Doron A. Peled (1999). *Model Checking*. MIT Press. ISBN: 0-262-03270-8. See p. 105.

Clocksin, William F. and Christopher S. Mellish (2003). *Programming in Prolog: Using the ISO Standard*. 5th. Springer-Verlag. ISBN: 978-3540006787. See p. 30.

Cocchiarella, N. B. (1991). *Formal Ontology*. In: *Handbook of Metaphysics and Ontology*. Ed. by H. Burkhardt and B. Smith. Munich: Philosophia Verlag, pp. 640–647. See p. 78.

Cohn, Avra (1987). *A proof of correctness of the Viper microprocessor: the first level*. Technical report 104. Computing Laboratory, University of Cambridge. See p. 82.

Connolly, Dan, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein (2001). *DAML+OIL Reference Description*. Tech. rep. Available online at `http://www.w3.org/TR/daml+oil-reference`. World Wide Web Consortium (W3C). See p. 56.

Cullyer, W. J. (1985). *Viper Microprocessor: Formal Specification*. RSRE Report. Royal SignalsRadar Establishment. See p. 82.

— (1989). *Implementing high integrity systems: the VIPER microprocessor*. In: *IEEE Aerospace and Electronic Systems Magazine* 4 (6 1989), pp. 5–13. ISSN: 0885-8985. URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=31813`. See p. 82.

Decker, K., K. Sycara, and M. Williamson (1997). *Middle-Agents for the Internet*. In: *Proceedings of the 15th International Joint Conference on Artificial Intelligence*. Nagoya, Japan. URL: `citeseer.ist.psu.edu/decker97middleagents.html`. See pp. 5, 18–20, 172, 190.

Dickinson, Ian (2004). *Implementation experience with the DIG 1.1 specification*. Tech. rep. Hewlett Packard Digital Media Systems Laboratory. See p. 92.

Dillon, Tharam, Elizabeth Chang, Robert Meersman, and Katia Sycara, eds. (2009). *Advances in Web Semantics I: Ontologies, Web Services and Applied Semantic Web*. Springer. See p. 12.

Esteva, M., J. Rodriguez, J. Arcos, C. Sierra, and P. Garcia (2000). *Formalising Agent Mediated Electronic Institutions*. URL: `citeseer.ist.psu.edu/esteva00formalising.html`. See p. 99.

Esteva, Marc, David de la Cruz, and Carles Sierra (2002). *ISLANDER: an electronic institutions editor*. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*. Bologna, Italy: ACM Press,

pp. 1045–1052. ISBN: 1-58113-480-0. DOI: http://doi.acm.org/10.1145/545056.545069. See p. 99.

Esteva, Marc, Julian A. Padget, and Carles Sierra (2001). *Formalizing a Language for Institutions and Norms*. In: *ATAL*. Ed. by John-Jules Ch. Meyer and Milind Tambe. Vol. 2333. Lecture Notes in Computer Science. Springer, pp. 348–366. ISBN: 3-540-43858-0. See p. 99.

Fallside, David C. and Priscilla Walmsley (2004). *XML Schema Part 0: Primer Second Edition*. Tech. rep. World Wide Web Consortium (W3C). URL: http://www.w3.org/TR/xmlschema-0/. See p. 85.

Farrell, Joel and Holger Lausen (2007). *Semantic Annotations for WSDL and XML Schema*. W3C Recommendation. World Wide Web Consortium (W3C). URL: http://www.w3.org/TR/sawsdl/. See pp. 8, 68, 87, 187.

Feigenbaum, Edward and Pamela McCorduck (1983). *The fifth generation: artificial intelligence and Japan's computer challenge to the world*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-11519-0. See p. 1.

Fellbaum, Christiane, ed. (1998). *WordNet: An Electronic Lexical Database*. MIT Press. ISBN: 026206197X. See p. 93.

Fensel and Bussler (2002). *The Web Services Modelling Framework*. See p. 64.

Fensel, D., I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein (2000). *OIL in a nutshell*. In: *12th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2000)*. Lecture Notes in Computer Science. Springer-Verlag. See p. 56.

Fensel, Dieter (2001). *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer. URL: citeseer.ist.psu.edu/413498.html. See p. 79.

Fensel, Dieter, V. Richard Benjamins, Enrico Motta, and Bob J. Wielinga (1999). *UPML: A Framework for Knowledge System Reuse*. In: *Proceedings of the 16th*

International Joint Conference on AI (IJCAI-99). Morgan Kaufmann. San Francisco, CA, USA, pp. 16–23. ISBN: 1-55860-613-0. See pp. 64, 66.

Fensel, Dieter, Holger Lausen, Axel Polleres, Jos de Bruijn, Michael Stollberg, Dumitru Roman, and John Domingue (2006). *Enabling Semantic Web Services*. Springer. See pp. 31, 64, 192.

Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee (1999). *Hypertext Transfer Protocol — HTTP/1.1*. Tech. rep. Internet Engineering Task Force. See pp. 3, 44.

Fielding, Roy Thomas (2000). "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis. University of California, Irvine. See p. 50.

Fortunato, Santo, Alessandro Flammini, Filippo Menczer, and Alessandro Vespignani (2005). *The egalitarian effect of search engines*. Nov. 2005. eprint: cs.CY/0511005. URL: http://arxiv.org/abs/cs.CY/0511005. See p. 165.

Foster, Ian, Nicholas R. Jennings, and Carl Kesselman (2004). *Brain Meets Brawn*. In: *Proceedings of the third international joint conference on Autonomous Agents and Multi-Agent Systems*. New York, New York, USA: ACM Press. See p. 6.

Foster, Ian, Carl Kesselman, and Steven Tuecke (2001). *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. In: *International Journal of Supercomputer Applications* 15, p. 2001. See p. 51.

Foster, Ian, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke (2002). *Grid Services for Distributed System Integration*. In: *Computer* 35.6 (June 2002), pp. 37–46. See p. 52.

Foundation for Intelligent Physical Agents (2002). *FIPA Contract Net Interaction Protocol Specification*. Dec. 2002. See p. 35.

Frey, J., T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke (2001). *Condor-G: a computation management agent for multi-institutional grids*. In: *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*. San Francisco, CA, USA, pp. 55–63. ISBN: 0-7695-1296-8. See p. 52.

Galizia, Stefania and Alessio Gugliotta (2008). *A Framework for Selecting Trusted Semantic Web Services*. In: *Proceedings of the first Future Internet Symposium*. Sematic Technology Institute. Vienna, Austria. See p. 192.

Galstyan, Aram, Karl Czajkowski, and Kristina Lerman (2004). *Resource Allocation in the Grid Using Reinforcement Learning*. In: *In 3 rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03*. IEEE CS Press, pp. 1314–1315. See p. 52.

Ganek, A G and T A Corbi (2003). *The dawning of the autonomic computing era*. In: *IBM Systems Journal* 42.1 (Jan. 2003), pp. 5–18. See p. 33.

Gardiner, Tom, Ian Horrocks, and Dmitry Tsarkov (2006). *Automated Benchmarking of Description Logic Reasoners*. In: *Proc. of the 2006 Description Logic Workshop (DL 2006)*. Vol. 189. CEUR (`http://ceur-ws.org/`). URL: `download/2006/GaHT06a.pdf`. See pp. 91, 92.

Gaston, Matthew E. and Marie desJardins (2005). *Agent-organized networks for dynamic team formation*. In: *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. The Netherlands: ACM Press, pp. 230–237. ISBN: 1-59593-093-0. DOI: `http://doi.acm.org/10.1145/1082473.1082508`. See p. 32.

Genesereth, Michael R. and Steven P. Ketchpel (1994). *Software agents*. In: *Commun. ACM* 37.7, 48–ff. ISSN: 0001-0782. DOI: `http://doi.acm.org/10.1145/176789.176794`. See p. 100.

Gerber, Aurona, Alta van der Merwe, and Andries Barnard (2008). *A Functional Semantic Web Architecture*. In: *Proceedings of the 5th European Semantic web Conference*. Vol. 5021. Lecture Notes in Computer Science. Tenerife: Springer, pp. 273–287. See p. 56.

Giampapa, Joseph A., Massimo Paolucci, and Katia Sycara (2000). *Agent interoperation across multiagent system boundaries*. In: *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*. Barcelona, Spain: ACM Press,

pp. 179–186. ISBN: 1-58113-230-1. DOI: http://doi.acm.org/10.1145/336595.337348. See p. 94.

Gil, Yolanda (2005). *Knowledge Mobility: Semantics for the Web as a White Knight for Knowledge-Based Systems*. In: *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. Ed. by Dieter Fensel, James A. Hendler, Henry Lieberman, and Wolfgang Wahlster. MIT Press. See pp. 79, 95.

Ginsberg, Matthew L. (1991). *Knowledge interchange format: the KIF of death*. In: *AI Magazine* 12, pp. 57–63. See p. 91.

Glinton, Robin, Paul Scerri, and Katia Sycara (2008). *Agent-based sensor coalition formation*. In: *11th International Conference on Information Fusion*. Cologne, Germany, pp. 1–7. ISBN: 978-3-8007-3092-6. See p. 32.

Goble, C, C Wroe, and R Stevens (2003). *The myGrid project: services, architecture and demonstrator*. In: *Proceedings of the UK e-Science All Hands Meeting*. See pp. 53, 55.

Goble, Carole and David De Roure (2002). *The Grid: an application of the semantic web*. In: *SIGMOD Rec.* 31.4, pp. 65–70. ISSN: 0163-5808. DOI: http://doi.acm.org/10.1145/637411.637422. URL: http://portal.acm.org/citation.cfm?id=637411.637422#. See p. 51.

Gomadam, Karthik and Amit Sheth (2008). *SA-REST: Using Semantics to Empower RESTful Services and Smashups with Better Interoperability and Mediation*. In: *Semantic Technology Conference 2008*. San Jose, CA, USA. URL: http://knoesis.wright.edu/library/download/gomadam_Sheth_SA-REST_semTech2008.ppt. See p. 50.

Goodenough, John B. and Susan L. Gerhart (1975). *Toward a theory of test data selection*. In: *ACM SIGPLAN Notices* 10.6, pp. 493–510. ISSN: 0362-1340. DOI: http://doi.acm.org/10.1145/390016.808473. See p. 81.

Gottschalk, K., S. Graham, H. Kreger, and J. Snell (2002). *Introduction to Web services architecture*. In: *IBM Systems Journal* 41.2, pp. 170–177. URL: http:

//researchweb.watson.ibm.com/journal/sj/412/gottschalk.html. See p. 3.

Gravano, Luis, Héctor García-Molina, and Anthony Tomasic (1999). *GlOSS: text-source discovery over the Internet*. In: *ACM Transactions on Database Systems* 24.2, pp. 229–264. ISSN: 0362-5915. DOI: http://doi.acm.org/10.1145/320248.320252. See pp. 191, 192.

Gruber, Thomas R. (1993). *A translation approach to portable ontology specifications*. In: *Knowl. Acquis.* 5.2, pp. 199–220. ISSN: 1042-8143. DOI: http://dx.doi.org/10.1006/knac.1993.1008. See p. 78.

Guo, L, Dave Robertson, and Yun-Heh Chen-Burger (2005). *Enacting the Distributed Business Workflows Using BPEL4WS on the Multi-Agent Platform*. In: *Proceedings of the MATES 2005 conference*. Vol. 3550. LNAI. See pp. 11, 106.

Gärdenfors, Peter (2000). *Conceptual Spaces: The Geometry of Thought*. MIT Press. See p. 80.

— (2004). *How to make the Semantic Web more semantic*. In: *Formal Ontology in Information Systems*. Ed. by A.C. Varzi and L. Vieu. IOS Press, pp. 19–36. See p. 80.

Hadley, Marc J. (2006). *Web Application Description Language (WADL)*. Nov. 2006. See p. 50.

Halevy, Alon, Peter Norvig, and Fernando Pereira (2009). *The Unreasonable Effectiveness of Data*. In: *IEEE Intelligent Systems* (March/April 2009), pp. 8–12. See p. 187.

Halpern, Joseph Y. and Yoram Moses (1984). *Knowledge and common knowledge in a distributed environment*. In: *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing*. Vancouver, British Columbia, Canada: ACM, pp. 50–61. ISBN: 0-89791-143-1. DOI: http://doi.acm.org/10.1145/800222.806735. See p. 101.

Hendler, James and Jennifer Golbeck (2008). *Metcalfe's law, Web 2.0, and the Semantic Web*. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 6 (1 2008), pp. 14–20. DOI: 10.1016/j.websem.2007.11.008. See p. 89.

Hendler, James, Nigel Shadbolt, Wendy Hall, Tim Berners-Lee, and Daniel Weitzner (2008). *Web science: an interdisciplinary approach to understanding the web*. In: *Communications of the ACM* 51.7, pp. 60–69. ISSN: 0001-0782. DOI: http://doi.acm.org/10.1145/1364782.1364798. See p. 190.

Horn, Paul (2001). *Autonomic Computing: IBM's Perspective on the State of Information Technology*. Online at http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf. Oct. 2001. URL: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf. See p. 32.

Horrocks, Ian (2002). *DAML+OIL: A Description Logic for the Semantic Web*. In: *IEEE Bulletin of the Technical Committee on Data Engineering* 25.1 (Mar. 2002), pp. 4–9. See p. 56.

Horrocks, Ian and Peter Patel-Schneider (1999). *Optimizing description logic subsumption*. In: *Journal of Logic and Computation* 9 (3 1999), pp. 267–293. See p. 26.

Horwood, Martin (2005). *CAD data quality*. In: *Engineering Designer* (May/June 2005). Online at http://www.infosys.com/industries/aerospace-defense/white-papers/cad-quality.pdf. See p. 90.

Hull, Duncan, Evgeny Zolin, Andrey Bovykin, Ian Horrocks, Ulrike Sattler, and Robert Stevens (2006). *Deciding Semantic Matching of Stateless Services*. In: *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06) and Eighteenth Innovative Applications of Artificial Intelligence (IAAI-06) Conference*. Boston, MA, USA, pp. 1319–1324. See pp. 64, 172.

Joseph, Sindhu, Adrián Perreau de Pinninck Bas, David Robertson, Carles Sierra, and Chris Walton (2007). *Interaction Model Language Definition*. In: *IJCAI Workshop*

AOMS Agent Organizations Models and Simulations. Ed. by Virginia Dignum, Frank Dignum, Eric Matson, and Bruce Edmonds, pp. 49–61. See p. 106.

Kalfoglou, Yannis and Marco Schorlemmer (2003). *Ontology Mapping: The State of the Art*. In: *The Knowledge Engineering Review* 18, p. 2003. See pp. 9, 93.

Kaufer, Frank and Matthias Klusch (2006). *A Logic Programming Based Hybrid Service Matchmaker*. In: *Proceedings of the 4th IEEE European Conference on Web Services (ECOWS 2006)*. Zurich, Switzerland: IEEE CS Press. See pp. 70, 80, 172.

Kavantzas, Nickolaos, David Burdett, and Greg Ritzinger (2004). *Web Services Choreography Description Language Version 1.0*. W3C Working Draft 27 April 2004. URL: http://www.w3.org/TR/ws-cdl-10/. See p. 89.

Kazhamiakin, Raman, Piergiorgio Bertoli, Massimo Paoluccci, Marco Pistore, and Matthias Wagner (2008). *Having Services "YourWay!": Towards User-Centric Composition of Mobile Services*. In: *Proceedings of the first Future Internet Symposium*. Sematic Technology Institute. Vienna, Austria. See p. 192.

Keller, Uwe, Rubén Lara, Axel Polleres, Ioan Toma, Michel Kifer, and Dieter Fensel (2004). *D5.1 WSMO Web Service Discovery*. Tech. rep. Web Services Modeling Ontology Working Group. URL: http://www.wsmo.org/2004/d5/d5.1/. See p. 66.

Kifer, Michael and Georg Lausen (1989). *F-logic: a higher-order language for reasoning about objects, inheritance, and scheme*. In: *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data*. Portland, Oregon, United States: ACM Press, pp. 134–146. ISBN: 0-89791-317-5. DOI: http://doi.acm.org/10.1145/67544.66939. See p. 64.

Kifer, Michael, Georg Lausen, and James Wu (1995). *Logical foundations of object-oriented and frame-based languages*. In: *J. ACM* 42.4, pp. 741–843. ISSN: 0004-5411. DOI: http://doi.acm.org/10.1145/210332.210335. See p. 64.

Kleinberg, J. (1999). *Authoritative sources in a hyperlinked environment*. In: *Journal of the ACM* 46.5, pp. 604–632. See p. 188.

Klusch, M., B. Fries, and M. Khalid (2005). *OWLS-MX: Hybrid Semantic Web Service Retrieval*. In: *Proceedings of the 1st International AAAI Fall Symposium on Agents and the Semantic Web*. See pp. 26, 70–72, 80, 164, 172.

Klusch, Matthias and Katia Sycara (2001). *Brokering and matchmaking for coordination of agent societies: a survey*. In: *Coordination of Internet agents: models, technologies, and applications*. Springer-Verlag, 197–224. ISBN: 3-540-41613-7. URL: http://www.dfki.de/~klusch/papers/chapter8.pdf. See p. 18.

Konstan, Joseph A, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl (1997). *GroupLens: applying collaborative filtering to Usenet news*. In: *Communications of the ACM* 40.3, pp. 77–87. See p. 31.

Kopecky, Jacek, Karthik Gomadam, and Tomas Vitvar (2008). *hRESTS: an HTML Microformat for Describing RESTful Web Services*. In: *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*. Sydney, Australia. See pp. 50, 68.

Kopecký, Jacek, Tomas Vitvar, Dieter Fensel, and Karthik Gomadam (2008). *D12 MicroWSMO: Semantic Annotations for RESTful Services*. Tech. rep. Conceptual Models of Services working group (CMS), STI International. URL: http://cms-wg.sti2.org/TR/d12/. See pp. 68, 187.

Kornfeld, W A (1979). *ETHER: A parallel problem solving system*. In: *Proceedings of the 6th International Joint Conference on Artificial Intelligence (IJCAI '79)*. Tokyo, Japan, pp. 490–492. See p. 34.

Kornfeld, William A. (1979). *Using Parallel Processing for Problem Solving*. Tech. rep. Artificial Intelligence Lab, Massachusetts Institute of Technology. URL: http://dspace.mit.edu/handle/1721.1/5719. See p. 34.

Kornfeld, William A. (1981). *The use of parallelism to implement a heuristic search*. In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*. Vancouver, Canada, pp. 575–580. See p. 34.

Korzybski, Alfred (1931). *A Non-Aristotelian System and its Necessity for Rigour in Mathematics and Physics*. In: *Meeting of the American Association for the Advancement of Science*. New Orleans, Louisiana, USA. See p. 87.

Kuokka, D. and L. Harada (1995). *Matchmaking for Information Agents*. In: *Proceedings of the 14th IJCAI*, pp. 672–678. See pp. 5, 39.

Kuokka, Daniel and Larry Harada (1996). *Integrating information via matchmaking*. In: *Journal of Intelligent Information Systems* 6.2-3, pp. 261–279. ISSN: 0925-9902. DOI: http://dx.doi.org/10.1007/BF00122130. See p. 39.

Labrou, Yannis and Tim Finin (1997). *A Proposal for a new KQML Specication*. Tech. rep. Baltimore, Maryland: Computer ScienceElectrical Engineering Department, University of Maryland Baltimore County. See pp. 37, 38.

Lambert, D. and D. Robertson (2006). *Selecting Web Services Statistically*. In: *Tenth International Workshop on Cooperative Information Agents*. Edinburgh, Scotland. See p. 12.

Lambert, David and David Robertson (2005). *Matchmaking and Brokering Multi-Party Interactions Using Historical Performance Data*. In: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05)*. Utrecht, Netherlands, pp. 611–617. See p. 12.

Lambert, David J. (2005). *Accounting for Valency in Service Composition*. In: *First Advanced Knowledge Technologies Doctoral Symposium*. Milton Keynes, England. See p. 12.

Lamport, Leslie (1987). *Distribution*. Online at http://research.microsoft.com/users/lamport/pubs/distributed-system.txt. URL: http://research.microsoft.com/users/lamport/pubs/distributed-system.txt. See p. 87.

Larsen, Peter Gorm, John Fitzgerald, and Tom Brookes (1996). *Applying Formal Specification in Industry*. In: *IEEE Software* 13 (3 1996), pp. 48–56. See p. 81.

Lathem, Jon, Karthik Gomadam, and Amit P. Sheth (2007). *SA-REST and (S)mashups : Adding Semantics to RESTful Services*. In: *Proceedings of the International Conference on Semantic Computing (ICSC 2007)*. Los Alamitos, CA, USA: IEEE Computer Society, pp. 469–476. DOI: http://doi.ieeecomputersociety.org/10.1109/ICSC.2007.94. See p. 90.

Lausen, Holger, Axel Polleres, and Dumitru Roman (2005). *Web Service Modeling Ontology (WSMO)*. Tech. rep. World Wide Web Consortium (W3C). URL: http://www.w3.org/Submission/WSMO/. See p. 8.

Lenat, Douglas B. and R. V. Guha (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley. See p. 193.

Li, L. and I. Horrocks (2003). *A software framework for matchmaking based on semantic web technology*. URL: citeseer.ist.psu.edu/li03software.html. See pp. 5, 172.

Li, Lei and Ian Horrocks (2004). *A Software Framework For Matchmaking Based on Semantic Web Technology*. In: *International Journal of Electronic Commerce* 8.4 (Summer 2004), pp. 39–60. See pp. vii, 62, 63, 80, 114.

Licklider, J.C.R. (1963). *Memorandum For Members and Affiliates of the Intergalactic Computer Network*. Apr. 1963. URL: http://www.kurzweilai.net/articles/art0366.html?printable=1. See p. 13.

Licklider, J.C.R. and Robert W. Taylor (1968). *The Computer as a Communication Device*. This paper was reprinted in: In Memoriam: J. C. R. Licklider 1915-1990 and Research Report 61 Digital Equipment Corporation Systems Research Center August 1990. Apr. 1968. URL: http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-061.html. See p. 13.

Liu, Weiru (2001). *Propositional, Probabilistic and Evidential Reasoning: Integrating Numerical and Symbolic Approaches*. Physica-Verlag. See pp. 106, 108.

Liu, Weiru, Alan Bundy, and Dave Robertson (1993). *On the Relations between Incidence Calculus and ATMS*. In: *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*. Springer-Verlag, pp. 249–256. See p. 111.

Lord, P, P Alper, C Wroe, and C Goble (2005). *Feta: A Light-Weight Architecture for User Oriented Semantic Service Discovery*. In: *Proceedings of the 2nd European Semantic web Conference*. Vol. 3532. Lecture Notes in Computer Science. Springer, pp. 17–31. ISBN: 978-3-540-26124-7. DOI: 10.1007/b136731. See pp. 54, 83, 85, 88.

Lord, Phillip, Sean Bechhofer, Mark D. Wilkinson, Gary Schiltz, Damian Gessler, Duncan Hull, Carole Goble, and Lincoln Stein (2004). *Applying Semantic Web Services to Bioinformatics: Experiences Gained, Lessons Learned*. In: *The Semantic Web — ISWC 2004*. Ed. by Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen. Vol. 3298. Lecture Notes in Computer Science. Hiroshima, Japan: Springer, pp. 350–364. See pp. 54, 77, 89, 193.

Luan, Xiaocheng (2004). "Adaptive Middle Agent for Service Matching in the Semantic Web: A Quantitative Approach". PhD thesis. University of Maryland, Baltimore County. URL: http://ebiquity.umbc.edu/paper/html/id/206/. See pp. 70, 76.

— (2004). *Quantitative Agent Service Matching*. In: *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*. See p. 70.

Manning, Christopher D. and Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. MIT Press. See p. 183.

Marcus, Sandra and John McDermott (1989). *SALT: a knowledge acquisition language for propose-and-revise systems*. In: *Artificial Intelligence* 39.1, pp. 1–37. ISSN: 0004-3702. DOI: http://dx.doi.org/10.1016/0004-3702(89)90002-7. See p. 30.

Martin, D., A. Cheyer, and D. Moran (1999). *The Open Agent Architecture: A Framework for Building Distributed Software Systems*. In: *Applied Artificial Intelligence* 13 ((1-2) 1999), pp. 92–128. See pp. 5, 25, 42.

Martin, David, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara (2004). *OWL-S: Semantic Markup for Web Services*. W3C Member Submission 22 November 2004. URL: `http://www.w3.org/Submission/OWL-S`. See pp. 8, 58, 60, 89.

McGinnis, Jarred P. (2006). "On the Mutability of Protocols". PhD thesis. School of Informatics, University of Edinburgh. See pp. 106, 189.

McIlraith, S., T. Son, and H. Zeng (2001). *Semantic Web services*. In: *IEEE Intelligent Systems* 16 (2 2001), pp. 46–53. ISSN: 1541-1672. URL: `citeseer.ist.psu.edu/mcilraith01semantic.html`. See pp. 8, 192.

Meyer, Bertrand (1985). *On Formalism in Specifications*. In: *IEEE Software* 2 (1 1985), pp. 6–26. See p. 81.

Microsoft Corporation (1996). *DCOM Technical Overview*. Online at `http://msdn2.microsoft.com/en-us/library/ms809340.aspx`. Nov. 1996. URL: `http://msdn2.microsoft.com/en-us/library/ms809340.aspx`. See p. 2.

Miles, S., J. Papay, Dialani, V., Luck, M., Decker, K., Payne, T., Moreau, and L. (2003). *Personalised Grid Service Discovery*. In: *Proceedings of the nineteenth annual UK Performance Engineering Workshop (UKPEW'03)*. See pp. 54, 173.

Milner, Robin (1999). *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press. See p. 100.

Mitchell, Tom M. (1997). *Machine Learning*. New York: McGraw-Hill. See p. 146.

Moto-oka, Tohru and Masaru Kitsuregawa (1985). *The Fifth Generation Computer: The Japanese Challenge*. John Wiley and Sons. See p. 1.

Motta, Enrico (1999). *Reusable Components for Knowledge Modelling*. Vol. 53. Frontiers in Artificial Intelligence and Applications. IOS Press. ISBN: 978-1-58603-003-2. See p. 81.

Nardi, Daniele and Ronald J. Brachman (2003). *An Introduction to Description Logics*. In: ed. by Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. Cambridge University Press. ISBN: 0-521-78176-0. See p. 5.

Norton, Barry and Carlos Pedrinaci (2006). *3-Level Service Composition and Cashew: A Model for Orchestration and Choreography in Semantic Web Services*. In: *2nd International Workshop on Agents, Web Services and Ontologies Merging (AWeSOMe'06)*. See p. 66.

OASIS UDDI Specification TC (2005). *Universal Description, Discovery and Integration v3.0.2 (UDDI)*. Feb. 2005. URL: `http://www.oasis-open.org/specs/index.php#uddiv3.0.2`. See p. 48.

Object Management Group (2004). *Common Object Request Broker Architecture: Core Specification*. Online at `http://www.omg.org/cgi-bin/apps/doc?formal/04-03-12.pdf`. Mar. 2004. URL: `http://www.omg.org/cgi-bin/apps/doc?formal/04-03-12.pdf`. See p. 2.

Oinn, T. M., M. Addis, J. Ferris, D. Marvin, R. M. Greenwood, T. Carver, M. R. Pocock, A. Wipat, and P. Li (2004). *Taverna: a tool for the composition and enactment of bioinformatics workflows*. In: *Bioinformatics* 20.17 (Nov. 2004), pp. 3045–3054. See pp. 4, 6.

Osman, N., D. Robertson, and C. Walton (2005). *Run-time model checking of interaction and deontic models for multi-agent systems*. In: *Proceedings of the European Multi-Agent Systems Workshop, 2005*. See pp. 11, 105, 119.

Osman, Nardine and David Robertson (2007). *Dynamic Verification of Trust in Distributed Open Systems*. In: *IJCAI 2007, Proceedings of the 20th International*

Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007. Ed. by Manuela M. Veloso. See pp. 105, 189.

Page, Lawrence, Sergey Brin, Rajeev Motwani, and Terry Winograd (1998). *The PageRank Citation Ranking: Bringing Order to the Web*. Tech. rep. Stanford Digital Library Technologies Project. URL: citeseer.ist.psu.edu/page98pagerank.html. See pp. 188, 191.

Palankar, Mayur R., Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel (2008). *Amazon S3 for science grids: a viable solution?* In: *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*. Boston, MA, USA: ACM, pp. 55–64. ISBN: 978-1-60558-154-5. DOI: http://doi.acm.org/10.1145/1383519.1383526. See p. 3.

Pan, Zhengxiang (2005). *Benchmarking DL Reasoners Using Realistic Ontologies*. In: *OWL: Directions and Experiences*. Galway, Ireland. See pp. 91, 92.

Paolucci, M., T. Kawamura, T. Payne, and K. Sycara (2002*a*). *Importing the semantic web in UDDI*. URL: citeseer.ist.psu.edu/article/paolucci02importing.html. See pp. 48, 62.

Paolucci, M., N. Srinivasan, K. Sycara, and T. Nishimura (2003*a*). *Towards a Semantic Choreography of Web Services: from WSDL to DAML-S*. In: *Proceedings of the International Conference on Web Services (ICWS 2003)*. Las Vegas, Nevada, USA. See p. 85.

Paolucci, Massimo, Takahiro Kawamura, Terry R. Payne, and Katia Sycara (2002*b*). *Semantic Matching of Web Services Capabilities*. In: *The Semantic Web — ISWC 2002: Proceedings*. See p. 5.

Paolucci, Massimo, Anupriya Ankolekar, Naveen Srinivasan, and Katia Sycara (2003*b*). *The DAML-S Virtual Machine*. In: *Proceedings of the Second International Semantic Web Conference*. Sanibel Island, Florida, USA. See p. 85.

Papazoglou, Mike P. (2003). *Service-Oriented Computing: Concepts, Characteristics and Directions*. In: *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE 03)*. See p. 2.

Payne, Terry R. (2008). *Web Services from an Agent Perspective*. In: *IEEE Intelligent Systems* 23.2 (March/April 2008), pp. 12–14. See p. 6.

Petrie, C., T. Margaria, H Lausen, and M Zaremba, eds. (2009). *Semantic Web Services Challenge: Results from the First Year*. Vol. 8. Semantic Web and Beyond. Springer Verlag. ISBN: 978-0-387-72495-9. See p. 192.

Pitkow, James Edward (1997). "Characterizing World Wide Web Ecologies". PhD thesis. Georgia Institute of Technology, Georgia, USA. See p. 188.

Proceedings of the 5th European Semantic web Conference (2008). Vol. 5021. Lecture Notes in Computer Science. Tenerife: Springer.

Proceedings of the first Future Internet Symposium (2008). Sematic Technology Institute. Vienna, Austria.

Proceedings of the first international joint conference on Autonomous agents and multiagent systems (AAMAS 2002) (2002). Bologna, Italy: ACM Press. ISBN: 1-58113-480-0.

Proceedings of the International Semantic Web Conference 2007 (2007). Busan, Korea: Springer.

Quan, Xueping, Chris Walton, Dietlind L. Gerloff, Joanna L. Sharman, and David Robertson (2007). *Peer-to-Peer Experimentation in Protein Structure Prediction: An Architecture, Experiment and Initial Results*. In: *GCCB*. Ed. by Werner Dubitzky, Assaf Schuster, Peter M. A. Sloot, Michael Schroeder, and Mathilde Romberg. Vol. 4360. Lecture Notes in Computer Science. Springer, pp. 75–98. ISBN: 3-540-69841-8. See p. 106.

Raman, R., M. Livny, and M. Solomon (1998). *Matchmaking: distributed resource management for high throughputcomputing*. In: *Proceedings of the Seventh Inter-*

national Symposium on High Performance Distributed Computing. Chicago, IL, USA, pp. 140–146. ISBN: 0-8186-8579-4. See p. 52.

Raman, R., M. Livny, and M. Solomon (2000). *Resource management through multilateral matchmaking*. In: *Proceedings of the ninth International Symposium on High-Performance Distributed Computing*. Pittsburgh, PA, USA, pp. 290–291. ISBN: 0-7695-0783-2. See p. 52.

Resnick, Paul and Hal R. Varian (1997). *Recommender systems*. In: *Commun. ACM* 40.3, pp. 56–58. ISSN: 0001-0782. DOI: http://doi.acm.org/10.1145/245108.245121. See pp. 10, 31.

Richardson, Leonard and Sam Ruby (2007). *RESTful Web Services*. O'Reilly Media, Inc. See pp. 47, 86.

Robertson, D., C. Walton, A. Barker, A. Besana, Y. Chen-Burger, F. Hassan, D, Lambert, G. Li, J. McGinnis, N. Osman, A. Bundy, F. McNeil, F. van Harmelen, C. Sierra, and F. Giunchiglia (2009). *Models of Interaction as a Grounding for Peer to Peer Knowledge Sharing*. In: ed. by Tharam Dillon, Elizabeth Chang, Robert Meersman, and Katia Sycara. Springer. See p. 12.

Robertson, David (2004). *A lightweight method for coordination of agent oriented web services*. In: *Proceedings of the 2004 AAAI Spring Symposium on Semantic Web Services*. California, USA. See pp. 11, 99, 185.

Robertson, David, Fausto Giunchiglia, Frank van Harmelen, Maurizio Marchese, Marta Sabou, W. Marco Schorlemmer, Nigel Shadbolt, Ronny Siebes, Carles Sierra, Chris Walton, Srinandan Dasmahapatra, David Dupplaw, Paul H. Lewis, Mikalai Yatskevich, Spyros Kotoulas, Adrian Perreau de Pinninck, and Antonis Loizou (2007). *Open Knowledge*. In: *LADS*. Ed. by Mehdi Dastani, Amal El Fallah-Seghrouchni, João Leite, and Paolo Torroni. Vol. 5118. Lecture Notes in Computer Science. Springer, pp. 1–18. ISBN: 978-3-540-85057-1. See p. 106.

Robertson, David Stuart, Flávio S. Corrêa da Silva, Wamberto Weber Vasconcelos, and Ana Cristina Vieira de Melo (2000). *A Lightweight Capability Communication*

Mechanism. In: *Intelligent Problem Solving. Methodologies and Approaches: 13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2000, New Orleans, Louisiana, USA, June 2000. Proceedings*. Vol. 1821. Lecture Notes in Computer Science. Springer-Verlag, pp. 660–670. See p. 43.

Roure, David De and Carole Goble (2007). *myExperiment—A Web 2.0 Virtual Research Environment*. In: *International Workshop on Virtual Research Environments and Collaborative Work Environments*. Edinburgh, United Kingdom. URL: http://eprints.ecs.soton.ac.uk/13961/. See p. 55.

SAP News Desk (2005). *Microsoft, IBM, SAP To Discontinue UDDI Web Services Registry Effort*. Online at http://soa.sys-con.com/node/164624. Dec. 2005. URL: http://soa.sys-con.com/node/164624. See p. 48.

Schafer, Ben J, Joseph Konstan, and John Riedl (1999). *Recommender systems in e-Commerce*. In: *Proceedings of the ACM Conference on Electronic Commerce*. See pp. 10, 32.

Schreiber, A T and W P Birmingham (1996). *Editorial: The Sisyphus-VT initiative*. In: *International Journal of Human-Computer Studies* 44 (3/4 1996), pp. 373–402. See p. 81.

Shirky, Clay (2003). *The Semantic Web, Syllogism, and Worldview*. First published as an email on the 'Networks, Economics, and Culture' mailing list. Available online at http://www.shirky.com/writings/semantic_syllogism.html. Nov. 2003. URL: http://www.shirky.com/writings/semantic_syllogism.html. See pp. 8, 79.

— (2005). *Ontology is Overrated: Categories, Links, and Tags*. Available online at http://www.shirky.com/writings/ontology_overrated.html. URL: http://www.shirky.com/writings/ontology_overrated.html. See p. 79.

Siebes, Ronny and Spyros Kotoulas (2007). *pRoute: Peer selection using shared term similarity matrices*. In: *Journal Web Intelligence and Agent Systems* 5 (1 2007), pp. 89–107. ISSN: 1570-1263. See p. 191.

Singh, Munindar P. and Michael N. Huhns (2005). *Service-Oriented Computing: Semantics, Processes, Agents*. Chichester, England: John Wiley & Sons. See p. 1.

Singh, N. (1993). *A Common Lisp API and Facilitator for ABSI*. Tech. rep. Logic Group, Computer Science Department, Stanford University. See pp. 5, 39.

Sirin, E., B. Parsia, and J. Hendler (2004). *Filtering and selecting semantic Web services with interactive composition techniques*. In: *IEEE Intelligent Systems* 19 (4 2004), pp. 42–49. ISSN: 1541-1672. See p. 63.

Smith, Michael K., Chris Welty, and Deborah L. McGuinness (2004). *OWL Web Ontology Language Guide*. W3C Recommendation 10 February 2004. URL: `http://www.w3.org/TR/owl-guide/`. See pp. 26, 56.

Smith, R. G. (1980). *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*. In: *IEEE Transactions on Computers* 29.12 (Dec. 1980), pp. 1104–1113. See pp. 4, 16, 17.

Smith, R .G. and R. Davis (1978). *Applications of the Contract Net Framework: Distributed Sensing*. In: *Proceedings ARPA Distributed Sensor Net Symposium*. Pittsburgh, USA, pp. 12–20. See p. 35.

Spivey, J. Michael (1992). *The Z Notation: a reference manual*. Second. Prentice Hall. See pp. 25, 81.

Srinivasan, R. (1995). *Remote Procedure Call Protocol Specification Version 2*. Tech. rep. Internet Engineering Task Force. URL: `http://www.ietf.org/rfc/rfc1831.txt`. See p. 14.

Stollberg, Michael, Martin Hepp, and Jörg Hoffmann (2007). *A Caching Mechanism for Semantic Web Service Discovery*. In: *Proceedings of the International Semantic Web Conference 2007*. Busan, Korea: Springer. See pp. 67, 172, 173.

Stollberg, Michael, Matthew Moran, Liliana Cabral, Barry Norton, and John Domingue (2006). *Experiences from Semantic Web Service Tutorials*. In: *Semantic Web Education and Training Workshop (SWET'06), First Asian Semantic Web Conference (ASWC)*. Beijing, China. See p. 83.

Studer, R., V. Benjamins, and D. Fensel (1998). *Knowledge engineering: Principles and methods*. In: *IEEE Transactions on Data and Knowledge Engineering* 25.1-2, pp. 161–197. URL: citeseer.comp.nus.edu.sg/225099.html. See pp. 30, 78.

Subrahmanian, V. S., Piero Bonatti, Jurgen Dix, Thomas Eiter, and Fatma Ozcan (2000). *Heterogeneous Agent Systems–Heterogeneous Agent Systems*. MIT Press. ISBN: 0262194368. See pp. 26, 40, 80, 129.

Sycara, K., M. Klusch, and S. Widoff (1999). *Dynamic Service Matchmaking Among Agents in Open Information Environments*. In: *SIGMOD Record* 28 (1 1999), pp. 47–53. See p. 5.

Sycara, K., J. A. Giampapa, B.K. Langley, and M. Paolucci (2003*a*). *The RETSINA MAS, a Case Study*. In: *Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications*. Ed. by Alessandro Garcia, Carlos Lucena, Franco Zambonelli, Andrea Omici, and Jaelson Castro. Vol. 2603. Lecture Notes in Computer Science. Berlin Heidelberg: Springer-Verlag, pp. 232–250. See pp. 40, 94.

Sycara, Katia, Seth Widoff, Matthias Klusch, and Jianguo Lu (2002). *LARKS: Dynamic Matchmaking Among Heterogeneous Software Agnets in Cyberspace*. In: *Autonomous Agents and Multi-Agent Systems* 5, pp. 173–203. See pp. 22, 40.

Sycara, Katia P., Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan (2003*b*). *Automated discovery, interaction and composition of Semantic Web services.* In: *J. Web Sem.* 1.1, pp. 27–46. See pp. 26, 61, 62, 71, 114.

Teije, A. ten, F. van Harmelen, and B. Wielinga (2004). *Configuration of Web Services as Parametric Design*. In: *Proceedings of the 14th International Conference, (EKAW-*

2004). Ed. by E. Motta, N. Shadbolt, A. Stutt, and N. Gibbins. Lecture Notes in Artificial Intelligence 3257. ISBN 3-540-23340-7. Whittleburry Hall, UK: Springer Verlag, pp. 321–336. See p. 31.

Teije, Annette ten, Frank van Harmelen, A. Th. Schreiber, and Bob J. Wielinga (1998). *Construction of problem-solving methods as parametric design*. In: *International Journal of Human Computer Studies* 49.4, pp. 363–389. See p. 31.

The DAML Services Coalition (2003). *DAML-S: Semantic Markup for Web Services*. Online at `http://www.daml.org/services/daml-s/0.9/daml-s.html`. URL: `http://www.daml.org/services/daml-s/0.9/daml-s.html`. See p. 62.

Tobies, Stephan (2001). "Complexity Results and Practical Algorithms for Logics in Knowledge Representation". PhD thesis. RWTH Aachen. See p. 26.

Toma, Ioan, Brahmananda Sapkota, James Scicluna, Juan Miguel Gomez, Dumitru Roman, and Dieter Fensel (2005). *A P2P Discovery mechanism for Web Service Execution Environment*. In: *Proceedings of the 2nd International WSMO Implementation Workshop*. Innsbruck, Austria. See pp. 35, 66.

Trastour, David, Claudio Bartolini, and Javier Gonzalez-Castillo (2001). *A Semantic Web Approach to Service Description for Matchmaking of Services*. In: *Proceedings of SWWS'01, The first Semantic Web Working Symposium*. Ed. by Isabel F. Cruz, Stefan Decker, Jérôme Euzenat, and Deborah L. McGuinness. Stanford University, California, USA, pp. 447–461. See p. 63.

Viceconti, Marco, Fulvia Taddei, Serge Van Sint Jan, Alberto Leardini, Gordon Clapworthy, Stefania Galizia, and Paolo Quadrani (2007). *Towards the multiscale modelling of musculoskeletal system*. In: see p. 85.

Vinoski, Steve (2002*a*). *Putting the "Web" into Web Services: Web Services Interaction Models, Part 1*. In: *IEEE Internet Computing* 6.3 (May-June 2002), pp. 89–91. See p. 84.

Vinoski, Steve (2002*b*). *Putting the "Web" into Web Services: Web Services Interaction Models, Part 2*. In: *IEEE Internet Computing* 6.4 (July-August 2002), pp. 90–92. See p. 84.

Vitvar, Thomas, Jacek Kopecky, Jana Viskova, and Dieter Fensel (2008). *WSMO-Lite Annotations for Web Services*. In: *Proceedings of the 5th European Semantic web Conference*. Vol. 5021. Lecture Notes in Computer Science. Tenerife: Springer. See p. 8.

Vitvar, Tomas, Jacek Kopecký, and Dieter Fensel (2008). *D11 WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web*. Tech. rep. Conceptual Models of Services working group (CMS), STI International. URL: http://cms-wg.sti2.org/TR/d11/. See pp. 69, 187.

Walton, Christopher (2004). *Model Checking Multi-Agent Web Services*. In: *Proceedings of the 2004 AAAI Spring Symposium on Semantic Web Services*. See pp. 105, 119.

Walton, D. N. and E. C. W. Krabbe (1995). *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. Albany, NY, USA: SUNY Press. See p. 89.

Walton, N. A., A. Lawrence, and T. Linde (2003). *AstroGrid: Initial Deployment of the UK's Virtual Observatory*. In: *Astronomical Data Analysis Software and Systems (ADASS) XIII*. Strasbourg, France. See p. 3.

White, James E. (1975). *A High-Level Framework for Network-Based Resource Sharing*. Tech. rep. Internet Engineering Task Force. See p. 14.

Wilkinson, Mark D. and Matthew Links (2002). *BioMOBY: An open source biological web services proposal*. In: *Briefings in bioinformatics* 3.4, pp. 331–341. DOI: doi:10.1093/bib/3.4.331. See pp. 3, 55.

Wilson, Brian (2008). *MAMA: W3C validator research*. Published online at http://dev.opera.com/articles/view/mama-w3c-validator-research-2/. Oct. 2008. See p. 82.

Winer, Dave (1999). *XML-RPC Specification*. Online at `http://www.xmlrpc.com/spec`. June 1999. URL: `http://www.xmlrpc.com/spec`. See p. 49.

Wing, Jeannette M. (1990). *A specifier's introduction to formal methods*. In: *Computer* 23 (9 1990), pp. 8,10–22,24. ISSN: 0018-9162. See p. 81.

Witten, Ian H., Alistair Moffat, and Timothy C. Bell (1999). *Managing Gigabytes*. Academic Press. See pp. 161, 174.

Wong, H. and K. Sycara (2000). *A Taxonomy of Middle-agents for the Internet*. In: *4th International Conference on Multi-Agent Systems (ICMAS 2000)*. URL: `citeseer.ist.psu.edu/wong00taxonomy.html`. See p. 22.

Wooldridge, M. (2000). *Reasoning About Rational Agents*. MIT Press. ISBN: 0-262-23213-8. See p. 6.

Wooldridge, Michael and Nicholas R. Jennings (1995). *Intelligent Agents: Theory and Practice*. `http://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-html.h` (Hypertext version of Knowledge Engineering Review paper). URL: `citeseer.ist.psu.edu/article/wooldridge95intelligent.html`. See p. 18.

Wooldridge, Mike and P. Ciancarini (2000). *Agent-Oriented Software Engineering: The State of the Art*. In: *First Int. Workshop on Agent-Oriented Software Engineering*. Ed. by P. Ciancarini and M. Wooldridge. Vol. 1957. Springer-Verlag, Berlin, pp. 1–28. URL: `citeseer.ist.psu.edu/wooldridge00agentoriented.html`. See pp. 2, 90.

Wroe, C, CA Goble, A Goderis, and P Lord (2007). *Recycling workflows and services through discovery and reuse*. In: *Concurrency and Computation: Practice and Engineering* 19 (2 2007), pp. 181–194. See pp. 77, 187.

Wroe, Chris, Carole Goble, Mark Greenwood, Phillip Lord, Simon Miles, Juri Papay, Terry Payne, and Luc Moreau (2004). *Automating Experiments Using Semantic Data on a Bioinformatics Grid*. In: *IEEE Intelligent Systems* 19.1 (January/February 2004), pp. 48–55. See pp. 23, 54, 55.

Yost, G R and T R Rothenfluh (1996). *Configuring elevator systems*. In: *International Journal of Human-Computer Studies* 44 (3/4 1996), pp. 521–568. See p. 81.

Zaremba, Maciej, Tomas Vitvar, Matthew Moran, and Thomas Hasselwanter (2006). *WSMX Discovery for SWS Challenge*. In: *SWS Challenge Workshop, at ISWC 2006*. Athens, Georgia, USA. See p. 66.

Zaremba, Michal, Matthew Moran, Thomas Haselwanter, Ho-Kyung Lee, and Sung-Kook Han (2004). *WSMX Architecture*. Tech. rep. Web Services Modeling Ontology Working Group. URL: http://www.wsmo.org/TR/d13/d13.4/. See p. 65.

Zhang, Zili and Chengqi Zhang (2002). *An improvement to matchmaking algorithms for middle agents*. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*. Bologna, Italy: ACM Press, pp. 1340–1347. ISBN: 1-58113-480-0. DOI: http://doi.acm.org/10.1145/545056.545129. See pp. viii, 69, 76, 88, 113, 128–130, 133, 136.