Adaptive Middle Agent for Service Matching in the Semantic Web: A Quantitative Approach

Xiaocheng Luan, Doctor of Philosophy, 2004

Dissertation Directed by:

Yun Peng, Associate Professor
Timothy Finin, Professor
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County

Abstract

With the advent of the Web services and the need for a Semantic Web, the agent technology is (in my view) finally becoming a viable solution to many real world problems. Effective service matching is key to the success of agent systems, but existing service matching methods mostly consider service descriptions as the only factor and many important issues have been largely overlooked. In the real world applications, agents with identical service descriptions may differ dramatically in performance levels; an agent may have strong and weak areas in its service offerings; and the distribution of services may provide helpful hints on which matches are more likely to be better than the others. Moreover, an agent's capability may change over time. These are very important issues and if left unaddressed, may become real problems in the real world applications.

In our approach presented in this dissertation, the middle agent establishes and refines an agent's capability model based on the domain ontology and through the interactions with the agents. In this framework, an agent's performance history is considered as an integral part of the agent's capability model and the agent's strong and weak areas can also be revealed. Moreover, the dynamically captured and updated service distribution in the service domain is considered as an important factor in service matching. Service matching here is carried out in two steps. In the first step, candidates are selected through the semantic service description matching. In the second step, the performance rating of each candidate with respect to the specific request is estimated based on the agent's capability model, and the candidates with the highest estimated performance ratings will be selected.

The major advantages over existing methods are the establishment and refinement of an agent's capability model and the use of such a model in service matching. A prototype system and an evaluation framework have been implemented to evaluate the ideas discussed in this work. The statistics collected from the experiment shows a significant improvement over typical service matching methods in terms of the accuracy in selecting the best service provider(s) for each request.

Table of Contents

1 Introduction 1

1.1 The Motivations 2

1.2 Thesis Statement 3

1.3 Dissertation Outline 4

2 Background and Related Work 6

2.1 Agent Communication Languages 8

2.2 The Ontology 10

2.3 The Connection Problem and Solutions 12

2.4 Web Services and the Semantic Web 13

2.4.1 Web services 14

2.4.2 The Semantic Web 15

2.5 Related Work 17

2.5.1 Service matching infrastructures 17

2.5.2 Other related work 19

2.6 Summary 20

3 An Abstract Multi-Agent System Model 22

3.1 The Abstract Multi-Agent System Model 22

3.2 Agent Service Description 24

3.3 A Simple Domain Ontology 25

3.4 Summary 27

4 Service Description Matching 29

4.1 An Interpretation of Service Descriptions and Service Matching 29

4.2 Parameter Matching 33

4.3 Parameter Pairing 34

4.3.1 Parameter pairing using maximum bipartite matching 35

4.3.2 Maximum weighted, maximum cardinality parameter pairing 36

4.4 Put it Together 37

5 The Agent's Capability Model 39

5.1 The Service Distribution Model 40

5.1.1 The service distribution model 41

5.1.2 The parameter type distribution model 42

5.2 The Performance Model 44

5.3 OWL-S Extension 48

5.4 A Note on Interaction Protocols 49

5.5 Summary 51

6 Put it Together - Quantitative Service Matching 52
6.1 The Basics 53
6.2 The Desired Properties for a Quantitative Measure 54
6.3 A Quantitative Measure for Service Matching 56
6.4 Proof of Conformance 57
6.5 The Complexity 60
6.6 Summary 61
7 The Experimental Implementation and Results 62
7.1 The Design and Implementation 62
7.1.1 Daml4jess 62
7.1.2 The Java modules 63
7.1.3 Put it all together 64
7.2 An Evaluation Framework 64
7.2.1 The distribution model 66
7.2.2 Generating advertised services 68
7.2.3 Generating requests and feedback 69
7.2.4 The matchmaker categories 70
7.3 The Result Statistics 71
8 Conclusion and Discussions 78
9 References and Bibliography 80

List of Figures

Figure 2-1. W3C Web services technologies 14
Figure 2-2. Top level of the service ontology 16
Figure 3-1. Operations in the matchmaker framework 23
Figure 3-2. The airport location class hierarchy 26
Figure 3-3. Class hierarchies for flight ticket and payment 26
Figure 4-1. Levels of matches 32
Figure 4-2. An example bipartite graph 35
Figure 5-1. Computer manufacture service distribution model 41
Figure 5-2 The distribution model 44
Figure 5-3. A simple example of rating entries 45
Figure 6-1. Service spaces of the request and the rating entries 54
Figure 7-1 Matchmaker agent design overview 63
Figure 7-2 The evaluation framework 65
Figure 7-3. Percentage of requests fulfilled 72
Figure 7-4. Mean of satisfaction ratings, subsumes off 73
Figure 7-5. Standard deviation of satisfaction ratings, subsumes off 74
Figure 7-6. The effect of feedback percentage 75
Figure 7-7. Mean of satisfaction ratings, subsumes on 76

List of Tables

Table 7-1. Paired t-test on the performance, type-2 vs. type-3 matchmaker 74
Table 7-2. Performance improvement, type-3 over type-2..... 75

1 Introduction

This research is concerned with how a middle agent (e.g., a matchmaker) can more accurately identify the best service provider agent(s) for a given request, by taking advantage of the domain ontologies and by continuously learning and refining the capability model of the service provider agents through the agent interactions.

An agent can be a physical agent (e.g., robot) or a software agent, but in this work, our focus is on the software agents. Loosely speaking, a software agent (agent, hereafter) is a piece of program that usually exhibits the following properties: autonomy, cooperation, and adaptation. Autonomy refers to the property that an agent can operate without the direct intervention of human or others; cooperation refers to the property of cooperating with the other agents to accomplish the tasks; and adaptation refers to the property that an agent can adapt to the environment and learn from past experience.

While an agent may not sound that different from an ordinary program, it takes an agent system to unleash the full potential of the agent technology. The idea is that in an agent system, the agents may cooperate and coordinate with each other as needed to jointly accomplish tasks that a single program may not be able to accomplish. Therefore, some researchers see agent technology as a new software engineering paradigm.

Agent researchers have been trying to find killer applications for the agent technology for over a decade, but in my opinion it has not been very successful. However, I think a killer application for the agent technology is emerging. The advent of the Web services, especially the need for a semantic web and "true" interoperability among Web services, provide challenges as well as opportunities for agent technology. Semantics and interoperability have been the focuses and strengths of the agent technology – from high level agent communication languages like KQML and FIPA ACL, to semantic languages like KIF and FIPA SL, and to ontology technologies like Ontolingua - virtually everything is crafted to support semantics and interoperability. Therefore, it does not take much imagination to see the fit. In my opinion, the Semantic Web activity is a major effort in that direction. Many believe that Web services and the Semantic Web will be the next big thing and this provides a huge opportunity for the agent technology to finally find its place in the industry. With OWL (Web Ontology Language) becoming a W3C recommendation, I think the trend has already started.

The power of the agent technology comes from the coordination and cooperation among the agents. But when an agent has a task (that is better performed by others), how does it find the agents that can perform the task for it? This is the well known "connection problem" [15] and has been a subject of extensive research in the past two decades. The solutions fall into one of the two broad categories: the broadcasting based solutions and the middle agent (or facilitator) based solutions. We will briefly illustrate the two approaches using examples here, and more details will be discussed in Section 2.3.

The Contract Net protocol [60] is an example of the broadcasting based solutions. In the Contract Net protocol, the manager (the agent that has a task to be executed) broadcasts a RFQ (Request For Proposal); those who are interested may respond with a

proposal. The manager will then evaluate the proposals and (possibly after negotiations) the contract will be awarded to the agent that is considered the most appropriate for the task. The agent that receives such an award is called a contractor.

A matchmaker is a type of middle agent that performs service matching. In a matchmaker framework, the service provider agents advertise their capabilities (or descriptions of services that they can provide) to the matchmaker; when a service consumer has a task to be performed it requests the matchmaker to "recommend" service provider(s) that can perform the task; the matchmaker, upon receipt of such a recommendation request, will try to find agents whose capability descriptions match that of the request, and then the top matches, if any, will be recommended to the requester.

The middle agent based approach is very flexible and is suitable for small and large agent systems alike. Most of today's middle agents perform service matching only based on service descriptions. In other words, service matching is characterized as service description matching. While this characterization is correct by itself, it directly lends itself to some issues when putting the agents into the real world applications, for example:

- Will the agents that advertise identical services perform equally well on the advertised services?
- Will an agent perform equally well across its service offerings?
- Are the advertised service descriptions accurate with respect to their true capabilities, even if the agents are "honest"?
- Could other factors play a role in selecting service providers, for example, the distribution of services, that is, frequency of request occurrence across sub service domains?

These are the issues that motivated this research.

1.1 The Motivations

If left unaddressed, the questions raised above can become real issues when agents are put into the real world applications.

In the real world, performance matters. For example, suppose your SUV engine fails and you look through the yellow pages to find an auto repair shop. The good news is that you find 25 local auto repair shops that say they can perform any auto-repair work. But your worry may not stop there, because you know too well that some auto repair shops would charge you a hefty price but do a poor job, especially when it comes to the engine repair – even though they all say that they can do it. Therefore, service descriptions alone may not solve your problems. Moreover, a shop may have strong and weak areas. A shop that satisfactorily fixed your car battery problem last time may or may not perform well on the engine problem.

Similarly, in the agent world, there bound to be good performers and bad performers and it would not be unusual for an agent to have strong and weak areas, and these factors can be crucial in selecting service provider agents for a given request. Moreover, an

agent's capability to perform specific tasks may change/improve over time, for example, through learning or just a hardware upgrade, while the advertised service descriptions may remain unchanged.

Service distribution refers to the frequency of request occurrences across sub-domains of a service domain. For example, in the computer retail domain, it might be that 75% of the requests fall in the sub-domain of desktop retails and 24% fall in the sub-domain of laptop retails. It may not be evident how service distribution can be a factor in selecting service providers, an example may help. Suppose an agent asks the middle agent to recommend an online store that sells computers. The middle agent has two relevant provider listings, laptop-express.com that advertised laptop retails and desktop-buy.com that advertised desktop retails. Without considering the service distribution factor, the middle agent could not tell which one is better since both providers partially match the request. With the distribution factor considered, say if today 75% of computer purchases are desktops and 24% are laptops, the middle agent should have more confidence in desktop-buy.com because there is a 75% chance that this provider will meet the need of the requester.

In summary, an agent's performance, especially an agent's performance with respect to the given request and the distribution of services in a service domain can be important factors in selecting the service providers but have largely been overlooked. This is exactly the motivation of this research.

1.2 Thesis Statement

This research is an attempt to address the issues raised in the previous sections, by developing a general, quantitative method that a middle agent can use to establish and refine a service provider agent's capability model with which the agent's performance with respect to a specific request may be accordingly estimated. As a result, the agents that are most appropriate for the given request may be discovered and selected.

Adaptation is one of the key properties of an agent, which is about adapting to the environment and learning from the past experience. But this property is largely missing in many (if not all) of today's middle agents. As a result, an agent's performance history is not taken into account because the middle agent is not learning about the agents' past experience; service distribution is not taken into account because the middle agent is not learning about the environment, for example, the distribution of services. In this work we will attempt to address the issues through agent adaptation, that is, through the middle agent learning. Formally defined domain ontologies will play an important role and an assumption is made that all agents in the system share common domain ontologies.

There are two parts to this work. The first part is concerned with the establishment and refinement of an agent's capability model, which in turn has two major components: a service distribution model and an agent performance model. The service distribution model is global to a reference service domain and dynamically captures the distribution of services as far as the middle agent's reach. Through such a model the middle agent may, for example, know that 75% of computer purchase requests are about desktops. The agent performance model captures an agent's past performance using a set of dynamically managed rating entries. A rating entry is a construct that at the core has a

profile and a rating value, where the profile specifies the scope (typically a subset of the advertised service) and the rating value is the cumulative rating on the agent with respect to the entry's profile.

The second part deals with how an agent's capability model may be used in service matching to estimate the performance of a service provider agent with respect to a given request. We proposed a two step approach. In the first step, candidates are selected through the regular semantic service description matching. In the second step, the agent capability model is used to estimate a candidate's performance with respect to the request. A weighted average based method is proposed, where the weights are decided based on the relevancy of a rating entry to the request.

The middle agent is chosen as the target for addressing the issues because of its special role in an agent system. Some agents might be smart enough to learn about the other agent's capabilities but it will be a heavy burden for the agents and it may not be very effective since an individual agent's experience is usually limited. A middle agent, however, is in an especially "convenient" position to take this responsibility. A middle agent takes service advertisements as well as service requests and therefore knows the service demands and the service offerings. A middle agent usually has more resources (e.g., memory, computing power) than the other agents and it may stay around longer (have longer life span), too. Capturing and modeling the agents' capabilities is consistent with its ultimate goal, that is, to select the most appropriate service provider(s) for the service consumers. Moreover, since the middle agent is involved in selecting the service providers, it is legitimate and proper for it to receive feedback from the service consumer agents, or for it to check with the consumer agents, "how was the service?"

In summary, this work is an attempt to prepare the middle agents for the real world applications by addressing the issues that have been largely overlooked in agent service matching, for example, whether and how factors other than the service descriptions should play a role in service matching. A prototype system and an evaluation framework have been implemented to evaluate the ideas discussed in this work, and the result statistics shows a significant improvement over typical matching methods in the accuracy of selecting the best service provider(s) for a given request.

1.3 Dissertation Outline

This dissertation is outlined as follows. In Chapter 2, we first provide an overview of the background for this work, including the agent technology (agent communication languages, ontologies, the connection problem and solutions), Web services, and the Semantic Web technology. We then briefly review the previous researches related to this work.

Chapter 3 presents a simple abstract multi-agent system model, introduces agent service description basics as well as the domain ontology used in this work. The purpose is to lay down the ground work for the discussions in the later chapters.

Chapter 4 discusses semantic service matching, that is, to determine if two given service descriptions semantically match. Two major issues are discussed: how to

determine if two given parameters match and how to determine which parameter (in one service description) corresponds to a parameter in the other service description.

In Chapter 5, we first introduce the agent capability model, including the service distribution model and the agent performance model, and then discuss the extension to the OWL-S ontology that is needed to support this work.

Chapter 6 focuses on the issue of how to make use of the agent capability model introduced in Chapter 5. A two step approach is proposed. In the first step, semantic matching is performed to find the candidates that semantically match the request; in the second step, a weighted average based method is used to estimate a candidate's performance with respect to the given request.

In Chapter 7, we describe the prototype matchmaker that is implemented under the OWL/OWL-S framework and the evaluation framework that is used to simulate the rest of the agent system, and then present and discuss the statistics collected from the experiment.

Chapter 8 concludes the dissertation by discussing some of the related issues as well as the directions of future research.

2 Background and Related Work

After nearly two decades, there still seems to be no consensus on the definition of “agent”. Researchers define agents from different perspectives. For example, "Autonomous agents are systems capable of autonomous, purposeful action in the real world" [7]; and in [42], "Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed".

I personally like best the way that Wooldridge and Jennings characterized it in [71] with the weak notion that an agent is a computer system that possesses the properties of autonomy, social ability, reactivity, and pro-activeness. These properties are explained in [71] as follow:

- autonomy: agents operate without the direct intervention of human or others, and have some kind of control over their actions and internal state (Castelfranchi, 1995);
- social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language (Genesereth and Ketchpel, 1994);
- reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;
- pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

The stronger notion of agents is given in terms of mentalistic notions, e.g., knowledge, belief, intention, and obligation. Some researchers go even further to consider emotional states of agents. For example, in the BDI model, an agent has Beliefs, Desires, and Intentions.

Some other researchers characterize agent with the properties of autonomy, cooperation, and adaptation. The first two properties are basically covered by the list above, but the property of adaptation is new. Adaptation refers to the notion that agents adapt to their environment and learn from the past experience. While it is generally agreed that an agent can be intelligent or non-intelligent, the notion of adaptation is a bit stronger and would generally require that agents be intelligent. We stop right here and will not get into the discussion of what is “intelligence”.

Agent technology incorporates (or attempts to incorporate) ideas from a wide range of areas, including, but not limited to: Artificial Intelligence and Cognitive Science, Database and Knowledge Base Technologies, Information Retrieval, Distributed Systems, and Linguistics. An agent can be a physical agent, for example, a robot, or a piece of program, that is, a software agent. Some researchers see agent technology as a new programming paradigm, a natural generalization of client-server architecture and

emphasis on interoperability. This work focuses on software agents, though the ideas and algorithms may be applicable to the other types of agents, too.

When it comes to agent systems, the agent interaction (e.g., cooperation and/or coordination) is an area of special interest. It is not just that more agents can do more work. The interactions enable the agents to achieve what otherwise could not be achieved. As Gerhard Weib indicated in **Error! Reference source not found.**, that, intelligence and interaction are deeply and inevitably coupled to each other. On the one hand, intelligence can help to achieve flexible and effective interactions; on the other hand, through the interactions, an agent can learn what it does not know before and can achieve what it can not achieve by itself, i.e., the interactions can increase their knowledge and power.

Agents typically interact (or communicate) at a higher level of discourse (than ordinary programs or processes), that is, at knowledge level. Agent communication languages such as KQML and FIPA ACL support high level agent communications. Both KQML and FIPA ACL are based on speech act theory and have similar syntax. For example, in a KQML message, the message content is represented using a content language (such as KIF) independent of KQML; a KQML performative is used to communicate the "intention" or "attitude" the sender has about the content of the message, such as querying, stating, believing, etc; and a set of KQML parameters may be used to specify such information as the sender, the receiver, the content language, etc. See Section 2.1 for more discussions on the subject of the agent communication languages.

For the agents to understand each other, they will not only need to speak the same language(s), they will also need to have the same understanding of meanings and semantics of the terms used – otherwise, some kind of translation will need to be involved to resolve the semantic differences. For example, in one agent's vocabulary, the computer hard disk may be called "hard disk", while in another agent's vocabulary the same thing may be called "hard drive". The agents will either need to use the same term or there needs to be a way to associate the two terms. Using shared domain ontologies will solve such problems. An ontology is a specification of a conceptualization [29], it serves as a common background for the domain of interest. An ontology will typically include a vocabulary of terms, their definitions, and the inter-relationships between these terms. Therefore, using formally defined ontologies will allow disparate systems to achieve mutual understanding. This is the subject of Section 2.2.

Before the agents can talk to each other, there need to be a way for the agents to get to "know" each other, for example, an agent needs to know what the other agents can do for it, and what the communication address is. This is called the "connection problem" [15]. The solutions fall into two broad categories, the broadcast-based solutions, such as the Contract Net protocol, and the facilitator-based solutions, in which a facilitator (a middle agent) helps connecting the agents. More discussions on this may be found in Section 2.3.

When I was defending the proposal for this work six years ago in August 1998 (that was a long time ago), I was asked the question of what was my view on the future of the agent technology. I was kind of pessimistic at the time for the near term, even though agent technology was in a red hot state. My main concern was that, on the "intelligent"

side, I did not think that we can build an agent that is as smart as it needs to be in order to play in the agent world we envisioned; on the "non-intelligent" side, I did not see how much better the agent technology can do than the component technology or the other distributed technologies.

However, I am much more optimistic today than I was six years ago. The progress on the Web services and the Semantic Web shed some light on the agent technology, or more boldly, I think the Web services and the Semantic Web may finally bring the agent technology to life (reality). According to [65], Web services are the programmatic interfaces of the web applications. The World Wide Web is more and more used for application to application communication and this offers a vast play ground for the agent technology. The work on the Semantic Web takes the right step in that direction. "The Semantic Web is a vision: the idea of having data on the web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications" [65].

In the rest of this chapter, we will first look into each of the areas discussed above in more details, and then we will look at some of the works related to this dissertation.

2.1 Agent Communication Languages

I remember Dr. Timothy Finin (UMBC) made the assertion in various talks that agent-agent communication is the key to realizing the potential of the agent technology. I could not agree with him more, and there is no doubt that an appropriate agent communication language (ACL) is essential for effective communication.

Typically agents communicate at a higher level of abstraction, i.e., the contents of the communication are meaningful statements about the agents' environment or knowledge [22]. This is in contrast to the low-level communications such as TCP/IP or method invocation in CORBA. In the early years, KQML was the de facto standard agent communication language and later FIPA ACL was developed and widely adopted by the agent community. Both KQML [37] and FIPA ACL [20] are based on the speech act theory and the two share similar syntax. In brief, the speech act theory is a high-level framework developed by philosophers and linguistics to account for human communication [36]. A speech act is called a "performative" (for example, "ask", "tell", and "agree") that expresses the intention and/or attitude of the speaker/sender toward the content of the speech or message.

KQML, Knowledge Query and Manipulation Language, is the result of Knowledge Sharing Efforts, External Interfaces Group. KQML is message-based and it considers that each message not only contains the content, but also the intention the sender has for that content, such as querying, stating, believing, requiring, achieving, subscribing, and offering. KQML is indifferent to the format of the content, nor the content itself. In a KQML message, there is a "performative" that is used to specify the intention (or attitude) the sender has toward the message; a set of "parameters" (name-value pairs) that are used to specify such information as the sender, the receiver, the content language, ontology, etc; and the message content itself is contained as sub-expressions in other, so-called "content language".

Here is an example KQML message:

```
(ask
    :sender agent-A@umbc.edu
    :receiver agent-B@yahoo.com
    :language KIF
    :ontology toy-onto
    :content (home-phone-number ?x)
    :reply-with “address update, case#1234”
)
```

By using the “ask” performative, the sender says that “I am asking you”, and the thing being asked about is specified in the content, “(home-phone-number ?x)”. The parameter “:language” indicates that the content “(home-phone-number ?x)” is written in a language called KIF (the Knowledge Interchange Format); the parameter “:ontology” specifies the ontology used in the content; and the “:reply-with” parameter suggests the receiver (of this message) should include the string “address update, case#1234” in the “:in-reply-to” field of the reply message. The “:reply-with” and “:in-reply-to” parameters may help in establishing a “conversation” context. The communication between agents usually occurs in the form of “conversations”. Informally, a conversation is a set of messages sent and/or received that aim at addressing some problem. The message example below together with the example message discussed above can be considered a conversation:

```
(tell
    :sender agent-B@yahoo.com
    :receiver agent-A@umbc.edu
    :language KIF
    :ontology toy-onto
    :content (home-phone-number 410-455-5555)
    :in-reply-to “address update, case#1234”
)
```

In [37], 36 reserved performatives are listed. These performatives are grouped into 3 categories: discourse performatives such as ask-if, tell and achieve; intervention and mechanics performatives such as error, sorry and discard; facilitation and networking performatives such as register, broadcast, and broker-one. Readers are referred to [37] for more details on KQML.

The content language is independent of KQML. KIF, the Knowledge Interchange Format, was designed to fill such a role. KIF is first order predicate logic based and has declarative semantics. It was developed by the Knowledge Sharing Effort, Interlingua Group. It is a system-neutral language that can be used for the interchange of knowledge among disparate systems.

FIPA ACL is another high-level agent communication language, developed by The Foundation for Intelligent Physical Agents, a non-profit association, and is now a FIPA

standard. FIPA ACL is also based on the speech act theory and its syntax is similar to KQML. More precisely, FIPA ACL has many components: the FIPA ACL agent message structure specification, the FIPA ontology service specification, the communicative acts library specification, the content language specifications, and the interaction protocol specifications. The list may still grow. The FIPA ACL message structure component together with the FIPA communicative acts library is roughly the equivalent of KQM. As with KQML, FIPA communicative acts are independent of the content language. FIPA communicative acts library may work with different content languages. In fact, FIPA specified its own content language, SL (Semantic Language), and the work on the content language specifications for KIF, RDF (Resource Description Framework), and CCL (Constraint Choice Language) is in progress.

FIPA is doing a lot of interesting work, much more than the agent communication languages. The specifications range from abstract architecture to agent management/ agent platform, to agent communication languages and agent message transportations. Interested users are referred to [20].

This layered architecture of agent communication languages separates the communicative acts (or performatives) from the actual content of the message. It appears to be based on sound theories (e.g., the speech act theory) but I think I have some reservations to this approach. The major concern to me is that the intention (attitude) toward the message content and the message content itself together should be one, integral entity, just as in English or any other natural languages. By separating these two components in two separate, independent layers in two independent languages, there is the risk that the semantic of a message is cut into two pieces and may or may not be completely recoverable; or from a different perspective, the performative (or communicative act) may or may not always match/work perfectly with specific message contents.

I think (blindly) that the speech act theory is sound, and should be applicable to virtually any languages, including agent communication languages, but that does not necessarily mean that the performatives should be separated from the (content) language. For example, the speech act theory is applicable to English, but you do not want to separate the performatives from English and then use English as the content language.

2.2 The Ontology

Suppose agent X manages a medical information database. When agent Y asks agent X to find some recent publications on “malignant neoplasm” (cancer), agent X replies “sorry, we do not have what you requested”. What is the problem here? Agent X does not know that “malignant neoplasm” and “cancer” actually mean the same thing.

Suppose later agent Y asks agent X for some recent articles on “cold and vitamin C”. This time agent X finds a long list of articles on the effectiveness of using vitamin C for the disease “cold”. But that was not what agent Y wanted – agent Y wanted articles on how low temperature affects the properties of vitamin C. What is the problem here then? The term “cold” may mean different things and the two agents used different meanings (interpretations) of the same term.

These examples are just special cases? Then how about this one: a consumer agent C asks the yellow page agent Y for nearby stores that sell laptops. The yellow page agent does have a long list of stores that sell computers but it could not find one for agent C – it does not know that a laptop is a computer.

Ontologies are exactly what can be used to address all these problems. An ontology is a specification of a conceptualization [29] – a conceptualization of a knowledge domain. An ontology will typically include a vocabulary of terms (in the problem domain), the term definitions, and possibly the inter-relationships between these terms. An ontology is more than just a vocabulary and may represent some of the basic domain knowledge. For example, "laptop is a kind of computer", "malignant neoplasm is the same as cancer", or "a computer must have a keyboard".

Ontologies are usually defined formally using ontology languages so that the defined ontologies can be understood by the agents, in other words, machine-readable. Ontolingua is such an early example of ontology languages developed by the KSL lab of the Stanford University. Its syntax and semantics are based on KIF and it extends KIF with standard primitives for defining classes and relations, and for organizing knowledge in object-centered hierarchies with inheritance. "Ontolingua" [29] is also the name of the set of tools developed at KSL for supporting the creation, translation, and the management of ontologies. The system provides several services, including KIF parsing, analysis of KIF-based ontologies, translation (of ontologies to other languages), and the generation of WWW documents. OWL (Web Ontology Language) [64] is another ontology language and has recently become the W3C recommendation in early 2004. OWL is a description logic based ontology language and is therefore especially suitable for defining concept hierarchies and the relationship between the concepts. OWL will be discussed in more details in Section 2.4.

Now we briefly look at how ontologies may help solve the problems in the three examples above. First of all, we assume that the two agents use the same ontology. For the first example, requesting articles on "malignant neoplasm", we may add an entry to the ontology that states, "cancer is equivalent to malignant neoplasm". Agent X would then know that agent Y was asking articles on "cancer" and the related articles will be retrieved and returned. In the second example, requesting articles on "cold and vitamin C", the ontology may explicitly define, for example, that when you refer to the disease "cold", the term "common cold" should be used, and the term "cold" means the condition of low temperature. Then the two agents may have a shared understanding of the term "cold". In the third example, "looking for stores that sell laptops", we may add an entry in the ontology stating that, "a laptop is a computer". Then the yellow page agent will at least have the basis to reason and conclude that stores that sell computers should or are likely to sell laptops, too, and return the list of computer-selling stores.

What if the two agents do not use the same ontology, for example, agent X uses an ontology that defines the term "cold" as the disease "cold", and agent Y uses an ontology that defines the term "cold" as the condition of low temperature? There is no doubt that the problem here is getting harder, but with formally defined ontologies, there may be solutions. In Section 2.1, we mentioned that an agent message may (and in general, should) specify what ontology it is using. When agent X receives the request message from agent Y, it will know that they are using different ontologies, and since both their

ontologies are formally defined, there is a chance that the terms in one ontology may be translated into terms of another ontology. Agent X may do the translation by itself if it can, or it may request other ontology translation service agents (if any) to do the translation for it. The translation approach is also applicable in the case when the two ontologies are in different languages, for example, one in Ontolingua and the other in OWL.

Using formally defined ontologies does not automatically guarantee the interoperability among agents. For example, an agent may not have the sufficient capability to utilize the knowledge specified in the ontology, or in the case of two agents using different ontologies, there may not readily be a translation available. However, using formally defined ontologies provides the basis for achieving such goals.

Interoperability is not the only purpose that domain ontologies can serve. Domain ontologies provide a domain model that may be used in many ways. For example, this work uses the domain ontology as the basis for refining an agent's capability model as well as in capturing the service distribution.

In summary, ontology is a very important piece in achieving knowledge sharing and interoperability among agents. It serves as the common background for the domain of interest. Without such a shared understanding, the agents may find it difficult to communicate with each other. As illustrated in the above examples, they may use different vocabularies; they may use different definitions of the same term; they may also have different assumptions regarding what is essentially the same subject matter; and so on. Using formally defined ontologies have the additional benefits in re-usability and reliability, since a system with well-defined ontology is more open, easier to work with others.

2.3 The Connection Problem and Solutions

As some have put it, agent-agent communication is the key to realizing the potential of the agent paradigm. To achieve effective communication, however, the "connection problem" [15], that is, how an agent that has tasks to be performed finds the agents that can accomplish the tasks, needs to be addressed.

The solutions fall into two broad categories, the broadcast-based solutions and the facilitator-based solutions. The Contract Net Protocol [60] is a good example of the broadcast-based solution. The Contract Net Protocol may be roughly described as follow. Each agent in the system takes one of the two roles with respect to the execution of an individual task: the manager or a contractor. An agent that has a task to be performed announces a "call for proposals" that specifies the terms of the task, and takes the role as the manager. Agents that receive the call for proposals may, if interested, reply with their proposals and become potential contractors. The manager evaluates the proposals and after going through a selection process, a contract may be established and the winner(s), if any, will be the contractor(s) to perform the task.

In the facilitator-based approach, a facilitator agent (nowadays more frequently called a middle agent) maybe used to help establish the connections. There are different ways to "facilitate" the connection and therefore there are different types of middle agents. Three

types of middle agents are identified in [34], namely the matchmakers, the brokers, and the mediators. In a facilitated framework, there are usually three types of agents: the service consumer (or requester) agents that have tasks to be performed, the service provider agents that have the capabilities and intention to perform tasks, and the middle agents that help establish the connections between the service consumers and the service providers.

A matchmaker serves the role similar to that of the yellow page. The service providers advertise their capabilities to the matchmaker; the service consumer agents request the matchmaker to lookup for service providers that can perform tasks of the given description, and if any are found, the top matches will be sent to the service consumer agent. A broker agent works in a different way. When it receives a request from the service consumer agent, it finds a (ideally the best) service provider to execute the task(s), and then returns the results of the execution back to the service consumer agent. A mediator works in a way similar to the broker but it does more. Its services may range from finding the right service providers, to the translations between different ontologies and/or languages, to decomposing the tasks to be executed by different agents and combining the results, etc.

There is actually a third type of solutions, the blackboard-based solutions. There are different variants of blackboard-based solutions. In some cases, for example [1], the blackboard serves as a data repository in which the agents who have tasks to be performed post their tasks to a central mechanism called the "blackboard", and agents who can perform tasks would monitor the blackboard and grab tasks that it can accomplish. In some other cases, as described in [11], the control shell, that is, the control component in a blackboard based system, selects what to be executed. Therefore, a blackboard may be considered as a kind of facilitator, and in [16], a blackboard is considered a middle agent.

2.4 Web Services and the Semantic Web

Web services are the programmatic interfaces that are made available for application to application communication on the web. "The power of Web services, in addition to their great interoperability and extensibility thanks to the use of XML, is that they can then be combined in a loosely coupled way in order to achieve complex operations. Programs providing simple services can interact with each other in order to deliver sophisticated added-value services." [66].

If taken out of the context, I would have thought the above comments are about the agent technology. But is it not so? I think what labels we use is not that important, be it Web services or agent technology, it is the fundamental ideas that matters, and I think Web services technology and the agent technology have a lot in common. Taking it the other way, many of the ideas and technologies developed over the years under the umbrella of "agent technology" may be applicable, possibly with changes, to the Web services technology, for example, formally defined domain ontology, service descriptions with semantics, etc. Not surprisingly, the work on the Semantic Web (details see Section 2.4.2), I think, is such an attempt and efforts are on the way to apply the Semantic Web technologies to the Web services.

Therefore, the first reason that I am including this section in the background review is that our work should be applicable to the Semantic Web technologies (and the Web services), too. The second reason is that our discussions and the experimental implementation are based on the OWL/OWL-S framework (RDF and OWL are at the core of the Semantic Web technology). I think with the emergence of the Web services technology and the Semantic Web technology, the agent technology may finally find its place in the industry.

2.4.1 Web services

"The World Wide Web is more and more used for application to application communication. The programmatic interfaces made available are referred to as **Web services**" [66]. Web services technology has great potential. Up until now, the web has been mainly used directly by human users. With Web services, however, "programs providing simple services can interact with each other in order to deliver sophisticated added-value services." [66], that is, it will enable computer-computer interactions and automate complex tasks, such as making the arrangements for a trip.

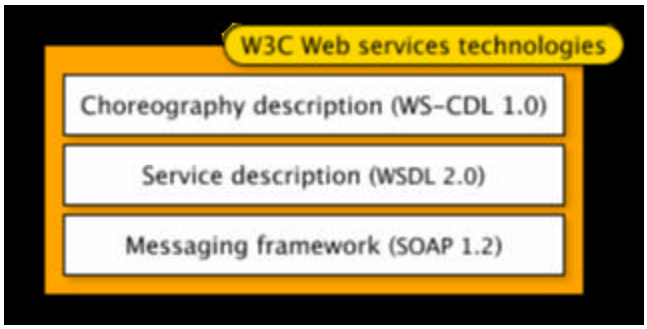


Figure 2-1. W3C Web services technologies

To realize this potential, there needs to be a set of standards and protocols on how the programs should interact. As illustrated in Figure 2-1 (borrowed from [66]), the W3C Web services technology (for now) has three core components. At the bottom of the stack is the XML based messaging framework, that is, SOAP 1.2, which is now a W3C recommendation. SOAP is a light weight protocol for structured message exchange and is independent of underlying transportation protocols, for example, HTTP or E-mail. It defines a high-level message structure, that is, the message envelop; a binding framework that describes the rules for defining a binding to a underlying protocol for SOAP message exchange; a processing model that specifies who and how a SOAP message should be processed; and a extensibility model that support the extension of SOAP.

One level up in the stack is the service description component, that is, WSDL (Web Service Description Language Version 2.0). WSDL is an XML based language for describing Web services. In WSDL, Web services are seen as a set of "endpoints" capable of exchanging and operating on messages. At the abstract level, WSDL describes a Web service in terms of the messages it sends and receives, and message exchange patterns are used to determine the sequence and cardinality (number of) messages. At the concrete level, it specifies the transport format and the address binding.

With a WSDL service description, a requester (an application that will try to use the service) will know how to use the service, that is, what is the format of the messages; what concrete protocols does the service use to exchange these messages; and what is the address of the endpoints providing this service [66]. WSDL does not depend on any particular message exchange protocol, but a binding to SOAP is being worked on at W3C.

On the top of the stack is the choreography description. W3C's Web Services Choreography Description Language (WS-CDL) is a work in progress. It is an XML-based language for describing peer-to-peer collaborations of Web Services "by defining, from a global viewpoint, their common and complementary observable behavior; where ordered message exchanges result in accomplishing a common business goal." [66].

Another necessary piece (for building successful Web services) that is not in the W3C Web services technology stack (yet?) is a protocol for service registration and discovery. UDDI (Universal Description, Discovery, and Integration) is such a protocol. "UDDI is a cross-industry effort driven by major platform and software providers, as well as marketplace operators and e-business leaders within the OASIS standards consortium." [63]. Service providers can use UDDI to advertise the services to a service registry and service consumers can use UDDI to discover services that they need. However, UDDI has limited search capability in the sense that it does not perform semantic matching.

2.4.2 The Semantic Web

"The Semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web." [65]. At the core of the Semantic Web technologies (at least for now) are RDF, the Resource Description Framework, and OWL, the Web Ontology Language.

RDF is a language for representing metadata (information about data, or in general, information about resources) in the World Wide Web. RDF describes things using RDF statements. An RDF statement is a triple of subject, predicate, and object, which can in general be interpreted as the resource "subject" has the property "predicate" with value "object". RDF identifies resources using URIs (Uniform Resource Identifier) and the RDF describing a particular resource does not have to be in one place and may be distributed across the Web. RDF statements can be represented in different ways, for example, as a graph, or using XML. RDF-S (RDF Schema) can be used to define vocabularies, for examples, classes and properties for use in describing resources.

As discussed in Section 2.2, formally defined ontologies are critical to achieving automated processing and integration of web resources as well as system interoperability in general. OWL, the Web Ontology Language, builds on top of RDF-S, provides additional vocabulary and a formal semantic. To meet the needs of specific communities of users and implementers, OWL comes in three increasingly expressive sublanguages, namely OWL-Lite, OWL-DL, and OWL-Full. OWL-Lite provides the support for the classification hierarchy and simple constraints; OWL-DL (the "DL" comes from "Description Logic", which is the foundation of OWL) provides the maximum expressiveness while retaining the computational completeness and decidability; and OWL-Full provides the "maximum expressiveness and syntactic freedom of RDF with no

computational guarantees" [65]. Every legal OWL Lite ontology is a legal OWL DL ontology and every legal OWL DL ontology is a legal OWL Full ontology.

OWL is an XML-based language and uses RDF's approach for representing data. For example, a class can be defined using OWL as follow:

```
<owl:Class rdf:ID="TravelTicket">
   <rdfs:subClassOf rdf:resource="#TravelThing"/>
</owl:Class>
```

RDF and OWL form the foundation for achieving the Semantic Web vision.

OWL-S [13] is an OWL-based Web service ontology by "The OWL Services Coalition". It defines a core set of markup language constructs for describing the properties and capabilities of Web services in a machine-readable way; therefore, it can support the automated service discovery, execution, monitoring, and composition.

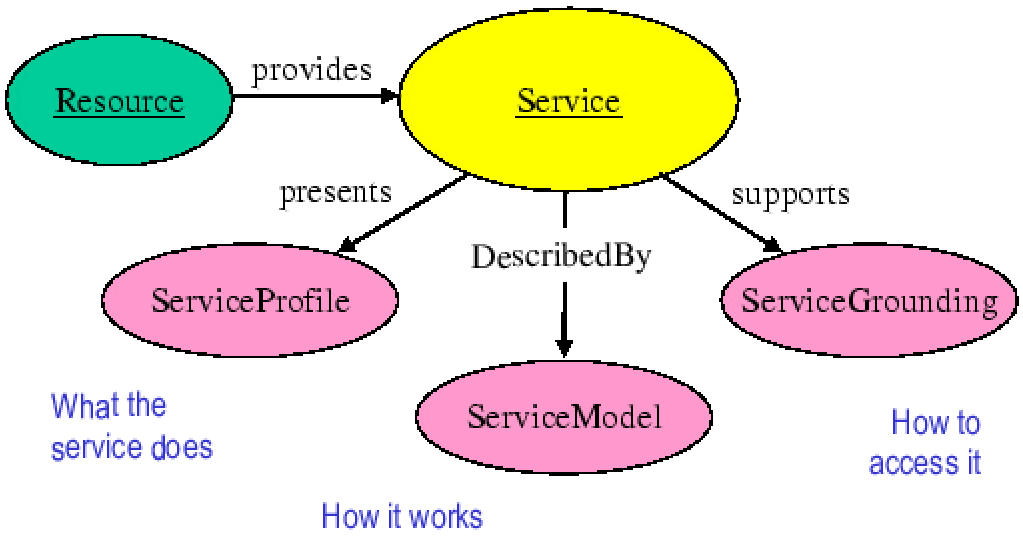


Figure 2-2. Top level of the service ontology

As illustrated in Figure 2-2 (borrowed from [13]), the class Service "provides an organizational point of reference for declaring Web services" [13]. An instance of Service "presents" an instance of the class ServiceProfile, which basically describes what the service does; is "describedBy" an instance of the class ServiceModel, which tells how

the service works; and supports an instance of the class ServiceGrounding, which tells how an agent can access the service.

The ServiceProfile class does not mandate any representation of services and OWL sub-classing may be used to define specialized representations. OWL-S defines such a subclass Profile (of ServiceProfile) that provides one possible way of representing service profiles. The class Profile describes a service in terms of three basic types of information: the provider information, the functionality description, and a host of features that specify the characteristics of the service.

The part (in a Profile) most relevant to our work is the functionality description. The OWL-S Profile represents two aspects of the functionality of the service: the information transformation (represented by inputs and outputs) and the state change produced by the execution of the service (represented by preconditions and effects). The inputs, outputs, preconditions, and the effects are the four types of parameters of a profile and are often referred to as IOPEs. The four types of parameters of a service profile may be specified using the properties hasInput, hasOutput, hasPrecondition, and hasEffect, respectively. An example piece of OWL-S ontology can be found in Section 3.2.

The ServiceModel and the ServiceGrounding classes are not directly relevant to our work and we will not go any further on that.

2.5 Related Work

The research summarized in this section is pertinent to the subject of this research as proposed in Chapter 1. The topics cover areas including service matching (and service discovery in general), reputation management, and agent learning.

2.5.1 Service matching infrastructures

The Stanford ABSI facilitator [26] was one of the earliest agent facilitators. In the Agent Based Software Interoperation (ABSI) architecture, the agents communicate with each other using KQML and normally, there is a facilitator running on each machine. The facilitator on a machine assists the agents running on that machine in collaborating (indirectly) with the agents running on other machines. An agent describes its capabilities to the facilitator in terms of what KQML performative(s) it can execute and the facilitator will use this information to conduct "content-based" routing, where the content language must be KIF. When an agent makes a request to the facilitator, the facilitator talks to the other facilitators to find an agent that can handle the request and (if one is found) the request will be forwarded to that agent. The facilitator can also help agents set up a direct communication among themselves to eliminate the overhead of message forwarding through the facilitators.

In the InfoSleuth Agent Architecture [54], the broker agents lie in the Generic Agent Layer. Individual agents advertise to the broker their information gathering and monitoring capabilities using semantic constructs from the InfoSleuth agent capability ontology. The capability ontology supports the descriptions of an agent's capabilities and an agent may constrain its capability advertisement to apply to only a selected set of concepts, relationships, or instances from a particular application domain. Service

matching is achieved "via the semantic constraint satisfaction services of the broker" [54] and by checking the syntactic constraints of their interfaces. InfoSleuth II explored the idea of multiple, specialized brokers working collaboratively to recommend service provider agents. The main motivation was to break the barrier to scalability and single point of failure.

The facilitator in the SRI Open Agent Architecture (OAA) [43] is responsible for coordinating agent communications and cooperative problem-solving. Service provider agents inform the facilitator agent of their capabilities and the facilitator maintains a knowledge base of this capability information. When a service consumer has tasks to be performed, it makes a request to the facilitator agent, which will then delegate the requests to the appropriate agent(s) for execution and finally return the results back to the service consumer agent. The facilitator "can employ strategies and advice given by the requesting agent, thus resulting in a variety of interaction patterns that may be instantiated in the satisfaction of a request". The facilitator may also be used to store global data and enable the blackboard-style agent interaction. The communication language used in OAA is ICL, Interagent Communication Language. ICL has two layers, a conversation layer and a content layer. The conversation layer is "similar in spirit to that provided by KQML" [43], and the content layer is based on Prolog. All requests are made using ICL.

LARKS, Language for Advertisement and Request for Knowledge Sharing, [62] is an agent capability description language developed at CMU. It describes an agent's capability by specifying the "Context", "Types", "Input", "Output", "InConstraints", "OutConstraints", and "ConcDescription". The "Context" slot specifies the local domain of the agent; the "Types" slot can be used to optionally define the data types used in this description; the "Input" and "Output" slots specify the input and output variables (parameters); the InConstraints" and "OutConstraints" are used to specify logical constraints on input and output respectively; and the "ConcDescripitons" slot allows you to give the definition of the concepts used in the description. The concept description language is a description logic based language called ITL (Information Terminological Language), which can be used to define local ontologies.

The matchmaking method used in LARKS is flexible and powerful. It has five filters each addresses the matching process from a different perspective. "Context matching" determines if two descriptions are in the same or similar context; "profile comparison", "similarity matching", and "signature matching" are used to check if two descriptions are syntactically match; "Semantic matching" checks if the input/output constraints of a pair of descriptions logically match. Based on the need of a specific application domain, these filters can be combined to achieve different types/levels of matching. In "Complete matching" mode, all the filter stages are considered; in "Relaxed matching" mode, the first three filters are used; in "Profile matching" mode, only the context matching and profile comparison are performed; in "Plug-In matching" mode, signature matching and semantic matching are performed. Therefore, it does not have to be a "pure-logic" based matchmaking, but it does support a full-fledged logic matching when needed. This is especially important since an agent does not have to be intelligent, and agent technology is often seen as a new software-engineering paradigm.

In [56], the authors discussed a matching algorithm for service matching in the DAML-S (the predecessor of OWL-S) framework. The degree (or level) of a service match is determined by the degree of parameter matches. There can be four levels (in descending order of preference) of parameter matches: exact match, plugin match, subsumes match, and fail. The degree an output parameter outA of the advertisement matches the corresponding output parameter outR of the request is decided as follow (from [56]):

degreeOfMatch(outR, outA):

if outA = outR then return exact

if outR subclassOf outA then return exact

if outA subsumes outR then return plugIn

if outR subsumes outA then return subsumes

otherwise, return fail.

The degree of input parameter matches is computed in a similar way but with the direction reversed.

The overall degree of a service match is then the lowest level of the parameter matches. However, in ranking the matches, the overall level (the lowest of all) of output parameter matches is the primary factor under the belief that, the requester expects first and foremost that the provider achieves the output requested at the highest degree.

The work in [40] takes a similar approach but from a different perspective. The degree of match is determined by the relationship between the advertised concept A and the requested concept R. The degree of match is an exact match if A = R; plugIn match if $A \supset R$; subsumes match if $A \subset R$; intersection match if $A \wedge R$ is satisfiable; otherwise, disjoint.

In [67], the authors discussed ways of comparing concepts in differentiated ontologies, that is, (different) ontologies evolved from a common base ontology. The concepts to be compared are represented in description logic. The paper describes roughly a dozen different measures that can be used to compute the compatibility of two concept descriptions. These measures fall into 3 main categories: the filter measures, the matching-based measures, and the probabilistic measures. The filter measures are basically based on how "close" the two concepts are in the concept hierarchy, and are inexpensive. The matching-based measures build and evaluate one-to-one correspondences between elements of concept definitions represented as graphs. The probabilistic functions require domain-specific knowledge of the joint distribution of primitives.

2.5.2 Other related work

Adaptation, that is, adapting to the environment and learning from the past experience, is a desirable property of agents. Traditional machine learning methods were designed for single, standalone agents/programs. In the world of multi-agent system, the situation has changed: a task is typically not accomplished by a single agent alone, but by the cooperation of a group of agents. Thus, the learning process may involve more than one

agent. Learning in a multi-agent world provides both opportunities and challenges. On the one hand, agents can help each other in the learning process, e.g., one agent could serve as the teacher of the other, or can at least provide some kind of feedback, which can make the learning easier. On the other hand, the agents need to coordinate with each other in the learning process and as a result, more uncertainties may present, and conflicts may occur.

One of the most commonly used learning methods in agent/multi-agent learning is reinforcement learning, especially the Q-learning algorithm (and its variants). In [30], the authors proposed a distributed reinforcement learning model (DRLM), a distributed Q-learning algorithm, and an architecture that allow agents to reward their peers' actions and share their experience. Ono and Fukumoto proposed in [55] a modular approach to multi-agent reinforcement learning. In this approach, the learning task is modularized and a mediator module is introduced to coordinate between the other learning modules and combine the learning results from them.

Other work on agent learning includes using game theory to model the agent interactions [5] [10]; distributed case-based reasoning and collective case-based reasoning [59]; using genetic programming to evolve behavioral strategies as in [31]; user-adaptation [39]; the list can go on.

Reputation is "usually defined as the amount of trust inspired by the particular person in a specific setting or domain of interest" [72]. It is an important factor in human interactions and the same is true in the agent world. In [72], the authors described two reputation mechanisms. Sporas is a simple reputation mechanism that provides a global reputation value for each user. After each rating, the reputation value is updated based on a formula. The second mechanism models pair-wise ratings (between two users) using a directed graph, in which the nodes represent the users and the weighted edges represent the most recent reputation rating given by one user to the other. With this graph, a more "personalized" reputation value of B (in the eye of A) can be computed from the ratings on the paths from A to B, based on certain criteria (e.g., the length of a path must be less than a given number N). The idea is that "social beings tend to trust a friend of a friend more than a total stranger".

The collaborative sanctioning model used in [49] is based on a concept called "encounter". "An encounter is an event between 2 agents (a_i, a_j) such that the query agent (a_i) asks the response agent (a_j) for a_j's rating of an object". The reputation of a_j in a_i's mind is defined here as the probability that in the next encounter, a_j's rating about a new object will be the same as a_i's rating.

2.6 Summary

This chapter provided an overview on a host of areas directly or indirectly related to our work. Agent researchers have been trying to find "killer applications" for the agent technology in the past decade and in my opinion it has not been very successful. The advent of the Web services, especially the need for a semantic web and "true" interoperability among Web services, provide both challenges and opportunities for the agent technology. One of the focuses of agent technology has been on the interoperability, for example, the research in the area of ontology and high-level agent communication

languages, and I think the achievements in agent technology could and should be applied to the Web services. In my opinion, the work on Semantic Web (in effect) is a major effort in that direction, and this provides huge opportunity for the agent technology and I think the agent technology may finally find its place in the industry.

However, existing works largely overlooked some important issues in agent service matching, for example, the issue of whether and how factors other than the service descriptions could play a role in service matching. This work is an attempt to address these issues by developing a method that the middle agents my use to establish and refine an agent's capability model based on an agent's performance history as well as the distribution of services.

3 An Abstract Multi-Agent System Model

This chapter introduces a simple multi-agent system model, the agent service description basics, and a simple domain ontology to lay the ground work for the discussions in the later chapters.

3.1 The Abstract Multi-Agent System Model

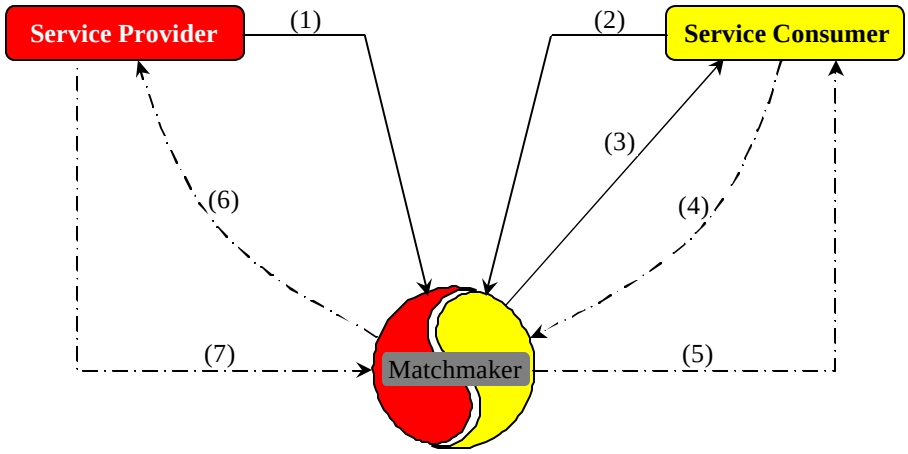
This work falls in the category of middle agent, therefore, the multi-agent system model described here is one that is based on the middle agent framework. Defining an abstract model would allow the discussion of this work without relying on any specific agent platforms. In this context, a multi-agent system (MAS) is an abstract system with three types of agents: the **service provider agents** (or providers, in short) that offer (or provide) services; the **service consumer agents** (or consumers, in short) that consume the services (provided by the service providers); and the **middle-agent** that helps to establish the connections between the providers and the consumers. There can be multiple middle-agents in a given agent system, but to simplify the discussion, we assume there is only one middle agent.

In [34], the authors discussed different types of middle agents, for example, matchmakers, brokers, and mediators. We will use a matchmaker agent as an example of a middle agent hereafter, and the ideas discussed should be generally applicable to the other types of middle agents as well.

The term "consumer agent" here is similar to the term "requester" in [34]. It is called "consumer agent" because in this work we will deal with such issues as feedback and satisfactions, and the term "consumer agent" may be more appropriate. Either "requester" or "consumer" may be used in the rest of this writing, whichever is more appropriate in the specific context. An agent may play the role of a provider in one specific transaction and play the role of a consumer in others.

Figure 3-1 illustrates the interactions (or operations) that are of interest to this work.

(1) A service provider agent may advertise its capabilities, that is, the descriptions of what services it can and intends to perform upon request, to the matchmaker. This voluntary process is called "advertise".



Note: The operation numbers do not imply the order of operations.

Figure 3-1. Operations in the matchmaker framework

(2) A service consumer agent asks the matchmaker agent to recommend one or more providers that can perform services or tasks of a given description. Such a request is called "recommendation request", or "request" for short.

(3) Upon the receipt of a recommendation request, the matchmaker performs "service matching", that is, tries to find service providers whose descriptions match that of the request. If some matches are found, recommendations will be made to the consumer agent as appropriate. The matches (if any) may be the result of a local lookup of its listings, or the matchmaker may ask other matchmakers (if available) to find the matches.

(4) A consumer agent may provide feedback to the matchmaker agent regarding its degree of satisfaction about the service(s) performed by providers previously recommended by the matchmaker agent. This process is called "feedback".

(5) The matchmaker agent may solicit feedbacks from the consumer agents that have previously received recommendations.

(6) The matchmaker may optionally ask a service provider agent if it can perform tasks of a certain description. The process is called "capability query".

(7) A service provider agent may conform or disconfirm a capability query.

The operations (1) - (3) are basic operations and are common to most of the matchmaking frameworks. In this work, we introduce the operations (4) - (7) as an additional dimension to the matchmaking framework to allow the matchmaker to learn more about the service provider agents. An agent is not required to participate in the interactions described in (4) - (7), but it is highly encouraged to do so, possibly with incentives. An agent is not required to go through a matchmaker agent to find a service

provider, nor is it required to advertise its services. However, these individuals will not be of much interest to what are to be discussed in this work.

3.2 Agent Service Description

As mentioned in the previous chapters, agent service description and service matching have evolved over the years, and in recent years, the focus has been on the use of service ontologies, e.g., OWL-S. The principles discussed in this work are independent of any specific service ontology but we will use the OWL-S framework for the purpose of discussion and experimental implementation.

In the OWL-S framework, a service description has three components: the service profile, the service model, and the service grounding. A service "presents" a service profile, is "described by" a service model, and "supports" a service grounding.

Service profile tells "what the service does" [13] and the main purpose is to enable service discovery. The service model tells "how the service works", that is, it describes what happens when the service is carried out [13]. This information would enable such activities as service composition, service monitoring, etc. Service grounding (or "grounding" for short) specifies the details of how an agent can access a service [13]. Grounding may specify such information as communication protocol, message formats, port numbers, etc.

This work mainly concerns with the issue of service discovery and therefore, the service model and service grounding are not of particular interest and we will focus on service profile.

The major components of a service profile are the parameters: input parameters, output parameters, preconditions, and effects (post conditions). The output parameters specify what the service can produce as outputs; the input parameters specify what inputs are needed in order to execute the service; the precondition parameters specify the conditions that need to be met for the successful execution of the service; and the effect parameters specify the effects that will result from the execution of the service. The inputs and outputs represent the information transformation while the preconditions and effects represent the state change [13].

Below is a simple example profile of a service that requires cash payment as the only input and produces an SUV as the only output.

```
...
<profile:Profile>
  <profile:hasInput>
    <process:Input rdf:ID="payment">
      <process:parameterType rdf:resource="http://my-ontology#Cash"/>
    </process:Input>
  </profile:hasInput>

  <profile:hasOutput>
```

```
        <process:Output rdf:ID="car">
          <process:parameterType rdf:resource="http://my-ontology#SUV"/>
        </process:Output>
      </profile:hasOutput>
    </profile:Profile>
    ...
```

As shown above, each service parameter has a "parameterType". In this example, the type of the payment parameter is "Cash", and the type of the car output parameter is SUV. The type Cash and SUV are classes (or concepts, types) defined in some domain ontology. In the rest of this writing, the term "type", "concept", or "class" may be used interchangeably to refer to a class defined in the domain ontology.

In the next section, we will briefly introduce a simple ontology as an example, and it will also be used in our experimental implementation.

3.3 A Simple Domain Ontology

The example domain is about airline tickets reservation services. The ontology has two main parts: part one defines a set of domain concepts; part two defines the ticket reservation service profile (as a subclass of the OWL-S Profile class) as well as the related properties and parameters.

Three major categories of classes are defined: airport locations, airline tickets, and payments. Figure 3-2 and Figure 3-3 illustrate the concepts hierarchies of the Airport Location, Flight Ticket, and Payment, respectively. A child node class is a subclass of the parent node class. For example, the class "Asia" is a subclass of "International". In OWL, this relationship may be represented as follow:

```
<owl:Class rdf:ID="Asia">
  <rdfs:subClassOf rdf:resource="#International"/>
</owl:Class>
```

Note that the pound sign "#" before "International" means the resource (the class, in this case) is the one defined in the current name space, not some "International" class defined in other name spaces.

The second part is about the airline ticket reservation service profile. In the OWL-S framework, a service presents a profile. We define the Travel Service Profile class as a subclass of OWL-S Profile class, and then define Flight Reservation Service Profile class as a subclass of the Travel Service Profile. The descriptions in OWL are as follow:

```
<owl:Class rdf:ID="TravelServiceProfile">
  <rdfs:subClassOf rdf:resource="&profile;#Profile"/>
</owl:Class>


<owl:Class rdf:ID="FlightReservationProfile">
```

```
    <rdfs:subClassOf rdf:resource="#TravelServiceProfile"/>
</owl:Class>
```

The Flight Reservation Service Profile has three input parameters and one output parameter. The input parameters are: payment with parameter type Payment; arrival airport with type Airport Location; and departure airport with type Airport Location. The output parameter is ticket with parameter type Flight Ticket.

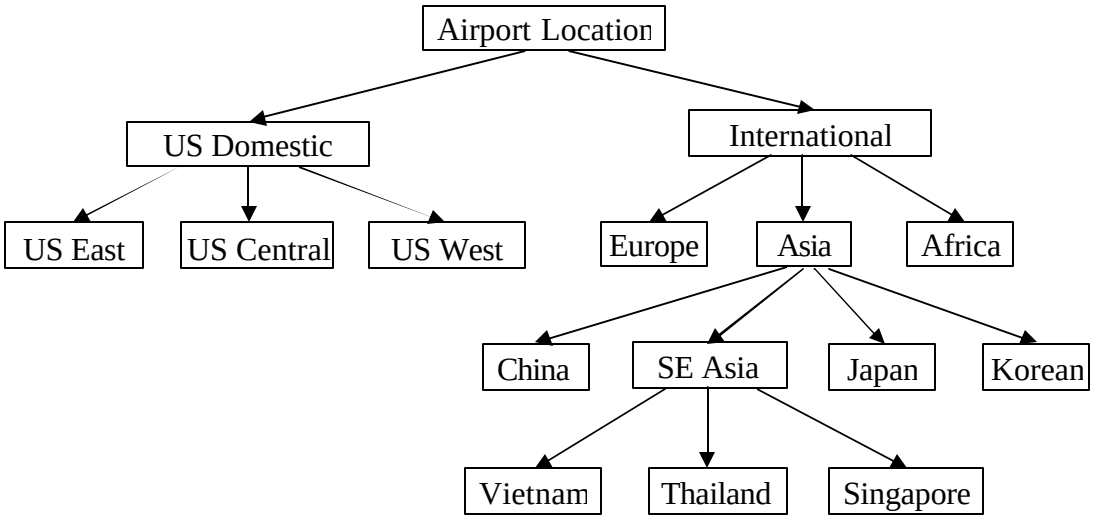


Figure 3-2. The airport location class hierarchy

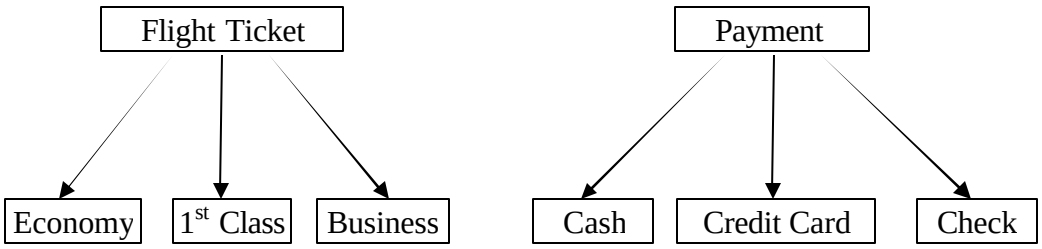


Figure 3-3. Class hierarchies for flight ticket and payment

With the ontology as defined above, a typical service profile description that sells flight tickets from US to Asia would look like:

```
...
<travelOnto:FlightReservationProfile rdf:ID=”GoldenBridgeTravel”>
   <profile:hasInput>
      <process:Input rdf:ID="departureAirport">
         <process:parameterType rdf:resource="&travelOnto;#UsDomestic"/>
      </process:Input>
   </profile:hasInput>

   <profile:hasInput>
      <process:Input rdf:ID="arrivalAirport">
         <process:parameterType rdf:resource="&travelOnto;#Asia"/>
      </process:Input>
   </profile:hasInput>

   <profile:hasInput>
      <process:Input rdf:ID="payment">
         <process:parameterType rdf:resource="&travelOnto;#CreditPayment"/>
      </process:Input>
   </profile:hasInput>

   <profile:hasOutput>
      <process:Output rdf:ID="flightTicket">
         <process:parameterType rdf:resource="&travelOnto;#FlightTicket"/>
      </process:Input>
   </profile:hasInput>
</travelOnto:FlightReservationProfile >
...
```

In the example, travelOnto (as in “travelOnto:FlightReservationService”) is the name space of the travel ontology; while “&travelOnto;” is an entity reference (an XML mechanism) to the full URI of the travel ontology.

3.4 Summary

In this chapter, we introduced an abstract multi-agent model, the agent service description in the OWL-S framework, and an example domain ontology. This laid down the ground work for the later discussions on service matching, agent capability modeling as well as for the discussion on the quantitative evaluation of the performance of candidate service providers.

To simplify the problem, we make the following assumptions:

- All the agents, including the matchmaker agent, in the system share common domain ontologies. The issue of ontology translation and/or semantic resolution may be addressed separately, for example, as discussed in [58].
- We assume a cooperative environment, in which all agents play in a cooperative way.
- The issues of trust, security, and privacy, though related to this work, may be addressed separately and we consider these issues orthogonal to what is discussed here.

4 Service Description Matching

In this chapter, we discuss the semantic matching of service descriptions, that is, to determine if the description of an advertised service semantically matches that of a requested service.

4.1 An Interpretation of Service Descriptions and Service Matching

As discussed in Section 3.2, service profile is the component in a service description that is of particular interest to this work. A service profile may have a set of parameters: the input parameters, output parameters, precondition parameters, and effect parameters. The preconditions and effects are outside the scope of this work in the sense that they are not part of the functional specification and that they do not participate in the agent capability modeling (Chapter 1) and quantitative service matching (Chapter 1). Therefore, the elements to be considered here are the input parameters, the output parameters, and the type of the profile itself - specific profile types may be defined (as subclasses of the Profile class in DAML-S) in the domain ontology, for example, a travel service profile.

A service description (in this context) may service one of the two purposes: to describe an advertised service or to describe a requested service. A service profile (or profile) is part of a service description. In the context of service matching, we are only concerned with service profiles therefore the term "service profile" (or profile) may be used interchangeably with the term "service description".

When a service description describes an advertised service, the interpretation is as follow. The profile type declares the type or category of the service. For example, a profile with type "Book Printing" declares that this service is about printing books, not about book sales service or any other services. The set of output parameters declares what the service can produce when the service is successfully performed. An output parameter with type T_{out} declares that anything (if requested) of type T_{out} can be produced. For example, in a car sales service, an output parameter with type "Car" declares that when such a service is performed (successfully), it can produce any types of cars as requested – a coupe, a sedan, an SUV, etc. The set of input parameters declares what may be needed as input in order for the service to be successfully performed. An input parameter with type T_{in} declares that the input requirement specified by this input parameter can be fulfilled with anything of type T_{in}. For example, in a car sales service, the payment input parameter with type "Check" declares that any check payment is ok, be it a personal check or a certified check.

When a service description describes a requested service, the interpretation is a bit different. The profile type declares the type or category of services the requester is looking for. For example, a profile with type "Book Printing" declares that the requester is looking for a book printing service, not book sales service or any other services. The set of output parameters declares what the requester would like the service to be able to accomplish. An output parameter with type T_{out} declares that the output (with respect to this parameter) the requester wants to accomplish may be anything of type T_{out}. For

example, in a car sales service, an output parameter with type Car declares that the requester would like the service to be able to produce something of type Car – it may be Sedan, SUV, etc, or even all types of cars. The set of input parameters declares what the requester can provide (if needed, to the service provider) to have the service performed. An input parameter with type T_{in} declares that the requester (if needed) may provide an input with type T_{in}. For example, in a car sales service, the payment input parameter with type "Check" declares that the requester may pay by check – but it does not promise that it can pay with certified check, for example.

Roughly speaking, service description matching is the process of comparing two service profiles (or descriptions), one advertised service profile "adv" and one requested service profile "req", to determine if the advertised service may fulfill the request. If so, such a (adv, req) pair is called a match, or the adv is said to match the req. Ideally, the service profile type of adv and req should be the same or compatible; the adv should be able to produce every output specified by req; and in order to successfully execute such a service, the req should be able to provide every input specified/required by the adv. In the rest of this section, we will try to give a more formal definition.

Definition 4-1

> In matching a given pair of service descriptions d_1 and d_2, each of the two service descriptions takes a different role: one (e.g., d_1) takes the role of describing the service needs, and the other (e.g., d_2) takes the role of describing the service offerings (or capabilities). **Service description matching** is the process of determining whether a service offering described by d_2 can potentially meet the need described by d_1. If so, it is said that **d_2 matches d_1**, or **d_1 is matched by d_2**.

For example, in matching an advertised service description and a service recommendation request, the request specifies the needs and the advertisement describes the service offerings or capabilities.

Definition 4-2

> In deciding if a service profile d_2 matches another service profile d_1, a pair of parameters p_1 (of d_1) and p_2 (of d_2) is said to be a **sufficiently matched parameter pair** (**SMPP**) if and only if:
>
> $$(p_1.io = p_2.io) \wedge (p_1.type \subseteq p_2.type)$$
>
> Where $p_x.io$ is the parameter category (e.g., input/output) of the parameter p_x; $p_x.type$ is the parameter type of the parameter p_x.

The predicate **$smpp(p_1, p_2)$** is true if and only if (p_1, p_2) is a sufficiently matched parameter pair.

First, in order for (p_1, p_2) to be an SMPP, the two parameters need to come from the same category, in other words, either both are input parameters or both are output parameters; an input parameter can not be used to match an output parameter, nor vice versa. Second, the parameter type ($p_2.type$) of the capability description must be the same as or is a super type of the parameter type ($p_1.type$) of the service needs description. In the case of output parameter matching, the interpretation is that the advertised service

should be able to at least produce what the request requires; in the case of input parameter matching, this means that in order for a sufficient match, the advertised service should be able to accept what the service requester can provide (with respect to this parameter) so as to perform the service.

Definition 4-3

> In matching two service profiles, a **parameter pairing** is an association of the parameters from one service profile (s1) to those in the other (s2), such that (1) a parameter (from either profile) may be used at most once; and (2) an associated pair of parameters must be from the same parameter category, that is, either both are input parameters or both are output parameters. A parameter pairing can be represented as a set of parameter pairs (p1, p2), where p1 and p2 are from the two service profiles s1 and s2, respectively.

This is an abstract definition. The exact strategy of how the parameters are paired (or associated) is left unspecified.

A service profile actually specifies a set of services, that is, all the services that can be described by the profile. For example, a travel service may include airline ticket reservation service, hotel reservation service, etc. In other words, any service that falls into the travel service category is a travel service. We say a service profile p defines a **service space**, a space that encompasses all the services whose profiles are the same or more specific than the profile p.

Definition 4-4

> A service profile defines a **service space**, which is a triple *S*(P, I, O), where P is the service profile type (class), I is the set of input parameters, and O is the set of output parameters. A service with profile type P', input parameter set I', and output parameter set O' falls in the service space *S* if and only if:
>
> 1) $P' \subseteq P$, and
> 2) There exists a parameter pairing *pp* such that
>
> $(\forall\ x \in O'\ (\exists\ y \in O\ ((x, y) \in pp\ \wedge\ smpp(x, y)))\ \wedge$
>
> $(\forall\ v \in I\ (\exists\ u \in I'\ ((u, v) \in pp\ \wedge\ smpp(u, v))$

The first condition in Definition 4-4 requires that *S* has the same or broader service profile type; the second condition requires that the service does not have more output capability than that specified in the service space *S* and that it does not require fewer inputs than that specified in the service space *S*.

Therefore, when a service provider posts an advertisement, it actually advertises a service space, that is, a set of services that it claims it can perform. Similarly, when a service consumer agent makes a service matching request, it actually wants a service provider agent that can perform (any) services in the service space defined by the service description given by the consumer. Service description matching is then an issue of determining the relationship of the two service spaces: the advertised service space and the requested service space. In the ideal case, if the requested service space falls in the

advertised service space, that is, the advertised service can produce any output the requester needs and requires no more input than what the requester can provide. There are less ideal cases in which the requested service space does not fall completely within the advertised service space but overlaps with it – these are potential matches.

Service space can be used as a tool for describing service description matching at the conceptual level. As illustrated in Figure 4-1, when the advertised service space and the requested service space overlap completely, the match is an exact match; when the requested service space falls completely within the advertised service space and not vice-versa, it is a plug-in match in the sense that the advertised service has sufficient capability and can be plugged in to fulfill the request; when the advertised service space falls completely within the requested service space and not vice-versa, it is a subsumes match; when the two service spaces overlap but neither falls completely within the other one, it is an intersection match; when the two service spaces are disjoint, it is not a match of any kind.

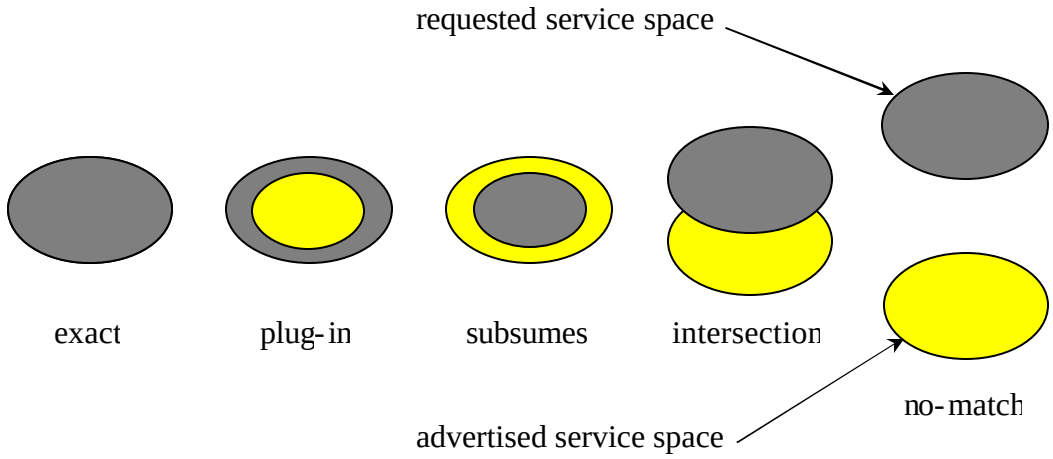


Figure 4-1. Levels of matches

Service description matching involves 1) profile type matching, 2) parameter pairing, and 3) parameter matching. Profile type matching is to make sure the type of an advertised profile is compatible with that of the requested service profile; parameter matching determines whether (and to what degree) two given parameters match; and parameter pairing associates (or pairs) the parameters from the two service profiles. Parameter matching is the basis for parameter pairing and both processes are mainly based on the parameter types.

So far we have been using the term “service description matching” and avoiding the term “service matching”. In this work, “service matching” is the process of finding the best service providers based on factors more than service descriptions, for example, the performance of the service providers, the service distributions, etc. This chapter focuses on service description matching, and service matching will be discussed in Chapter 1.

The next two sections will focus on parameter matching and parameter pairing, after which we give a summary on service description matching.

4.2 Parameter Matching

Parameter matching determines if a parameter p2 matches another parameter p1 and is the basis for service description matching. The OWL-S service ontology is not quite mature yet and (at least for now) the main construct that can be used for parameter matching is the parameter type. We take a similar approach as what are described in [40] and [56]. We define four levels of parameter matches, namely exact match, plug-in match, subsumes match, and intersection match.

Definition 4-5

With respect to whether and how a parameter p2 (of an *advertised* service description) matches p1 (of a *requested* service description), define:

exact match: if p2.type = p1.type; otherwise,

plug-in match: if p2.type $\supset$ p1.type; otherwise,

subsumes match; if p2.type $\subset$ p1.type; otherwise,

intersection match: if p2.type n p1.type ? f ; otherwise,

No match.

The term "**match**" is used loosely to refer to any of the four match levels unless explicitly stated in the context; the term "**sufficient match**" may be used to refer to both exact matches and plug-in matches; the term "**potential match**" may be used to refer subsumes matches and intersection matches, since in these cases, the advertised service may potentially fulfill the requirements set forth by the request.

There is no need to explain the "exact match". For plug-in match, it basically means that p2, though not exactly the same as p1, may be plugged in to fulfill the requirement(s) as specified in p1. For example, if the requested service has an output parameter of type "HDTV" (e.g., in a TV-selling service), then an output parameter of type "TV" in an advertised TV-selling service would logically fulfill the requirement of selling "HDTV". Subsumes match is a potential match. In the example above, if the requested service has an output parameter of type "TV" and the advertised service has an output parameter of type "HDTV", then this would be a subsumes match since the type TV subsumes HDTV. In this case, the service of selling "HDTV" may only potentially fulfill the requirement of selling TV.

Intersection match is another form of potential match in which p2.type and p1.type have some common subset(s). For example, the class electronic equipment and the class entertainment equipment have common subsets (such as TV), but there are some electronic equipments that are not entertainment equipments and there are some entertainment equipments that are not electronic equipments.

These are just possible levels of matches since some may prefer to disable/turn off subsumes-level match and/or intersection-level match, which are not "real" matches in

the logical sense. The four match levels defined here are slightly different from that defined in [40].

In OWL-S, a parameter may have multiple parameterType properties, that is, the parameter may take any one of the specified types. The above described definitions should still be applicable because such a description may be transformed into an equivalent description in which the parameter takes a single type, and this type is defined as the disjunction of all the parameter types the parameter can take. However, this may require that the OWL inference engine must support disjunction.

4.3 Parameter Pairing

To match two service profiles, we need to figure out which parameter in one service profile corresponds to a parameter in the other service profile. Parameter pairing is the process that establishes such associations. At the first glance, this seems to be a simple issue or even a non-issue. But it could become complicated because a service requester may not know exactly what parameters an advertised service profile may have when the recommendation request is made to the matchmaker agent, especially given the distributive nature of OWL and OWL-S.

As discussed earlier, the main factor in parameter matching is the parameter's type. But in a service description multiple parameters may have the same type. For example, in the flight ticket reservation service, there is a departure airport and an arrival airport, both of which may have type Airport Location. It is possible to require that all the parameters of every profile be explicitly defined in the domain ontology and require that an advertisement or a request must reference the profiles and parameters exactly as defined, but we think that such requirements are too strong, especially for large, open agent systems or other similar Internet based frameworks.

Our approach to parameter pairing is a two-step one. In the first step, we pair the parameters (from two given profiles) based on the parameter URIs. That is, two parameters are considered a corresponding pair if they are references to the same resource explicitly defined somewhere, for example, in a domain ontology. If every parameter that needs to be matched is matched, then we are done. If not, we go to step two, the bipartite matching, to pair the (remaining) parameters. The goal here is to match as many pair of parameters as possible and the maximum bipartite matching algorithms can make sure that the maximum number of parameters be paired.

The example in Figure 4-2 illustrates why it is important to match as many pairs of parameters as possible. In this example, the advertised service has two output parameters a1, a2 and the requested service has two output parameters r1 and r2. Suppose a1 matches both r1 and r2, but a2 only matches r2, that is, the possible pairs are {(a1, r1), (a1, r2), (a2, r2)}. If we pair a1 and r2, then a2 and r1 will not match and we will not get an overall match. A better way is to use a1 to match with r1 and a2 to match r2, then all of the parameters are matched. When such a parameter pairing problem is transformed in to a bipartite matching problem, the maximum bipartite matching algorithms can make sure that the maximum number of parameters be paired.

4.3.1 Parameter pairing using maximum bipartite matching

Parameter pairing problem may be transformed into a bipartite matching problem, and then the maximum bipartite matching algorithms may be applied to find the maximum parameter pairing.

Suppose G = (V, E) is an undirected graph, where V is the set of vertices and E is the set of edges. A **matching** is a subset of edges $M \subseteq E$ such that for all vertices $v \in V$, at most one edge of M is an incident on v (that is, has v as one of the two end points). A maximum matching is a matching of maximum cardinality, that is, a matching M such that for any matching M', we have |M| >= |M'|.

A **bipartite graph** is an undirected graph whose vertex set V can be partitioned into $V = L \cup R$, where L and R are disjoint and all edges in E go between L and R. We do not need to go into the details and it suffices to say that there are tractable algorithms (e.g., using the Ford-Fulkerson method) that can find the maximum matching in a bipartite graph.

adv.outputs:
a1, a2

req.outputs:
r1, r2

Possible parameter matches:
(a1, r1), (a1, r2), (a2, r2)

Maximum matching:
{(a1, r1), (a2, r2)}

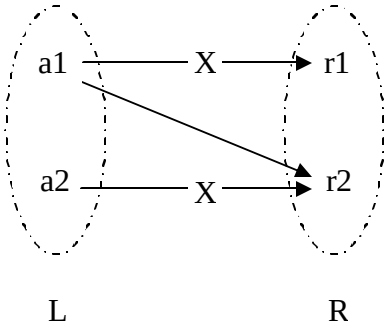


Figure 4-2. An example bipartite graph

Now let us see how to transform the parameter pairing problem into a bipartite matching problem. Let the parameters from the advertised service description form the vertex set L and the parameters from the requested service description form the vertex set R. Now draw an edge between a vertex v1 in L and a vertex v2 in R if and only if:

(1) The parameter v1 and the parameter v2 are of the same parameter category, that is, either both are input parameters, or both are output parameters. There should be no edge from an input parameter to an output parameter or from an output parameter to an input parameter; and

(2) The two parameter types match (exact match, plug-in match, subsumes match, or intersection match), as discussed in the previous sections.

(3) Note that a vertex (i.e., a parameter) may be used more than once at this point.

For example, the transformed bipartite graph of the previous example is illustrated in Figure 4-2. Note that the edges marked with an "X" are those in the resulting maximum matching. Now that we have a bipartite graph, using any of the maximum bipartite matching algorithms would find us a matching with the maximum cardinality, that is, to find the most possible number of parameter pairs. If every parameter that needs to be matched is matched, we have a match; otherwise, the two given service descriptions are considered not match.

There can be more than one maximum cardinality matching. In the next section, we will discuss how to find the "best" matching, by transforming the problem into a maximum weighted, maximum cardinality bipartite matching problem.

4.3.2 Maximum weighted, maximum cardinality parameter pairing

There can be more than one maximum cardinality matching. To illustrate the consequences, we make some changes to the previous example. Suppose the possible parameter level matches are:

(a1, r1: exact), (a1, r2: exact), (a2, r1: subsumes), (a2, r2: exact)

There will be two maximum matchings in the bipartite graph:

{(a1, r1: exact), (a2, r2: exact)}, and

{(a1, r2: exact), (a2, r1: subsumes)}

We can see that the first matching certainly represents a better way of pairing the parameters than the second one. Therefore, what we want is a parameter pairing that not only maximizes the number of parameter pairs, but also pairs the parameters in the best way. This problem of finding the "best" parameter pairing may be modeled as a problem of finding the maximum weighted, maximum cardinality matching in a bipartite graph, or put it more precisely, a problem of finding the maximum cardinality matching that has the maximum total weight. We favor maximum weighted, maximum cardinality matching over maximum weighted matching because whether a parameter is matched determines whether the two service profiles match; while the weight determines the degree of a match, if there is a match.

To transform the parameter pairing problem into a maximum weighted, maximum bipartite matching problem, it is first transformed into a bipartite matching problem as described above, and then weights are assigned to the edges in the transformed bipartite graph. How the weights are assigned largely depends on the exact behavior you want to achieve. For example, suppose you may have three plug-in parameter pairs or you may have one exact parameter pair and two subsumes parameter pairs, which way would you prefer? In general, we would want

$$w(\text{exact}) >= w(\text{plug-in}) >= w(\text{subsumes}) >= w(\text{intersection}) \qquad \text{(EQ 4-1)}$$

Where w(level) is the weight assigned to the edges whose parameter match level is "level". That is to say, in general, we favor exact matches over plug-in matches; favor plug-in matches over subsumes matches and intersection matches.

With the weights assigned, a maximum weighted, maximum cardinality bipartite matching algorithm, for example, the one discussed in [47], may be used to find the "best" way of parameter pairing.

4.4 Put it Together

As discussed earlier, an advertised service profile matches a requested service profile if and only if the two service spaces (defined by the two service profiles) overlap, that is, the advertised service may potentially fulfill the needs of the requester. Depending how the two service spaces overlap, there can have different level (or degree) of matches.

Definition 4-6

The level that an advertised service description (with service space S_{adv}) matches a requested service description (with service space S_{req}) is defined to be:

exact match, if $S_{req} = S_{adv}$;

plug-in match, if $S_{req} \subset S_{adv}$;

subsumes match, if $S_{req} \supset S_{adv}$;

intersection match, if $(S_{req} \cap S_{adv} != f) \wedge \neg(S_{req} \subseteq S_{adv} \vee S_{req} \supseteq S_{adv})$;

no match, otherwise.

An exact match or a plug-in match is called a **sufficient match**; a subsumes match or an intersection match is called a **potential match**.

Service description matching may also be described in terms of profile type matching, parameter matching and parameter pairing as follow. An advertised service profile matches a requested service profile if and only if:

(1) The profile type of the advertised service (a.profileType) is compatible with the profile type of the requested service (r.profileType), in other words, a.profileType $\cap$ r.profileType != f ; and,

(2) Each output parameter of the requested service profile is matched by an output parameter in the advertised service profile; and,

(3) Each input parameter of the advertised service profile is matched by an input parameter in the requested service profile; and,

(4) A parameter can be used at most once in parameter matching.

Each output parameter from the requested service profile must have a matching output parameter from the advertised service profile because the output parameters represent what the service requesters want to achieve and have to be fulfilled in order to call it a match. Similarly, each input parameter from the advertised service profile must have a matching input parameter from the requested service profile because the input parameters represent what the service provider requires/needs in order to perform the advertised service.

However, in some cases, it might be desirable to relax the requirements on the match of input parameters, since a service requester (or consumer), though usually knows what it wants to achieve, may not necessarily know exactly what the service providers require in order for the providers to perform the services. That is, at the time of making a recommendation request, a requester may not know exactly what input parameters to specify.

The ultimate goal of service matching is to find the service providers that are most likely to perform the requested services with high degree of satisfaction. In that regard, service description matching is a major factor, but not the only factor. In Chapter 1, we will introduce a method with which the matchmaker agent may dynamically establish and refine the capability model of a service provider agent and in Chapter 1, we will discuss how such a capability model may be used in service matching.

5 The Agent's Capability Model

With OWL becoming a W3C recommendation and with the progress on OWL-S, the door is now wide open for the matchmaker agents to provide added values, that is, to go beyond service description based matchmaking. Formally defined domain ontologies and standardized service ontology not only allow semantic-level service matching, but also allow the matchmaker agents to establish and refine a service provider agent's capability model.

Agents are not created equal and the service provider agents that advertise services of identical descriptions may not necessarily have the same level of capabilities in performing the advertised services. Moreover, an agent may have strong and weak areas across its service offerings. Therefore, performance should be an integral part of an agent's capability model, and capturing an agent's strong and weak areas is just as important as telling good performers from bad performers.

Some agents might be smart enough to learn about the other agent's capabilities but it would be a heavy burden and it may not be very effective since an individual agent's experience is usually limited. A matchmaker agent, however, is in an especially "convenient" position to take this responsibility. For example, a matchmaker agent receives service advertisements as well as service match requests and therefore knows the service demands and the service offerings. Matchmaker agents usually have more resources (e.g., memory, computing power) than the other agents and it may stay around longer (have longer life span), too.

Moreover, this is certainly consistent with the matchmaker's ultimate goal, that is, to recommend the right service providers to the service consumers. Therefore, a matchmaker agent can, and should, do something to learn about the capabilities of the service provider agents.

In our multi-agent system model as discussed in Chapter 1, a service consumer agent may optionally provide feedback to the matchmaker agent on the service rendered, which may include such information as the exact description of the service performed and the satisfaction rating. Since the matchmaker agent is usually the one that recommends the providers to the consumers, it is legitimate and proper for the matchmaker agent to receive feedback from the consumer agents, or for the matchmaker agent to check with the consumer agents, "how was my recommendation?"

An agent capability model is established within a pre-specified scope called a **reference service domain (or reference domain, in short)**. The term "reference domain" is used loosely to refer to a category of services that, from the matchmaker's perspective, share the common core functionality and similar service profiles. A natural way to specify such categorization may be to follow the profile class hierarchy. For example, we may specify three reference domains for the travel service domain: the airline ticket reservation reference service domain, the hotel reservation reference service domain, and the rental car reference service domain.

The agent capability model has two major components, a service distribution model that is global to the reference domain, and a performance model that is specific to each advertised service. The domain ontology provides the basis and the concept hierarchy for the construction, update, and refinement of both components. In order to achieve this, we extended the OWL-S service ontology with constructs for supporting feedback from the consumer agents.

5.1 The Service Distribution Model

First of all, we give the term "service distribution" a more precise meaning.

Definition 5-1

> In this context, **service distribution** refers to the frequency of service request occurrences across the sub-domains of a pre-specified reference service domain.

For example, in the computer manufacture service reference domain, the distribution may be that 75% of manufacture requests are about desktops, 24% about laptops, and 1% about the other categories, e.g., servers.

Service distribution model is a key component to the agent capability model, although service "distribution" does not seem to have anything to do with an agent's "capability". Let us use an example to illustrate why service distribution matters.

Suppose the class "Car" has two subclasses, "GasPoweredCar" that accounts for 99% of all cars, and "SolarPoweredCar" that accounts for 1% of all cars. Without considering this distribution information, the level of match for (Car, SolarPoweredCar) would be the same as that of (Car, GasPoweredCar), because SolarPoweredCar and GasPoweredCar are both direct subclasses of the class Car, that is, both match levels are subsumes match. But with the distribution information, we know that the difference (in the degree of match) is huge, it is 1% vs. 99%.

The difficulty here is that the exact distribution is dynamic and usually unknown. What a matchmaker agent can do is to capture the service distribution through the interaction with the other agents. The distribution captured is "local" to the reference domain and "local" to the particular matchmaker agent's reach for a specific time period, and may not be exactly the same as the "global" distribution. This should not be a major problem and may actually be an advantage because the "local" information is usually more important for "local" transactions. If all the cars are solar-powered in the island country of Sunshine Paradise, the global distribution would be virtually irrelevant on the island.

Each reference domain (e.g., flight ticket reservation service) has associated with it a distribution model that captures the frequency of service request occurrences across the reference domain. Unless otherwise specified, the distribution refers to the **local service distribution** in the world that the matchmaker agent can see. For the purpose of capturing service distribution, a service profile *s* is said to be an **occurrence** in a service space *S* if *s* falls in the service space *S*. In this context, we are only concerned with the occurrences seen by the matchmaker agent as requested service profiles. Occurrences as advertised service profiles may be counted in, too, if so desired.

We propose two service distribution models here, a service distribution model that is based on the concept of service space and a simplified, parameter type distribution model that approximately tracks the service distribution based on the distribution of parameter types. The service distribution model is introduced in Section 5.1.1 and the parameter type distribution model is introduced in Section 5.1.2.

5.1.1 The service distribution model

The service distribution model tracks the distribution of services across the reference domain. The central part of a service distribution model is a DAG (Directed Acyclic Graph) where a node represents a unique service profile and a directed edge goes from a child node to a parent node – a node c is a child of node p if the service space of c falls completely inside the service space of p. It is a DAG because you can not have the situation in which a node x is an ancestor of node y and at the same time node y is an ancestor of node x.

Figure 5-1 illustrates an example distribution model for a computer manufacture service reference domain.

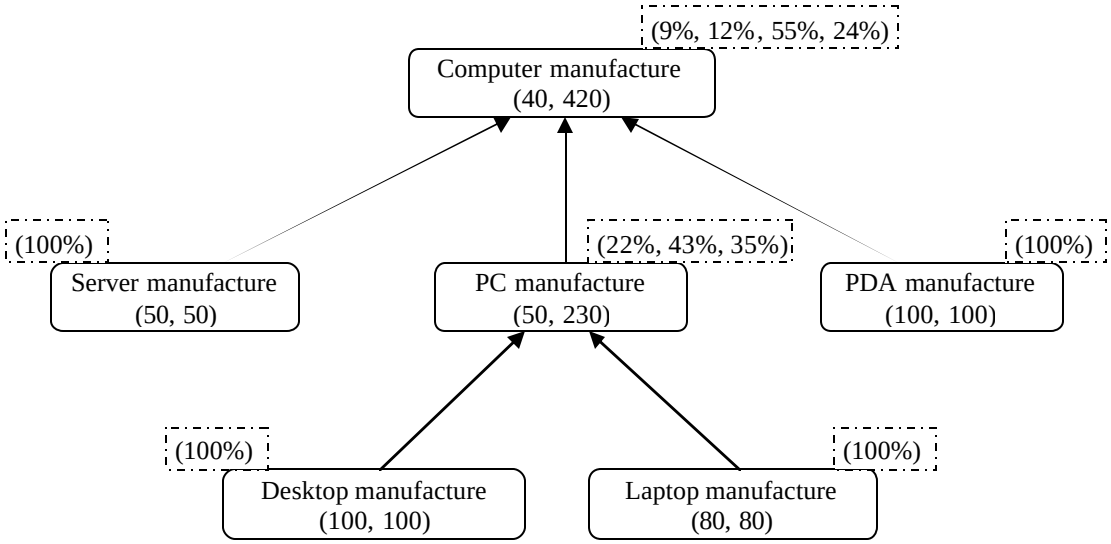


Figure 5-1. Computer manufacture service distribution model

Each node has two occurrence counts, a direct occurrence count and a total occurrence count. The direct occurrence count represents the occurrences that come from the requests whose profiles are exactly the same as that of the node. The total occurrence count of a node is the total number of occurrences of the requests that fall in the service space of the node. More precisely, for a request *r* and a node *n* in the distribution graph, the direct occurrence count of node *n* is increased by 1 if and only if $S_r = S_n$, where S_r is the request service space and S_n is the service space of node *n*; the total occurrence count

is increased by 1 if and only if $S_r \subseteq S_n$. The total occurrences and the direct occurrences of a leaf node should be identical.

Each node also has a distribution vector, each element of which is the percentage of a child node total in the total of the current node, except that the first element is the percentage of the node's direct occurrences in the node's total.

For example, in Figure 5.1, the node "PC manufacture" has a total of 230 occurrences, that is, 230 requests are about PC manufacture, 50 of which are directly about PC manufacture (no specifics were given on the exact sub-categories). The distribution vector of the node shows that direct occurrences accounts for 22% of the total occurrences; Desktop manufacture (with 100 occurrences) accounts for 43% of the total occurrences; Laptop manufacture (with 80 occurrences) accounts for 35% of the total occurrences.

The distribution vector can be interpreted in conventional probability terms. For example, the vector (22%, 43%, 35%) associated with "PC manufacture" can be read as

Pr(PC manufacture | ~Desktop manufacture, ~Laptop manufacture) = 0.22,

Pr(Desktop manufacture | PC manufacture) = 0.43,

Pr(Laptop manufacture | PC manufacture) = 0.35

The way the probability information is organized in the distribution vectors makes it easy for update.

The occurrence counts are used to capture the dynamic service distribution and for computing the distribution vector. It is the distribution vector that will be used by the system, for example, to compute the degree of a match.

The distribution model is updated dynamically. For example, if the matchmaker agent receives a service request of "Laptop manufacture", then both the direct occurrences and the total occurrences of the node "Laptop manufacture" will be increased by 1, and the total occurrences of the node "PC manufacture" and "Computer manufacture" will be increased by 1, too.

One dimension that is not discussed yet is the time dimension. Will the distribution be the cumulative distribution since the "very beginning" or will it be the distribution within a time window, for example, two months? This may depend on the specific application domains. In more stable situations the cumulative distribution might make more sense; for more volatile situations, some types of moving average may be more appropriate.

The distribution model can be initialized with some initial distribution to take advantage of the prior knowledge (if any) on service distribution.

5.1.2 The parameter type distribution model

In this section, we introduce a simplified model in which the service distribution is approximated based on the distribution of parameter types. The idea is that a service profile is basically defined in terms of a set of parameters and therefore the distribution of the parameter types may be used to estimate the distribution of the services. Compared

with the service distribution model introduced in Section 5.1.1, the main advantages of this parameter type distribution model are its simplicity and computational efficiency. It is less accurate, however, because it relies on the Assumption 5-1 below, which may not be always true.

The distribution of parameter types may be defined similarly as the distribution of services. An **occurrence** of a parameter type *T* in the reference domain is defined as a reference of the type *T* or its sub-type *T'* as the type of a parameter in a service profile. In this work, we are only concerned with the occurrences of parameter types in the requested service profiles, that is, the service demands. The **parameter type distribution** refers to the frequency of parameter type occurrences in the reference domain.

Assumption 5-1

> In a specific reference domain, the distribution of one parameter type is independent of the distribution of the other parameter types.

With this assumption, the service distribution can be approximately computed from the parameter type distribution, since a service description is largely defined by the set of parameters of the profile, that is, for a service profile s that has n parameters, we have:

$$P(s) = P(t_1, t_2, ..., t_n) = P(t_1) * P(t_2) ...* P(t_n) \qquad \text{(EQ 5-1)}$$

Where $P(s)$ is the probability that an occurrence of a service would have the profile s; $P(t_1, t_2, ..., t_n)$ is the probability that the parameters of service profile s would take the types $t_1, t_2, ..., t_n$; and $P(t_i)$ is the probability that the i^{th} parameter takes the type t_i.

The parameter type distribution model is also a DAG (Directed Acyclic Graph) and is constructed based on the domain ontology type/class graph. It is a DAG because the domain ontology graph should not contain cycles, that is, a class can not be a sub-class of its own sub-class. It includes all the type nodes that are used as parameter types in the reference domain and those whose super types are used as parameter types. Figure 5-2 illustrates the distribution graph for the region parameter of some reference domain.

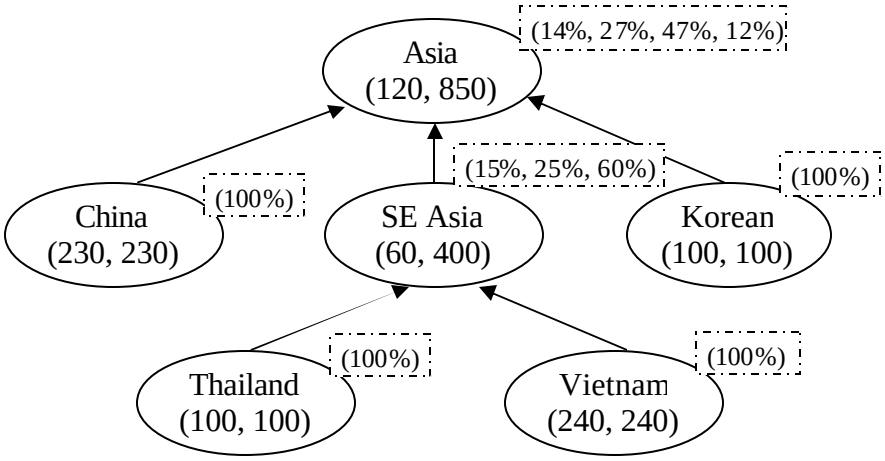


Figure 5-2 The distribution model

As with the case of the service distribution model, each node in this parameter type distribution model also has two occurrence counts, a direct occurrence count and a total occurrence count. The direct occurrence count represents the occurrences that come from the direct use of the type itself; the total occurrence count of a type is the total of its direct occurrence count plus all the direct occurrence counts of its descendents. The total occurrences and the direct occurrences of a leaf node (a node that does not have any sub-types) should be identical.

A node also has a distribution vector, each element of which is the percentage of a child node's total in the total of the current node, except that the first element is the percentage of the node's direct occurrences in the node's total. As with the case of the service distribution model, the occurrence counts are used to compute the distribution vector and it is the distribution vector that will be used by the system.

We can see that the parameter type distribution model and the service distribution model have similar structures and each has advantages and disadvantages. The matchmaker may actually use different models at the same, for example, use service distribution model for one reference domain while use parameter type distribution model for another. The experiment in this work uses the parameter type distribution model because of its simplicity.

5.2 The Performance Model

As discussed earlier, the agent capability model has two major components: a service distribution model and a performance model. While the service distribution model tries to capture the service distribution in a reference domain, the performance model tracks the service provider's past performance that can be used to predict/estimate an agent's performance with respect to future requests. It will not only try to capture the overall

performance rating of an agent, but will also try to reveal the agent's strong and weak areas within its advertised service space.

At the core of the performance model is a set of rating entries. For each advertised service, there is a set of rating entries that are constructed and updated with the evaluation information such as service feedback from the consumer agents and/or settings specified by a human administrator. Figure 5-3 shows two rating entries for the advertised service "Computer manufacture service".

At the minimum, a rating entry consists of an entry profile and a rating value. The entry profile defines the scope of the entry, usually a sub-set of the advertised service. The rating value $r \in [0, 1.0]$ represents the cumulative satisfaction rating on the agent with respect to the services described by the entry's profile. For the example in Figure 5-3, the rating entry-1 has a profile "Laptop manufacture service" with a rating value of 0.3, while the rating entry-2 has a profile "Desktop manufacture service" with a rating value of 0.8. What these rating entries may tell us is that although the service provider advertised "Computer manufacture services", its cumulative satisfaction rating on the Laptop manufacture service is very bad, but it is doing very well in the area of Desktop manufacture services. A rating of 0.0 is the worst (the agent could not perform the tasks at all) and a rating of 1.0 is the best (perfect).

The simplest way to establish rating entries would be to pre-specify the set of rating entries and then the only changes/updates needed would be to update the rating values. A more general approach is to dynamically establish and manage the set of rating entries as service feedback or other evaluation information is received.

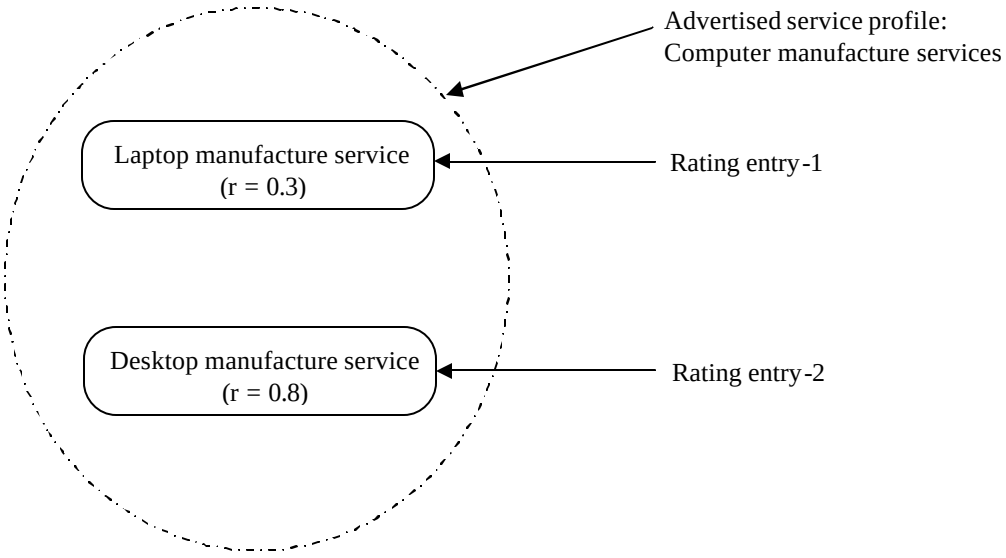


Figure 5-3. A simple example of rating entries

The performance model is constructed, updated, and refined based on the evaluation information, the domain ontology, and the distribution of services in the reference

domain. The evaluation information can be anything that may provide some hint on the performance of a provider agent, for example, an evaluation rating by some human administrator, service feedback from the consumer agents, what the matchmaker learns in the interaction with the agents, etc. This work mainly focuses on the evaluation information that is in the form of consumer agent feedback.

A feedback (in this context) is an opinion a service consumer agent has about a specific service provided by a specific agent(s), for example, in the form of satisfaction rating. It can certainly have more information than an overall satisfaction rating, but to simplify the discussion, we assume the evaluation will be an overall satisfaction rating in the range of [0, 1.0], where 0.0 means totally unsatisfied and 1.0 means perfect.

When a feedback is received, the matchmaker agent checks to see if an entry already exists with the same profile as that of the feedback, and if so, the entry is updated with the feedback. Otherwise, a new rating entry may be created. When the number of entries exceeds a certain limit, which is called the pruning limit, the entry set is consolidated and shrinks to a predefined size, called the base size. The purpose of having a base size and a pruning limit is to avoid frequent consolidation.

A rating entry represents the cumulative satisfaction rating of the service provider agent with respect to the entry's profile. When a feedback is applied to an existing entry, the rating will need to be updated. There can be different ways to update the rating, but EWMA (Exponentially Weighted Moving Average) should work well for most of the cases. EWMA is widely used in process control as well as in forecasting. The definition for EWMA is as follow (from [27]):

$$\tilde{x}_t = (1 - \alpha) x_t + \alpha \tilde{x}_{t-1} \quad \text{for } t = 1, 2, \ldots \qquad \text{(EQ 5-2)}$$

Where

- $\tilde{x}_t$ is the new forecast or expected value of x;
- x_t is the current observation value (in our case, the new feedback rating)
- $\tilde{x}_{t-1}$ is the previous forecast or expected value;
- $0 \leq \alpha < 1$ is a constant that determines the depth of memory of the EWMA. The smaller the α, the more weight is given to the most recent values.

The advantage of using EMWA for rating update is its adaptive nature, that is, it can capture the trend of the dynamic changes. For example, if an agent's performance is getting better or worse, that trend may be captured because more recent data are given larger weights in EWMA. The computation is also very simple since the new overall rating may be calculated from the previous overall rating and the current feedback rating.

In some cases, it might be desirable to use a time series analysis model to capture the internal structure of the rating data, for example, seasonal variations. Readers interested in further details on time series analysis are referred to [52].

Entry management deals with issues like entry creation, update and consolidation, but the focus is on the consolidation, that is, how to combine and/or delete entries when the total number of entries exceeds the pruning limit.

Different entry management strategies may be used depending on the needs of specific application domains. The strategy to be discussed here is similar to the LFRU (Least Frequently Recently Used) strategy used in storage (e.g. cache, RAM, disk) management because there are some similarities between the storage management and the rating entry management. For example, in cache management, you want the most frequently accessed (subset of the) data to stay in cache for fast access. In rating entry management, you want to capture as much detailed and accurate information as possible on the (subset of the) service that are most frequently requested.

For the purpose of entry management, we differentiate three categories of rating entries: the main entry that has exactly the same profile as the advertised service profile, the specialized entries whose profiles are specializations of the advertised service profile, and the generalized entries whose profiles are generalizations of the advertised service profile.

The LFRU-like algorithm for consolidating rating entries when the number of entries exceeds the pruning limit can be illustrated as follow:

```
While(entrySize > baseSize)
{
    Find the LFRU-entry;
    Remove the LFRU-entry.
    entrySize = entrySize – 1;
}
```

Over the time, more entries will be dedicated to areas in which services are more frequently requested.

More refined strategies may be employed for specific application domains. A strategy is usually developed based on a set of assumptions which may or may not be universally true. For example, the LFRU-family algorithms are mainly based on the assumption that if something is recently used (e.g., a memory block), it is likely that it will be used again (in the near future). This is called the principle of temporal bcality. We may use the storage management as an example to illustrate the idea that certain assumptions may not be universally true. Suppose there are two levels of cache. The principle of temporal locality holds well for the level-1 cache. For example, if you compute x = y + z, it is very likely that you will use “x” shortly. Therefore, it makes sense to keep/put “x” in the cache. For the level-2 cache, however, the same principle may not hold (as) well (or in the same way). With the same x = y + z example, since the “x” is most likely in the level-1 cache, it is unlikely that it will be accessed from the level-2 cache again in the “short term”.

In our case, we think the principle of temporal locality should generally hold, but for a specific application domain, it may require the analysis of the real data to determine whether and to what extent it holds, and then develop strategies accordingly.

Other areas of consideration may include what to do with the removed entries, what are the other factors (other than LFRU) that may affect the selection of “victim” entries (that is, the entries to be eliminated). For example, we may consider removing

generalized entries before removing the specialized entries, because the generalized entries cover more service space than what has been advertised by the provider and therefore tend to be less relevant. We may also choose to "combine" a victim entry with some other entries, e.g., its least general parent entry, instead of just removing it.

The performance model can be used to predicate/estimate the capability of a provider agent with respect to any specific given description, which may or may not be identical to the description advertised. More details will be described in the next chapter.

The advantage of modeling the agent's capability this way is that we can not only tell the overall performance of an agent (with respect to the advertised service description), but will also be able to estimate the agent's capability with respect to specific service descriptions as requested. This is important since we can take advantage of an agent's strong areas and avoid its weak areas. This model can also be extended to support capability modeling based on domain specific "features" such as "amenities" for hotel services.

5.3 OWL-S Extension

OWL-S does not provide any mechanism for describing service feedback. In order to support what is being discussed in this work, we extend OWL-S with constructs that would support feedback descriptions.

The top-level construct "ServiceFeedbck" is defined as follow:

```
<owl:Class rdf:ID="ServiceFeedback">
    <rdfs:subClassOf rdf:resource="&profile;#Profile"/>
</owl:Class>
```

The class ServiceFeedback is defined as a sub-class of the OWL-S Profile class. A feedback profile is the description of exactly what kind of service this feedback is about. This description may or may not be exactly the same as what was previously advertised and/or requested. For example, the recommendation request could be about travel service for Asia but the feedback may be actually about travel services for China.

The ServiceFeedback class has a "feedbackAbout" property that specifies which advertised service this feedback is about. With this property, the matchmaker agent would be able to obtain such information as which provider provided the service, and what was the advertised service description, etc. The definition is as follow:

```
<owl:ObjectProperty rdf:ID="feedbackAbout">
    <rdfs:domain rdf:resource="#ServiceFeedback"/>
    <rdfs:range rdf:resource="&service;#Service"/>
</owl:ObjectProperty>
```

A new property feedbackBy is defined to allow the specification of who provided this feedback. This may not be necessary if the sender information is complete in the agent communication messages. The definition is:

```
<owl:ObjectProperty rdf:ID="feedbackBy">
    <rdfs:domain rdf:resource="#ServiceFeedback"/>
</owl:ObjectProperty>
```

The range of the feedbackBy property is not defined here because this information may be platform dependent.

The new property overallRating represents the overall rating the consumer agent gives the service as specified by the feedbackAbout property, with respect to the description as specified by the ServiceFeedback profile. This property is defined as follow:

```
<owl:DatatypeProperty rdf:ID="overallRating">
    <rdfs:domain rdf:resource="#ServiceFeedback"/>
    <rdfs:range rdf:resource="&xsd;#float"/>
</owl:DatatypeProperty>
```

Please note that if this work is extended to support feature-based capability modeling, new properties may need to be defined to support the ratings on features, instead of (or in addition to) this overall rating.

5.4 A Note on Interaction Protocols

In the last section, we discussed an extension to OWL-S to support the description of service feedback. It might be sufficient for some agent platform, but for the others, some kind of support may be needed at the communication protocol and/or interaction protocol level. For example, in the speech act based agent platform such as KQML and FIPA, defining additional performatives may be appropriate.

We will define a few KQLM style performatives for this purpose but it is purely conceptual in the sense that the issue may be addressed in different ways and defining new performatives may or may not be the best way to do it. We assume that the agent platform already has performatives such as (or equivalent to) “advertise”, “recommend”, “inform”, etc.

The set of new performatives are described as follow:

(1) <follow-on-recommend> The matchmaker agent, if it chooses to, may notify an agent about the availability of new (better) service providers for a service that was previously requested.

(2) <feedback> An agent can voluntarily, or in response to a request, send <feedback> message(s) to the matchmaker agent to indicate how well the previously recommended service provider performed.

(3) <request-for-feedback> The matchmaker agent may ask an agent (to whom recommendations of service providers were made) how well the previously recommended providers performed. The agents asked are encouraged to give a timely response.

(4) <capability-query> The matchmaker agent may ask a service provider agent about its capabilities with respect to services of a given description.

Again, what described here is purely conceptual and just to provide a framework to discuss this work. The exact way of interaction will depend on the specific agent platforms. If an application agent (a service provider agent or a service consumer agent) does not understand the feedback related constructs, the agent should still be able to interact with the matchmaker agents on the core tasks, such as advertising a service and/or requesting a recommendation. However, it may not be able to take the full advantage of the matchmaker's capability, and may not be able to contribute to the success of the matchmaker agent and therefore, to the success of the overall agent system. If an agent does not comply with the feedback-related interaction protocol, it should not directly affect any other agents.

For the sake of completeness, when an advertisement is un-advertised, everything related to that advertisement can be trivially thrown away – mainly the rating entries.

There are at least two more questions that have to be answered. The first question is why a consumer agent would want to provide feedback. I think the first and foremost reason is to be cooperative, just as many (human) consumers would respond to various surveys (without any compensation). These surveys may usually bring changes that are favorable to the consumers. In our case, an informed matchmaker would certainly be beneficial to the consumer agents. The matchmaker agent can also take some measures to reward those agents that are cooperative. For example, for those who would provide feedback on the service providers recommended, the matchmaker may pick candidates using methods that take advantage of the agent's capability model; and for those who do not, the matchmaker may use the service description matching to find candidates without taking advantage of the agent capability model. As for the issue of the truthfulness of the feedback, that is, the trust issue, we consider that as an orthogonal issue that can be addressed separately.

The second question is why a service provider agent would work with a matchmaker agent that may learn about their capabilities. I think a good matchmaker would attract the service consumer agents, which in turn would attract the service provider agents. A similar example in the real life is the Better Business Bureau (BBB). Many businesses participate in the BBB mainly because being a member of BBB would win the trust of more customers, even though such participation might mean that they may have to subject themselves to certain rules. Just as some businesses may not choose to participate in BBB, some agents may not choose to work with such a matchmaker.

5.5 Summary

This chapter described a way of modeling an agent's capabilities that is based on two main factors: an agent's performance history and the service distribution within the reference domain.

In this model, the matchmaker agent dynamically captures the service distribution and an agent's performance model is established and refined whenever some evaluation information (e.g., feedback) is received. The central component of the performance model is a set of rating entries, where each entry represents the cumulative satisfaction rating on a provider agent in a specific service area. To support the additional agent interaction introduced in this work, e.g., service feedback, we extended OWL-S with constructs for describing service feedback.

There may be different ways to make use of such an agent capability model. In this work, it is the basis for computing an agent's (estimated) capability with respect to a given request. The approach discussed in this work uses a weighted average based method in which the weight is computed based the distance between a rating entry and the request, in other words, the weight is based on the relevancy or the degree of match. That is the subject of the next chapter.

6 Put it Together - Quantitative Service Matching

As discussed in Chapter 4, there can be four levels of service description match, namely exact match, plug-in match, subsumes match, and intersection match. The service match levels are defined based on how the advertised service space S_{adv} overlaps with the requested service space S_{req}. Briefly it is an exact match if $S_{req} = S_{adv}$; plug-in match if $S_{req} \subset S_{adv}$; subsumes match if $S_{req} \supset S_{adv}$; otherwise, intersection match if $S_{req} \cap S_{adv} != \phi$.

These match levels provide a metric to determine which matches are better than the others. Needless to say, exact matches are the best. Plug-in matches are "sufficient matches" in the senses that based on the semantics of the service descriptions the advertised service meets the requirements set forth in the request. Subsumes matches and intersection matches are "potential matches" because based on the semantics of the service descriptions, the advertised service only partially meets the requirements set forth in the request and may potentially perform the requested services.

While the match levels may tell us which matches are better based on the semantics of the service descriptions, there is no hint on which matches are better if the two matches are of the same match level, for example, when both are exact matches. This is because the match levels are determined only based on the service descriptions and some other important factors, such as the performance of the service providers, the service distribution, etc, are not taken into account.

It is easier to imagine why performance can be a factor. For example, two service providers may advertise services of similar or even identical descriptions but their actual performance levels may be quite different, and the requester would certainly prefer the provider(s) with better performance.

Service distribution can be an important factor, too, especially in the case of potential matches (e.g., subsumes matches or intersection matches). As in the example of solar powered car (mentioned in Chapter 1), both "SolarPoweredCar" and "GasPoweredCar" would match "Car" at subsumes level, but we know that in general, choosing "GasPoweredCar" (over SolarPoweredCar) would be correct in some 99% of the time, since GasPoweredCar accounts for 99% of the market. That is to say, even though both "SolarPoweredCar" and "GasPoweredCar" would match "Car" at subsumes level, the actual "degree" of match differs significantly, and in this case, it is 1% vs. 99%.

In this work, factors such as service distribution and agent performance will be taken into account and service matching is accomplished in two phases. The first phase performs semantic service description matching to find all the candidates, that is, all the advertised services whose service descriptions semantically match that of the requested service description. The second phase attempts to quantitatively evaluate the candidates and single out the candidates that have higher probabilities of success. The basis of such evaluations is the agent capability model discussed in Chapter 5.

6.1 The Basics

In Chapter 1, we discussed how the matchmaker agent may build and refine the capability model for an agent based on the agent's past performance and the service distribution. In the rest of this chapter, we will focus on how to take advantage of the agent capability model in finding the best matches.

The agent capability model has two main components, the performance model and the service distribution model. The service distribution model tracks the distribution of services, for example, what percentage of computer purchase requests are about laptops. The core of the performance model is a set of rating entries, each representing the cumulative satisfaction rating on an advertised service with respect to the entry profile. For example, for a computer manufacture service, a rating entry may have a profile of laptop manufacture service with a rating value 0.8.

For the clarity of the discussion, we will need to introduce a few terms related to performance rating.

Definition 6-1

> **Capability rating** is the rating on the intrinsic capability that a service provider agent has to perform tasks of a specific description. A capability rating r_c is in the range [0, 1.0], with 0 as the worst and 1.0 as the best. An agent will likely have different capability ratings on performing tasks of different descriptions. We may never know the true "capability rating" of an agent and it is what the matchmaker is trying to capture, or more accurately, to approximate. See also Definition 6-3.

Definition 6-2

> In this context, **satisfaction rating** refers to an individual consumer agent's degree of satisfaction towards specific execution(s) of a service. A satisfaction rating r_s is also in the range of [0, 1.0], with 0 as totally unsatisfied and 1.0 as 100% satisfied.

Definition 6-3

> In this context, the term **performance rating** is used to refer to the matchmaker's overall estimation of a service provider agent's capability rating (see Definition 6-1) with respect to performing tasks of a specific description. In other words, it is the matchmaker agent's overall confidence in an agent's capability of handling the request. Performance rating is computed by the matchmaker agent based on the request service description and the agent's capability model, which in turn is mainly established and refined based on consumer agents' satisfaction ratings (see Definition 6-2) on the service. This performance rating of a provider agent is the overall score that the matchmaker agent will use to rank the candidate matches.

It is important to point out that when talking about capability rating, satisfaction rating, or performance rating of an agent, it is always with respect to tasks of some specific description.

The core of an agent's performance model is a set of rating entries as discussed earlier in Section 5.2. Figure 6-1 shows the example rating entries for an advertised computer manufacture service. There are three rating entries: entry-1 with a profile of "Laptop manufacture" and a rating value of 0.8; entry-2 with a profile of "Desktop manufacture" and a rating value of 0.9; and entry-3 with a profile of "Server manufacture" and a rating value of 0.5. Based on such a model, the performance of the agent with respect to a specific request may be estimated. For example, if the request is about Personal computer manufacture, then we can estimate that the performance rating of the agent with respect to Personal computer manufacture would be somewhere between 0.8 and 0.9 (suppose personal computer includes only desktops and laptops).

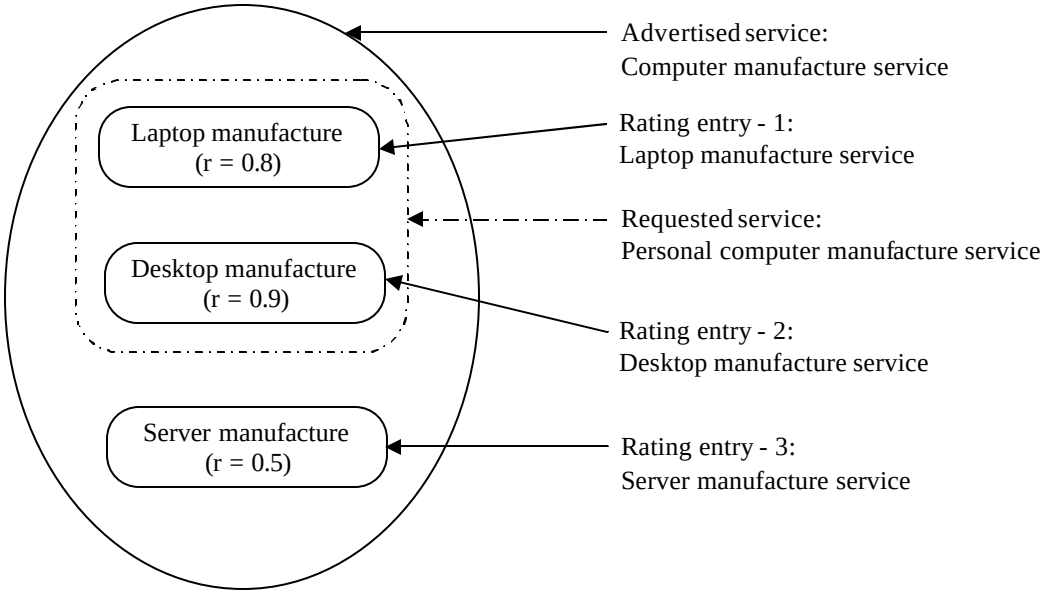


Figure 6-1. Service spaces of the request and the rating entries

6.2 The Desired Properties for a Quantitative Measure

As we have seen in the previous section, the agent capability model described in Chapter 1 may be used to estimate an agent's performance with respect to any given request. What we need is some method (or measure) that can use the agent capability model to estimate the performance of an agent with respect to a given request. We start by systematically analyzing what properties such a method should possess, and then a method will be proposed, with a proof that the properties hold.

To assist the discussion, we first define a few symbols:

- n is the total number of rating entries of a agent's performance model.
- $r_i \in [0, 1.0]$ is the rating value of the i^{th} rating entry;

- $p_i \in [0, 1.0]$ is the relevancy (to be defined later) of the i^{th} rating entry with respect to the request;
- R_n is the overall performance rating of the agent with respect to the request, with all the n entries considered. That is, R_n is the final estimation we want to compute.

Now, we conduct a systematic analysis of what properties a good method should possess.

There are totally four types of elements or factors here:

n, R_n, r_i, and p_i.

We first look at "n", the total number of rating entries. It is not directly relevant. We know that n>=1 since there is at least a default rating entry (initially).

Next we look at the final outcome R_n. First of all, it is evident that it must fall in [0, 1.0]. Moreover, it must be less than or equal to the largest entry rating, since with the largest entry rating in place, the entries with lower ratings should by no mean have the effect of increasing the overall rating. Therefore, we have property (1):

(1) $R_n \in [0, r_{max}]$, where $r_{max} = \max \{r_i\}$, $i = 1, 2, \ldots, n$.

Next, we study how r_i should influence R_n. With all other factors stay the same, a larger r_i should result in a larger R_n; a smaller r_i should result in a smaller R_n. This can be written as follow in property (2):

(2) a) $r_i >= r_i' \rightarrow R_n >= R_n'$; b) $r_i < r_i' \rightarrow R_n < R_n'$, where R_n' is the result of replacing the rating value r_i of the i^{th} with r_i'.

The way the relevancy p_i influences the result R_n is a bit more complex. The basic behavior should be that the larger the relevancy, the larger the influence. Therefore, with a smaller r_i (relative to the rating of the other entries), the larger the p_i, the larger the negative influence it has on R_n; with a larger r_i (relative to the rating of the other entries), the larger the p_i, the larger the positive influence it has on R_n. Now the issue is, where should the line be drawn (for r_i) between the negative influence and the positive influence of an entry?

One way to find it out is to first (hypothetically) remove the i^{th} rating entry and (hypothetically) compute the overall rating value R_{n-1}. Let R_n' be the result of (all n entries) but with p_i replaced by p_i'. Then when the i^{th} rating entry is added (back), if $r_i > R_{n-1}$, we would like to see $p_i > p_i' \rightarrow R_n > R_n'$; if $r_i < R_{n-1}$, we would like to see $p_i > p_i' \rightarrow R_n < R_n'$. In other words, the line (between positive and negative influence) is the overall rating value without the i^{th} rating entry. This can also be seen from a different perspective, that is, when a new entry is added, if the new entry rating is larger than the current overall rating, then the larger the relevancy of the new entry, the larger the new overall rating; if the new entry rating is smaller than the current overall rating, then the larger the relevancy of the new entry, the smaller the new overall rating. In summary, the desired influence of p_i may be formulated as follow in property (3):

(3) a) if $r_i >= R_{n-1}$, then $p_i > p_i' \rightarrow R_n >= R_n'$; b) if $r_i < R_{n-1}$, then $p_i > p_i' \rightarrow R_n < R_n'$, where R_{n-1} is the overall rating without considering the i^{th} rating entry; R_n' is the overall rating of all n entries but with p_i replaced by p_i'.

Now that we have the set of desired properties, in the next section, we will propose a method that may be used for estimating an agent's performance with respect to a given request and that has all these properties discussed above.

6.3 A Quantitative Measure for Service Matching

In this section, we will develop a method for estimating an agent's capability with respect to a given request based on the agent's capability model. For a given request, a rating entry may be abstracted as e(r, p), where r is the rating value of the entry and p is the relevancy of the entry with respect to the request. Now the problem becomes how to compute the rating value with respect to the request based on the set

$$\{e_i(r_i, p_i) \mid i = 1, 2, \ldots n\}.$$

There may certainly have different ways to abstract the problem and/or to make use of the agent capability model, but we think a weighted average based method is a natural choice because it allows an entry that is more relevant to have more "influence" in the final outcome. The formula is shown below as in EQ 6-1 below:

$$R_n = \sum_{i=1}^{n} (r_i * p_i) \Big/ \sum_{j=1}^{n} p_j \qquad \text{(EQ 6-1)}$$

Where r_i is the rating value of the i^{th} entry; p_i is the relevancy of the i^{th} entry; and R_n is the final, overall estimation of the agent's capability with respect to the given request. We will prove in the next section that such a method possesses all the desired properties listed in the previous section.

Now we look at how to compute the relevancy of a rating entry for a given request. A natural choice is the degree a rating entry (profile) matches the request. Since each service profile defines a service space, the degree that a rating entry matches the request is therefore the probability that an instance in the requested service space falls in the service space defined by the rating entry. Therefore, the relevancy of an entry to a request, rel(entry, req), can be computed as follow:

$$\text{rel(entry, req)} = P(s \in S_{entry} \mid s \in S_{req}), \text{ that is,}$$

$$\text{rel(entry, req)} = |(S_{entry} \cap S_{req})| / |S_{req}| \qquad \text{(EQ 6-2)}$$

Where S_{entry} is the service space defined by the rating entry profile; S_{req} is the requested service space. A pair of vertical bars gets the cardinality of the enclosed set. Note that the relevancy of two service profiles is not symmetric, that is, for two given service profiles A and B,

$$\text{rel(A, B)} \mathrel{!}= \text{rel(B, A)},$$

Because in general,

$$P(s \in S_a \mid s \in S_b) \mathrel{!}= P(s \in S_b \mid s \in S_a),$$

where S_a and S_b are the service space defined by A and B, respectively. To illustrate with a simple example, suppose A is a computer manufacture service, B is a desktop

manufacture service. A service provider with the capability A (by description) should be able to perform service B, but not the other way around.

Equation (EQ 6-2) may be computed based on the service distribution. There is usually no way to know the exact service distribution in the global world (the world of agents), but with the distribution model discussed in Chapter 5, the matchmaker agent can dynamically capture the service distribution in the scope of its reach. This local distribution may actually be more relevant than the global distribution for this purpose of computing the degree of match.

Therefore, if an entry matches the request as an exact match, that is, $S_{entry} = S_{req}$, then rel(entry, req) = 1. In other words, the relevancy (or degree of match) would be 100%.

If the match is a subsumes match, that is, $S_{entry} \subset S_{req}$, then

$$\begin{aligned} rel(entry, req) &= | (S_{entry} \cap S_{req}) | / | S_{req} | \\ &= | S_{entry} | / | S_{req} | \end{aligned}$$

Then the degree of match will be the degree of subsumption, that is, what percentage does the entry service space accounts for in the requested service space.

For intersection match, the degree of match is the percentage the common subset accounts for in the requested service space, that is, $| (S_{entry} \cap S_{req}) | / | S_{req} |$.

The case of plug-in match is similar to exact match. For a plug-in match, we have $S_{entry} \supset S_{req}$. Therefore,

$$\begin{aligned} rel(entry, req) &= | (S_{entry} \cap S_{req}) | / | S_{req} | \\ &= | S_{req} | / | S_{req} | \\ &= 1 \end{aligned}$$

That is, the degree of match (or the relevancy) is 100%.

In the next section, we will give the proof that the method proposed in this section has all the desirable properties discussed in Section 6.2; then give a brief analysis on the computational complexity in 6.5; and finally summarize the chapter in Section 6.6.

6.4 Proof of Conformance

In this section we give proof that the weighted average method proposed in Section 6.3 has all the desired properties listed in Section 6.2.

Theorem 6.1

The formula (EQ 6-1) has the properties (1) – (3) as described in Section 6.2.

Proof:

Let us prove the properties one at a time.

1. Property (1):

$R_n \in [0, r_{max}]$, where $r_{max} = \max \{r_i\}$, i = 1, 2, …, n.

Proof for property (1):

Since $r_i >= 0$ and $p_i >= 0$, it trivially follows that $R_n >= 0$.

$$R_n = \sum_{i=1}^{n} (r_i * p_i) / \sum_{j=1}^{n} p_j$$

$$<= \sum_{i=1}^{n} (r_{max} * p_i) / \sum_{j=1}^{n} p_j$$

$$<= r_{max} \sum_{i=1}^{n} (p_i / \sum_{j=1}^{n} p_j)$$

$$<= r_{max}$$

Therefore, we have $R_n \in [0, r_{max}]$.

End of proof for property (1).

2. Property (2):

   a) $r_i >= r_i' \rightarrow R_n >= R_n'$;

   b) $r_i < r_i' \rightarrow R_n < R_n'$,

   where R_n' is the result of replacing the rating value r_i of the i^{th} with r_i'.

Proof of property (2)

Let $P_n = \sum_{j=1}^{n} p_j$, $T_n = \sum_{i=1}^{n} (r_i * p_i)$,

Since $R_n = \sum_{i=1}^{n} (r_i * p_i) / \sum_{j=1}^{n} p_j$,

$R_n' = (T_n - r_i * p_i + r_i' * p_i) / P_n$

$R_n - R_n' = T_n / P_n - (T_n - r_i * p_i + r_i' * p_i) / P_n$

$= T_n / P_n - (T_n - (r_i - r_i') * p_i) / P_n$

$= T_n / P_n - (T_n / P_n - (r_i - r_i') * p_i / P_n)$

$= T_n / P_n - T_n / P_n + (r_i - r_i') * p_i / P_n$

$= (r_i - r_i') * p_i / P_n$ EQ 6-3

Therefore, $r_i >= r_i' \rightarrow R_n >= R_n'$; $r_i < r_i' \rightarrow R_n < R_n'$

End of proof for property (2).

3. Property (3):

   a) if $r_i >= R_{n-1}$, then $p_i > p_i' \rightarrow R_n >= R_n'$;

   b) if $r_i < R_{n-1}$, then $p_i > p_i' \rightarrow R_n < R_n'$,

   where R_{n-1} is the overall rating without considering the i^{th} rating entry; R_n' is the overall rating of all n entries but with p_i replaced by p_i'.

Proof of property (3)

Let $P_n = \sum_{j=1}^{n} p_j$, $T_n = \sum_{i=1}^{n} (r_i * p_i)$,

Since $R_n = \sum_{i=1}^{n} (r_i * p_i) / \sum_{j=1}^{n} p_j$,

$R_n' = (T_n - r_i * p_i + r_i * p_i') / (P_n - p_i + p_i')$

$R_{n-1} = (T_n - r_i * p_i) / (P_n - p_i)$

$R_n - R_n' = T_n / P_n - (T_n - r_i * p_i + r_i * p_i') / (P_n - p_i + p_i')$

$$= \frac{T_n(P_n - p_i + p_i') - (T_n - r_i * p_i + r_i * p_i')P_n}{P_n(P_n - p_i + p_i')}$$

$$= \frac{-T_n * p_i + T_n * p_i' + P_n * r_i * p_i - P_n * r_i * p_i'}{P_n(P_n - p_i + p_i')}$$

$$= \frac{P_n * r_i(p_i - p_i') - T_n(p_i - p_i')}{P_n(P_n - p_i + p_i')}$$

$$= \frac{P_n * r_i - T_n}{P_n(P_n - p_i + p_i')}(p_i - p_i') \quad \text{EQ 6-4}$$

If $r_i >= R_{n-1}$, that is,

$r_i >= (T_n - r_i * p_i) / (P_n - p_i)$, therefore,

$T_n - r_i * p_i <= (P_n - p_i) * r_i$, therefore,

$T_n - r_i * p_i <= P_n * r_i - p_i * r_i$, therefore,

$P_n * r_i >= T_n$ EQ 6-5

Therefore, if $p_i > p_i'$, combine EQ 6-5 into EQ 6-4, we have

$R_n - R_n' = \frac{P_n * r_i - T_n}{P_n(P_n - p_i + p_i')}(p_i - p_i') >= 0$, that is, $R_n > R_n'$.

If $r_i < R_{n-1}$, that is,

$r_i < (T_n - r_i * p_i) / (P_n - p_i)$, therefore,

$T_n - r_i * p_i > (P_n - p_i) * r_i$, therefore,

$T_n - r_i * p_i > P_n * r_i - p_i * r_i$, therefore,

$P_n * r_i < T_n$ EQ 6-6

Therefore, if $p_i > p_i'$, combine EQ 6-6 into EQ 6-4, we have

$R_n - R_n' = \frac{P_n * r_i - T_n}{P_n(P_n - p_i + p_i')}(p_i - p_i') < 0$, that is, $R_n < R_n'$.

End of proof for property (3).

End of proof (for theorem 6.1)

Again, this variant of weighted average method is certainly not the only way to compute the overall performance rating using the agent capability model. Different strategies may be employed depending on the specific application domain.

6.5 The Complexity

In this section, we briefly discuss the complexities of the various service matching operations. The complexities of the basic operations in Description Logic, such as subsumption, have been well studied, for example, in [4], therefore, we will focus on the complexity on top of these basic operations, in other words, we will use these basic operations as the basic units.

Definition 6-4

> For the purpose of discussing the computational complexity of this work, the concept subsumption operation is considered a special unit operation called **CSS**, which stands for "Class Subsumption"; the operation of checking the satisfiability of concept intersection is considered as a special unit operation, called **INT**.

The handling of incoming advertisement is relatively simple. It mainly involves two operations, the registration of the advertisement and the initialization of a performance model. The time complexity is therefore O(C.

The main source of complexity for handling a recommendation request is service matching. The first step in service matching is to find the list of candidates, that is, the advertised services that match the given request. If the parameters can be paired by their unique names, the time complexity for this step is O(N*P*CCS), where N is the total number of advertised services in the reference domain, and P is the number of parameters in a pair of profiles. If the bipartite matching process is needed in the parameter pairing, then the time complexity becomes $O(N*P^3*CCS)$. The P^3 component comes from the maximum weighted, maximum cardinality bipartite matching [47].

The second step in service matching is about the estimation of an agent's performance with respect to the given request. The complexity for this step is O(M*K*P*CCS) without the need for bipartite matching in parameter pairing, and $O(M*K*P^3*CCS)$ if bipartite matching is needed in the parameter pairing, where M is the number of candidates and K is the number of rating entries. Since in the worst case, the total number of candidates can be as many as the total number of advertised services, therefore, the overall complexity of service matching (two steps) is O(N*K*P*CCS) without bipartite matching, and $O(N*K*P^3*CCS)$, with bipartite matching.

The main source of complexity for handling service feedback is to find out the rating entry to which the feedback should be applied, or to make sure such an entry does not exist yet. It uses the same procedures as semantic service matching, therefore, it is O(K*P*CCS) or $O(K*P^3*CCS)$, if bipartite matching is needed in parameter pairing, where K is the number of rating entries and P is the number parameters in each pair of service profiles. An LFRU-based (Least Frequently Recently Used) rating entry set management algorithm has a complexity between O(C) and O(logK) [73], where K is the number of rating entries.

6.6 Summary

The fundamental idea behind this work is that service matching should be more than the matching of service descriptions. Many other factors, such as the service distribution, the performance of the service providers, may tell us further information about which provider is a better fit for a given request and should therefore be taken into consideration.

To summarize, the service matching is a two-phase process. The first phase mainly deals with the matching of service descriptions, that is, performs semantic matching to check if the description of an advertised service logically matches that of the request. In the second phase, the matchmaker agent goes beyond the service descriptions and quantitatively evaluates the candidate matches selected in the first phase. The basis for such evaluation is the agent capability model that the matchmaker has established and has continuously refined over the course of the interactions with the agents. The result of such evaluation is the matchmaker agent's overall confidence that a provider is up to the tasks requested, and this will be the basis for ranking (and therefore, for selecting) the providers to recommend to the requesters.

7 The Experimental Implementation and Results

To evaluate the effectiveness of the ideas and algorithms discussed in this dissertation, a prototype matchmaker and an evaluation framework have been designed and implemented under the OWL/OWL-S framework. The example problem domain is a simple flight ticket reservation domain as discussed in Chapter 1. In this chapter, we will first describe the design and the implementation of the prototype and the simulation framework, and then the result statistics will be presented and analyzed.

7.1 The Design and Implementation

In this prototype implementation, OWL is chosen as the representation language for ontology and service descriptions and OWL-S is used as the service ontology. To support service feedback, OWL-S is extended with constructs that would allow the description of service feedback. The details on the extension have been discussed in Chapter 1.

The overall design is illustrated in Figure 7-1. Daml4jess is a Jess-based DAML inference engine, slightly modified to support basic OWL inference. In this prototype, it is used to manage the knowledge base of ontologies and service descriptions; it is also where OWL inferences take place for tasks like semantic matching. The "Match Rules" module consists of a set of rules to be used for service matching, that is, to decide if an advertised service description logically matches that of the request. The Java modules in the diagram complement the daml4jess in areas where it is not quite efficient or "convenient" for Jess.

You may have noticed that the "Profile Management" box and the "Agent Messaging" box are in gray. The "Profile Management" is in gray because it is basically the same service profile information as that in the knowledge base and is duplicated for performance considerations. The "Agent Messaging" box is also in gray because there is no real agent communication system in place, at least not for now. The agent messaging is simulated in the simulation engine.

7.1.1 Daml4jess

In looking for an OWL (well, DAML, at the time) inference engine, DAMLJessKB came into our radar screen. On the back end, it uses Jess as the underlying inference engine and a set of Jess rules/axioms is defined to implement the DAML semantics; on the front end, Jena ARP RDF parser is used to parse the RDF/DAML document and then the DAML statements are transformed into Jess facts.

In general, DAMLJessKB is a very good tool, especially the intimate interaction among Jess, DAML, and Java language offers the flexibility you need in the prototype development. However, in our specific case, some much desired capabilities are either missing in DAMLJessKB or have not quite been designed in the way we need. As a result, we decided to rewrite DAMLJessKB and the new (rewritten) package is called daml4jess. The DAML semantic rules from DAMLJessKB are preserved as they are, except for insignificant format changes. The Java package, while following a similar

philosophy, is virtually a complete rewrite. It was then slightly modified to support basic OWL inference.

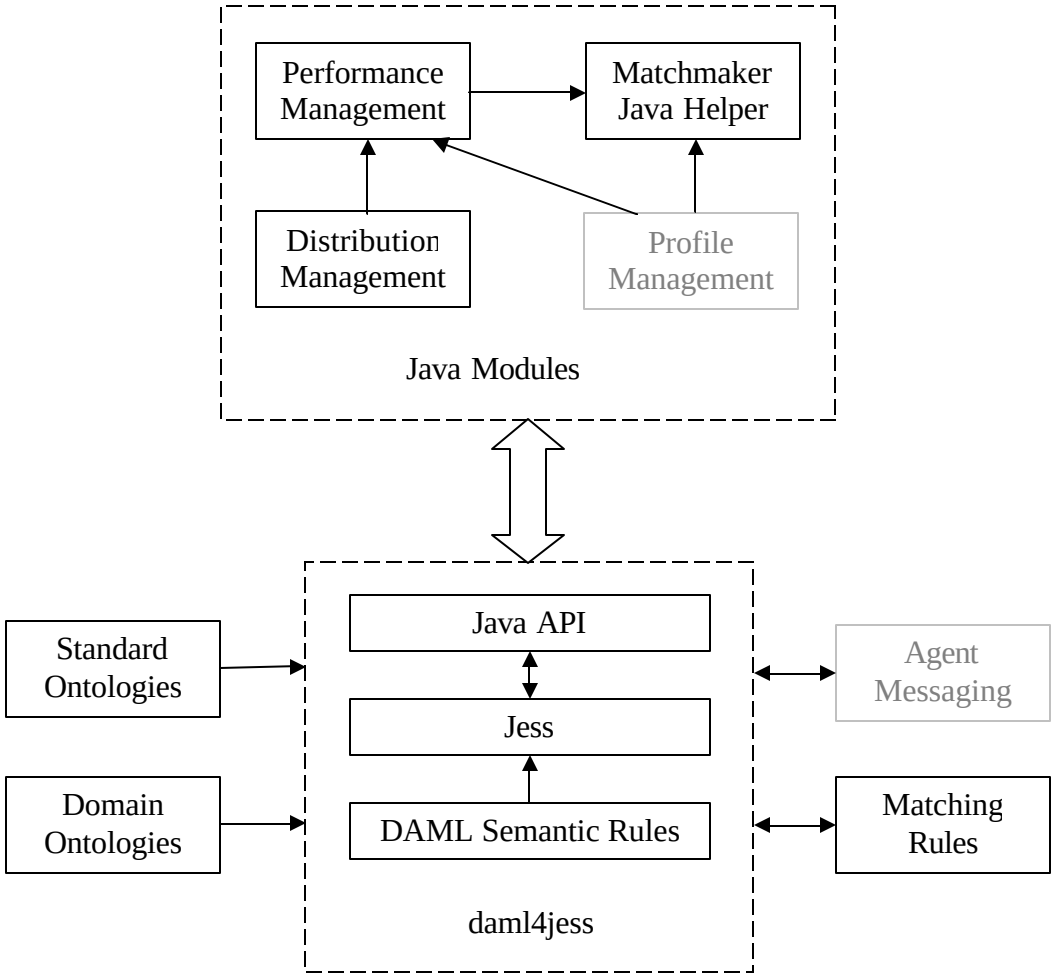


Figure 7-1 Matchmaker agent design overview

7.1.2 The Java modules

The performance management module is responsible for managing the service provider agents' capability model. An agents' capability model is updated as evaluation information (such as feedback) is received. This module is also responsible for estimating an agent's capability with respect to a given request, which may or may not be (and is usually not) identical to the service descriptions advertised by the agents.

The matchmaker helper module takes service matching beyond the semantic service description matching. It performs operations that are not quite "convenient" for the Jess

inference engine and most importantly, it calls on the performance management module to perform the quantitative service matching, that is, to estimate an agent's performance with respect to the given request based on the agent capability model. The distribution management module maintains and updates the service distribution information.

7.1.3 Put it all together

When an advertisement message is received, daml4jess is called on to extract the service description information and have it registered with the system. When a service matching request is received, the daml4jess is again called on to extract the requested service information, register the information, and then initiate the service matching process. The first phase of the matching will be conducted in the daml4jess/Jess inference engine. This phase performs semantic matching, that is, to check if an advertised service matches the request. Those that pass the semantic match check are considered candidates and will proceed to the next phase for further evaluation.

In the second phase, the Java side matchmaker will evaluate each candidate to find the best matches. In this process, the performance management module will be leveraged to estimate the performance of a candidate with respect to the request based on the agent capability model. Finally, the winners will be selected and recommended to the requester. The service distribution will then be updated. Note that we used some random factor in selecting the final winner to avoid the situation in which one good provider gets all the relevant requests and to give the new comers a chance to excel.

Some service consumer agents may choose to provide feedback on the service performed by the recommended service providers, while some others may provide feedback information upon the request of the matchmaker agent. When a feedback is received by the matchmaker agent, the daml4jess is called on to extract the information and then forwards it to the performance management module where the capability model for the service provider(s) involved will be refined and updated.

7.2 An Evaluation Framework

An evaluation framework has also been designed and implemented to simulate the agent interactions. We decided not to deploy a "full-fledged" multi-agent system because we believe that a simulator should be sufficient for the purpose of validating ideas and it would save us some efforts.

The evaluation framework is illustrated in Figure 7-2. On the right is the matchmaker agent block, which is the experimental implementation discussed in the last section. On the left is the simulation engine. It takes multiple roles - all roles needed in a multi-agent system except for the matchmaker agent. It generates service advertisements and plays the role as service providers; it generates service requests and plays the role as service requesters (service consumers); when a service provider is recommended (by the matchmaker to the simulator) for a service request, the simulator acts as if the service has been performed and then a degree of satisfaction will be assessed, after which it may optionally generate a feedback to the matchmaker agent with respect to the service performed.

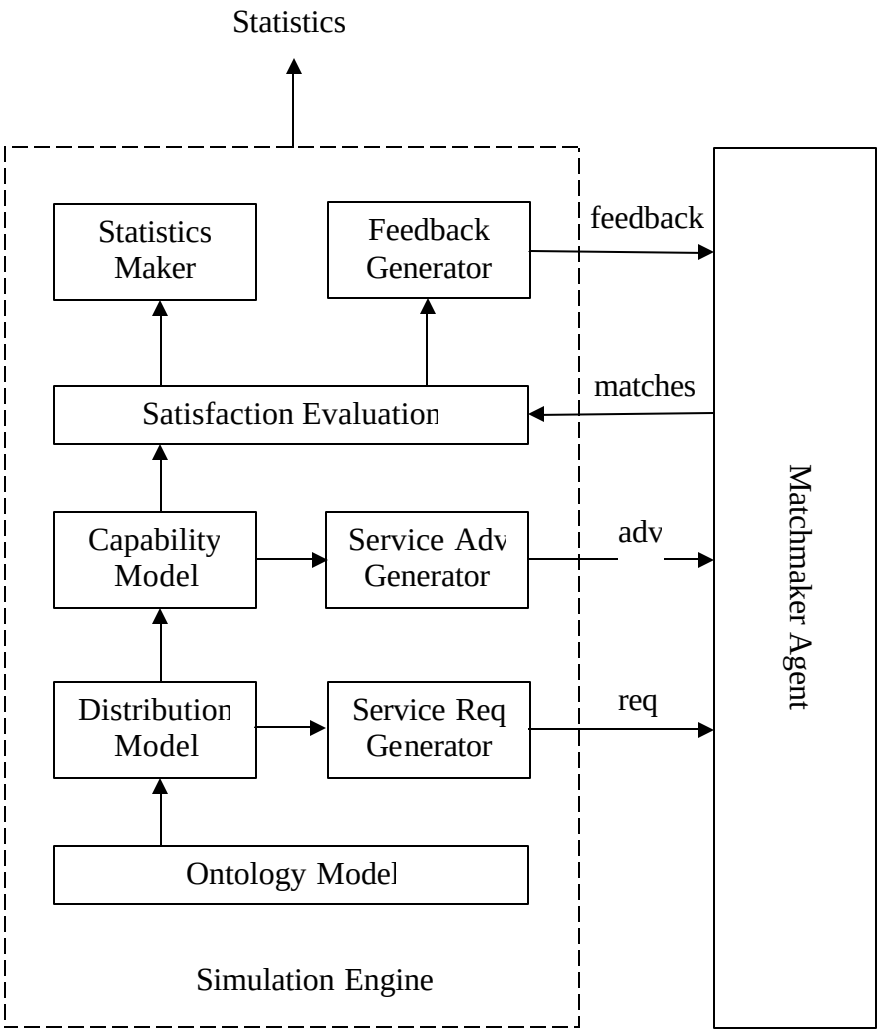


Figure 7-2 The evaluation framework

In this evaluation framework, the advertised service descriptions as well as the recommendation request descriptions are generated (with random factors) based on a given service profile template and a pre-generated service distribution model. The profile template specifies information such as the profile types, the base parameter types (the type that a parameter is restricted to for any generated service profile), etc. Here is an example profile template for international flight service:

hasInput: departureAirport

type = file:/home/grad3/xluan1/prototype/ontology/TravelOntology.owl#UsDomestic

hasInput: arrivalAirport

type = file:/home/grad3/xluan1/prototype/ontology/TravelOntology.owl#I18nApl

hasInput: payment

type = file:/home/grad3/xluan1/prototype/ontology/TravelOntology.owl#Cash

hasOutput: ticket

type = file:/home/grad3/xluan1/prototype/ontology/TravelOntology.owl#FlightTicket

In this template, the departure airport location is restricted to US domestic, the arrival airport location is restricted to international airports, the payment type is restricted to Cash payment, and the ticket type is restricted to flight ticket. Note that when we say a parameter is restricted to a certain type, it includes its subtypes, too.

The service distribution model governs the distribution of the services generated. For example, if the distribution model shows that only 15% of the time an international flight will depart from a US central region airport, then the probability that a generated service description has a US central region as the departure airport location will be around 15%.

In Chapter 1, we defined three kinds of ratings regarding a service, namely the capability rating, the satisfaction rating, and the performance rating. In this simulation framework, the capability ratings of a service provider agent are generated algorithmically as queried. The algorithms make sure that identical queries always yield the same capability rating value. The capability ratings are used as the basis for generating satisfaction ratings. Note that the capability ratings will never be known to the matchmaker agent.

The satisfaction rating a consumer agent gives toward a provider agent with respect to a specific service performed is generated according to Gaussian distribution, where the mean is the service provider agent's capability rating (with respect to the service performed) and the standard deviation is a fraction of the mean. The assumption here is that statistically the service consumer agents experience the "true" capability of the service provider agents.

The performance rating about a service provider agent with respect to a specific request is the overall evaluation of the agent by the matchmaker agent.

The performance of the matchmaker agent itself is evaluated by comparing it against three other categories of matchmakers, namely the type-1 basic matchmaker, type-2 advanced matchmaker, and type-4 ideal matchmaker. Our matchmaker is type-3 in the group. Details will be discussed in the later sections.

7.2.1 The distribution model

Service distribution model is a key component to this evaluation framework. It is used to simulate the service distribution that serves as the basis for generating service profiles. This service distribution model is a component of the evaluation framework and will NOT be available to the matchmaker agent. The matchmaker agent is supposed to capture the service distribution dynamically to build a distribution model of its own.

To simplify the simulation, we use the parameter type distribution model (details see Section 5.1.2), that is, to approximate service distribution with the parameter type distribution. As discussed in Chapter 1 and as illustrated in Figure 5-2, the distribution model is a directed acyclic graph (DAG) that is constructed based on the domain ontology graph. It includes all the type nodes specified in the profile template as well as

all their sub-type nodes. In the above flight ticket reservation example template, the nodes to be included in the distribution model will be the type nodes for US-Domestic, International, Flight-Ticket, Payment, and their descendant nodes.

To generate such a distribution model (for the purpose of simulation), we take a bottom up approach, because in a leaf node the total distribution value (occurrences) is the same as the direct distribution value and we can work upward to generate the distribution for their parent nodes. The algorithms can be illustrated as follow:

```
generateDistribtionMode()
{
    rootNodes = find-all-top-level-nodes;
    foreach <node> in rootNodes {
        bottomUpGeneration (node);
    }
}
```

Where the bottomUpGeneration()is a recursive process as described below:

```
bottomUpGeneration(currentNode)
{
    if (currentNode is processed) {
        return;
    }
    if(currentNode is a leaf node) {
        node.directDistribution = GaussianRandom (mean, sd); //predefined mean, sd
        node.totalDistribution = node.directDistribution;
        return;
    }
    childTotal = 0;
    foreach <child> of currentNode {
        bottomUpGeneration (child);
        childTotal = childTotal + child.totalDistribution;
    }
    currentNode.directDistribution = GaussianRandom(mean, sd);
    currentNode.totalDistribution = parent.directDistribution + childTotal;
    compute the distribution vector for the currentNode;
    return;
}
```

This bottom-up approach has at least two advantages. First, a leaf node's total occurrence count is the same as its direct occurrence count and is therefore a good point to start; second, the counts of lower level nodes naturally add up to get the counts for their parent nodes.

7.2.2 Generating advertised services

Advertised service descriptions are generated (with random factors) based on the given service profile template. For example, an international flight ticket reservation service profile template may restrict the departure airport location to US-Domestic, restrict arrival airport location to international, and restrict the ticket type to flight ticket. From this template, service descriptions like ticket reservation service for reserving tickets from US east coast to China may be generated.

When generating a service profile (to be advertised), each parameter type is randomly selected among the types that are compatible with what is specified in the template. For example, in the previous template, the arrival airport location parameter (international) can be China, South East Asia, etc, but it can not be anything of US-Domestic. This generated service description is called the base description of the advertised service. It is called "base description" in this context because later we may discuss the agent's capability with respect to specific descriptions that may or may not be exactly the same as its advertised description.

The base capability rating, that is, the capability rating for the agent with respect to this base description is generated randomly according to Gaussian distribution using a predefined mean (e.g., 0.6) and standard deviation (e.g., 0.1). It is important to point out that this capability rating will not be given to the matchmaker agent, but rather, is kept in the simulation engine for purposes such as generating satisfaction ratings. The matchmaker will establish and refine an agent's capability model through the interaction with the agents.

The capability ratings of an agent with respect to specific service descriptions will be generated based on the base rating, dynamically as needed. For example, for an agent that advertised a service for international flight ticket reservation and with base rating 0.75, its capability rating with respect to the service of flight ticket reservation to China could be 0.8. Because the capability ratings are dynamically generated as queried, one of the most important issues is consistency. If the capability of an agent with respect to a specific service description is queried twice, it should produce the same rating.

Another perspective on the consistency issue is that the ratings should be consistent between the sub-service categories and their super-service category. Suppose there are only two kinds of computers, laptops and desktops, and a company makes only computers. If the company receives 80% satisfaction rating on its laptops and 85% satisfaction rating on its desktops, then the overall satisfaction rating on computer products can not be anything lower than 80% or higher than 85%.

The capability rating of an agent with respect to a given (or query) service description is determined by three factors: the base rating (of the advertised service), how the base description matches the given query description, and the distribution of the services. Briefly, the algorithm for capability rating generation can be described as follow:

1) If the query description is identical to the advertised service (base) description (an exact match), the capability rating with respect to the query description will be the same as the base rating.

2) If the base description does not match (in the sense of a service matching) the query description, the capability rating with respect to the query description will be 0.0.

3) If the base description matches the query description at subsumes level, that is, if the base description is more specific than the query description, the rating (with respect to the query description) will be the base rating multiplied by a factor *f*, where *f* is the percentage the base description accounts for in the query service space, that is,

   $f = | S_{adv} | / | S_{query} |$

   Where S_{adv} is the advertised service space, S_{query} is query service space. The factor *f* may be computed based on the distribution model.

4) If the base description matches the query description at intersection level, that is, if the base description and the query description has common subset, the rating (with respect to the query description) will be the base rating multiplied by a factor *f*, where *f* is the percentage the common subset accounts for in the query service space, that is,

   $f = | S_{adv} \cap S_{query} | / | S_{query} |$

   As in (3), *f* may be computed based on the distribution model.

5) If it is none of the above, we know it is a plug-in match, that is, the advertised service is more general than the query service. To reflect the assumption that an agent may have strong and weak areas in its service offerings, the rating with respect to the specific query is computed randomly according Gaussian distribution where the mean is the base rating and the standard deviation is specified as a fraction of the mean.

7.2.3 Generating requests and feedback

The generation of service recommendation requests is much easier. The service descriptions are generated based on the given profile template and the pre-generated distribution model. For example, if the distribution model shows that the economy ticket accounts for 80% of all the flight tickets, then the chance that the flightTicket parameter has type EconomyTicket in the generated flight ticket reservation service profile will also be around 80%.

When matches are recommended by the matchmaker agent (in response to a matching request), the simulation engine acts as if it is the service consumer agent and assigns a satisfaction rating to the service. The satisfaction rating is calculated based on the service provider's “true” capability rating with respect to this specific request, with the assumption that statistically a service consumer's experience should be consistent with the "true" capability of the service provider agent. Then feedback will be generated and sent back to the matchmaker agent. Feedback can be turned on or off in the simulation engine.

7.2.4 The matchmaker categories

The performance of our matchmaker agent is evaluated by comparing it against three other types of matchmakers, namely the type-1 matchmaker, type-2 matchmaker, and type-4 matchmaker. Our matchmaker is type-3 in this group. These matchmaker types are hypothetical and are defined solely for the purpose of the evaluation of this work.

A **type-1 matchmaker** is the basic matchmaker in the group. It performs semantic match and it only tells if an advertised service matches a request and will not tell the degree (or level) of the match. In other words, you will have the list of matches (if any) but there is no hint on which one is better (or worse) than any of the others. At this level, an advertised service matches a given request if all of the following hold:

- The profile type (T_a) of the advertised service is compatible with that of the requested service (T_r), that is, $T_a \subseteq T_r$ or $T_r \supseteq T_a$.
- For each input parameter in the advertised service description, there is a matching input parameter in the requested service description.
- For each output parameter in the requested service description, there is a matching output parameter in the advertised service description.
- A parameter can be used at most once in parameter matching.

As discussed in the previous chapters, there can be four levels of parameter match: exact match, plug-in match, subsumes match, and intersection match. Exact match and plug-in match are always considered match (by a basic level matchmaker). As for subsumes match, it may be considered a match only when the system configuration setting for "include-subsumes" is true. Intersection match is not considered in this simulation.

A **type-2 matchmaker** is an advanced matchmaker that works in a way similar to a type-1 matchmaker but in addition to determining whether an advertised service matches a requested service, it also tells you the level of match in terms of a set of pre-defined match levels. The work in [56] falls in this category. In [56], there are three levels of parameter matches: exact match, plug-in match, and subsumes match. The lowest input match level of all input parameters is the overall match level of input parameters; similarly, the lowest output match level of all output parameters is the overall match level of output parameters. The rule for comparing two service matches is described in [56] as follow:

```
If match1.output > match2.output then
        match1 > match2;
If match1.output = match2.output && match1.input > match2.input then
        match1 > match2
If match1.output = match2.output && match1.input = match2.input then
        match1 = match2
```

The output is the principle factor in the ordering of matches because in [56], its assumed that the main concern of the requester is that the provider can achieve the output with the

highest degree. The major differences of a type-2 matchmaker and our matchmaker (type-3) is that a type-2 matchmaker considers only service descriptions in service matching, while type-3 matchmaker takes advantage of the agent capability model in service matching, as discussed in Chapter 1.

A **type-4 matchmaker** is an ideal or perfect matchmaker that always (magically) finds the best matches available. Here the term "best" is not judged in terms of the profile descriptions, but in terms of the "actual" capability that a provider can perform the requested services with the highest degree of overall satisfaction. In this experiment, the type-4 matchmaker is hooked to the simulator in the same way as the other three types of matchmakers except that it directly gets the best service provider (for a incoming request) from the simulator based on the "true" capabilities of the service provider agents.

7.3 The Result Statistics

To run the test, a few things need to be prepared. First, a distribution model is generated as discussed earlier. Second, a sequence of advertised service profiles is generated. And third, a sequence of (matching) request service profiles is generated by following the distribution model. In the initial design, these service profiles are generated on the run, but later we decided to have them generated offline for easier debugging and evaluation.

The test process is as follow. The simulator advertises 50 services to the matchmaker, followed by 1000 matching (recommendation) requests. When the simulator receives recommendations from the matchmaker (in response to a matching request), a satisfaction rating is generated based on the service provider agent's "true" capability rating, and then a feedback is generated and sent back to the matchmaker agent. Unless otherwise specified, the statistics discussed in this section is based on 20 such tests.

As discussed earlier, the evaluation of the matchmaker agent will be primarily about comparing against the other three types of matchmakers: type-1 matchmaker, type-2 matchmaker, and type-4 matchmaker (our matchmaker is classified as type-3 matchmaker in the group). The process described above is repeated for each of the four types of matchmakers. The statistics fall into two major categories: those with the subsumes-level match turned on (in which a subsumes-match is considered a match) and those with the subsumes-level match turned off (in which a subsumes-match is not considered a match). The intersection level match is not considered in this test, mainly due to the lack of comparison metrics and in general, intersection match is of less interest.

Before analyzing the statistical numbers and figures, it is essential to point out that the absolute satisfaction rating values do not mean anything by themselves because for some request, there is simply no good providers to recommend; it is the comparison across different types of matchmakers that tells the story.

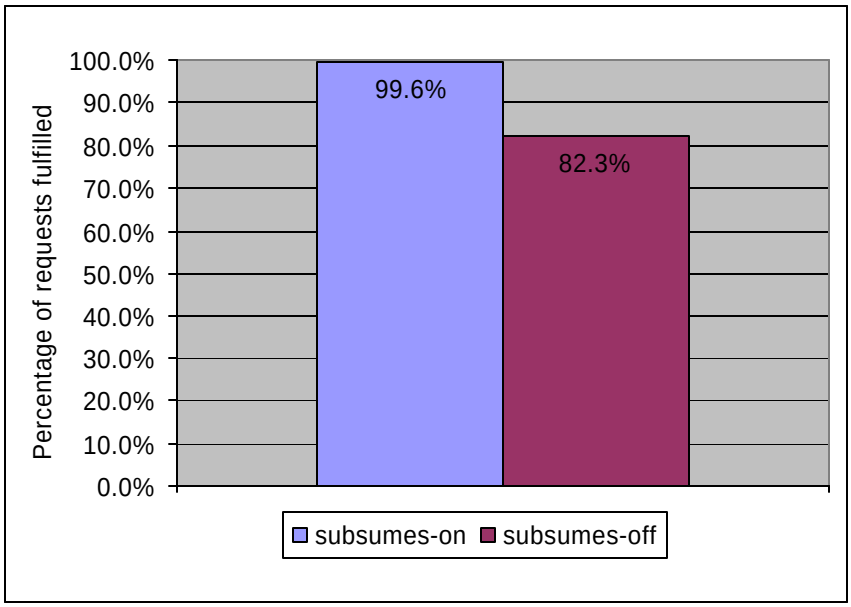


Figure 7-3. Percentage of requests fulfilled

As mentioned earlier, the intersection level match is not considered in this experiment, and the subsumes-level match may be turned on (that is, subsumes matches are considered as valid matches, though only potentially), or turned off (that is, subsumes matches are not considered as valid matches). Figure 7-3 shows that in our tests, some 82.3% of the requests are fulfilled (i.e., some matches are found) when subsumes-level match is turned off; while with subsumes-level match turned on, nearly 99.6% of the requests are fulfilled, though some 17.3% (99.6% - 82.3%) may get only potential matches. This is the same across all four types of matchmakers.

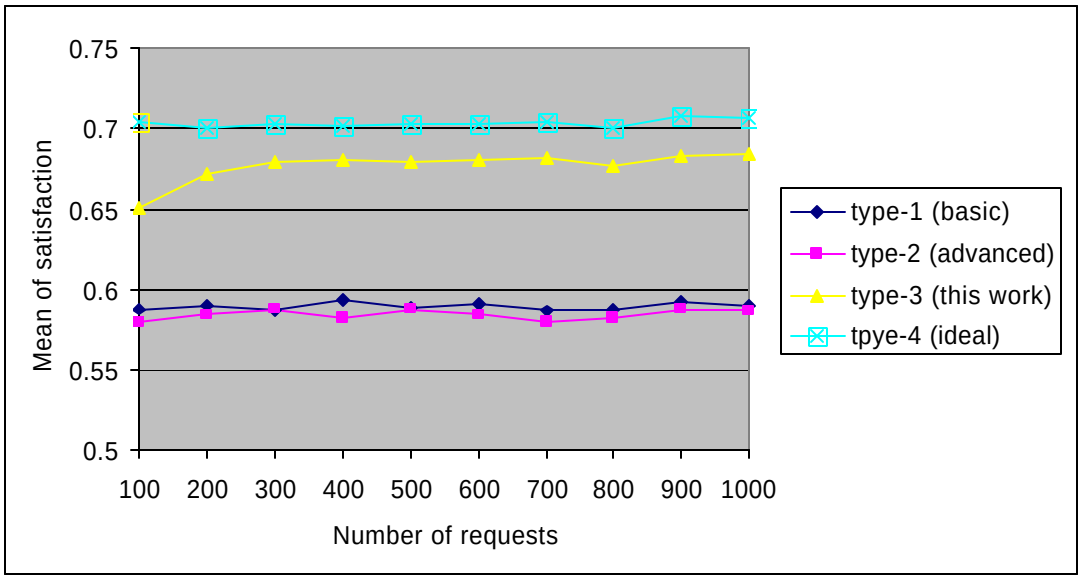


Figure 7-4. Mean of satisfaction ratings, subsumes off

Figure 7-4 shows the comparison among the different types of matchmakers on the mean of satisfaction ratings. In this figure, the subsumes-match is turned off, that is, only exact matches and plug-in matches are qualified as matches.

In the first 100 requests, our matchmaker (type-3) does not perform very well but it still performs better than the type-1 matchmaker and type-2 matchmaker. In the second and the third 100 requests, our matchmaker appears to have learned quite a bit about the agents' capabilities and its performance is catching up with the type-4 (ideal) matchmaker. Starting from the fourth 100 requests, our matchmaker performs better than the type-1 and type-2 matchmaker by about 15-20%, and is about 3% below the ideal matchmaker.

The figure also shows that the type-1 matchmaker and the type-2 matchmaker perform similarly, which is not a surprise. The main advantage of type-2 matchmaker over type-1 matchmaker is that the type-2 matchmaker differentiates different levels of matches, for example, an exact match is considered better than a plugIn match, which in turn is considered better than a subsumes match. With subsumes-level match turned off, the only matches are exact matches and/or plug-in matches and in this simulation, the way an agent's capability rating is generated does not explicitly favor exact matches over plugIn matches. As a result, the type-2 matchmaker and the type-1 matchmaker have similar performance levels, as shown in Figure 7-4.

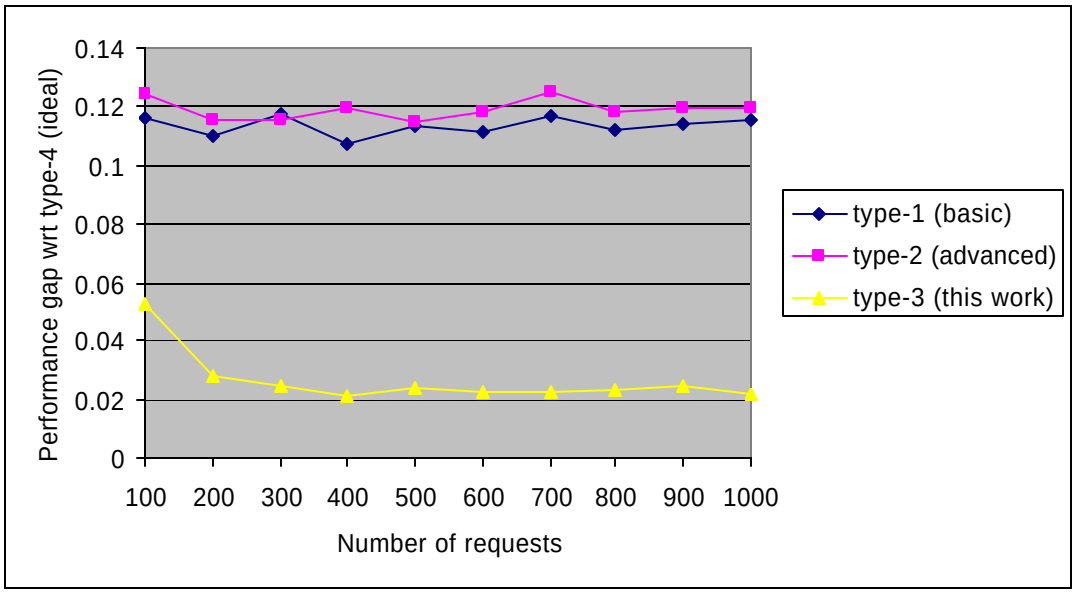


Figure 7-5. Standard deviation of satisfaction ratings, subsumes off

Figure 7-5 shows for each type of matchmaker the performance gap (difference of satisfaction ratings) with respect to the type-4 ideal matchmaker. Basically it is the same information as in Figure 7-4 but from a different perspective.

t-Test: Paired Two Sample for Means

	Type -3	*Type - 2*
Mean	0.6769215	0.5840265
Variance	0.007171654	0.003818955
Observations	20	20
Pearson Correlation	0.800953213	
Hypothesized Mean Difference	0	
Df	19	
t Stat	8.136125347	
P(T<=t) one-tail	6.51631E-08	
t Critical one-tail	1.729131327	
P(T<=t) two-tail	1.30326E-07	
t Critical two-tail	2.093024705	

Table 7-1. Paired t-test on the performance, type-2 vs. type-3 matchmaker

As mentioned earlier, we ran the test 20 times (on different data) for each type of matchmaker. To find out if the performance difference between the type-2 matchmaker and the type-3 matchmaker is statically significant, we computed the average satisfaction rating for each test for both type-2 and type-3 matchmakers and performed paired t-test on their average satisfaction ratings (20 for each type of matchmaker). The result of this

paired t-test is shown in Table 7-1. To claim with 95% confidence that the difference is statically significant, we need a t-value of at least 2.09 and our t-value is 8.13, which far exceeds the minimum requirement of 2.09.

Improvement ratio (type-3 over type-2)	
Mean	0.160135458
Standard Error	0.020687086
Count	20

Table 7-2. Performance improvement, type-3 over type-2

Based on the same data (average satisfaction rating for each test) as above, Table 7-2 shows that statistically, the type-3 matchmaker performs some 16% better than the type-2 matchmaker, with a standard error of about 2%.

Now we look at the issue of how the feedback ratio, that is, the percentage of service consumers that will provide feedback to the matchmaker for a service recommendation, affects the performance of type-3 matchmaker.

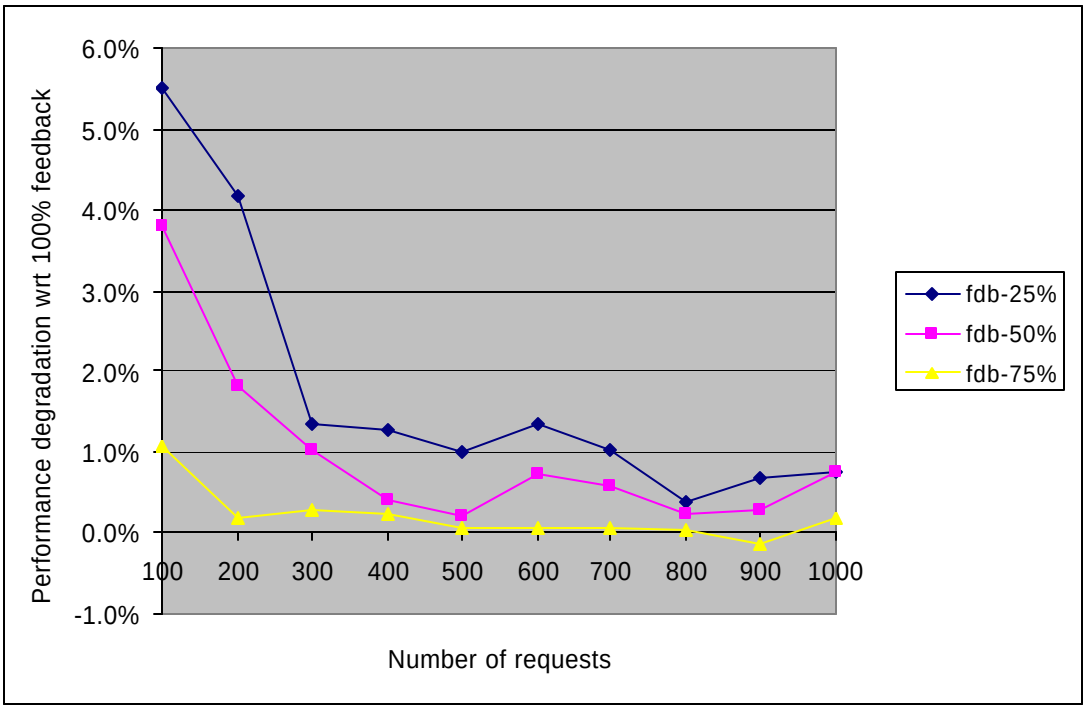


Figure 7-6. The effect of feedback percentage

Figure 7-6 shows the performance differences (degradation) for 25%, 50%, and 75% feedback ratio with respect to 100% feedback ratio. It is not surprising that the higher the feedback percentage, the better the overall performance; the lower the feedback percentage, the longer it takes for the matchmaker to capture the capabilities of the agents. But overall, after 300 to 400 requests, the performance differences among the different feedback ratios become very small, to about 1%.

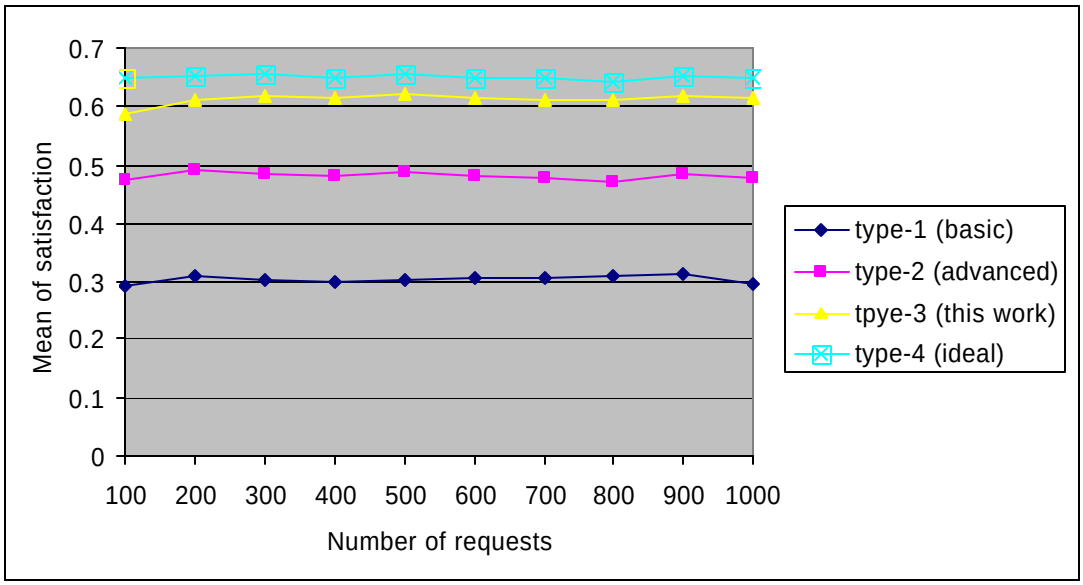


Figure 7-7. Mean of satisfaction ratings, subsumes on

Figure 7-7 shows similar measures as Figure 7-4 but with the subsumes-level match turned on. In other words, a subsumes-level match will be considered as a match, though subsumes matches are potential matches and may not be as good as the other levels of matches. The type-4 matchmaker performs the best (of course), followed by our (type-3) matchmaker, but this time, the type-2 matchmaker performs much better than the type-1 matchmaker.

This test sets apart the type-2 matchmaker from the type-1 matchmaker mainly because the type-1 matchmaker does not differentiate different levels of matches and therefore can not tell which matches are better then the others. The type-2 matchmaker, however, ranks matches based on the match levels.

Another observation is that with subsumes match turned on, the improvement of our (type-3) matchmaker over type-2 matchmaker is at least as great as when subsumes match is turned off (see Figure 7-4). We think that there are three reasons behind this. First of all, type-3 matchmaker takes into account an agent's performance history in service matching while type-2 does not. Second, our matchmaker (type-3) takes into account the service distribution factor in deciding the degree of a potential match and therefore, our (type-3) matchmaker may tell that certain potential matches are actually more relevant than the others. Third, the type-2 matchmaker favors matches with higher

overall output parameter match levels (over the input match levels) because it assumes that the main concern of the service requester is that the provider can achieve the output with the highest degree. For example, suppose there are two matches, match-1 and match-2. In match-1, all output parameters match at the exact match level and all input parameters match at subsumes level; in match-2, all parameter matches are exact matches except that one of the output parameter matches at the subsumes level. In this case, the type-2 matchmaker will choose match-1 over match-2 since match-1 has better overall output match level. This assumption, however, is not reflected in the simulation framework in the sense that it is not taken into account in generating the satisfaction ratings (for feedback). In this sense, this comparison may be a bit unfair to the type-2 matchmaker, though we do not think this is the major factor.

In summary, we think that the experiments and the analysis demonstrate that the ideas and the algorithms discussed in this work enable the matchmaker agent to establish and refine the capability model of the service provider agents and can significantly improve the quality of the matchmaking services of the matchmaker agents.

8 Conclusion and Discussions

Agent technology is becoming real. In the real world applications, an agent's performance matters; an agent may or may not perform equally well across its service offerings, in other words, an agent may have strong and weak areas; an agent's capability to perform specific tasks may change/improve even though its advertised service descriptions remain unchanged, for example, as a result of learning; and the distribution of services may give us some hint on which potential matches are better than the others. However, these issues are largely overlooked in today's service matching methods.

This dissertation took the position that an agent's performance history should be an integral part of its capability model in addition to the advertised service (capability) descriptions. We proposed a method for the middle agent to learn about the capabilities of the service providers by establishing and refining an agent's capability model through the interactions with the agents. With such a capability model, an agent's performance may be estimated with respect to any given request, and service distribution is taken into account, especially in deciding the strength of potential matches. Service matching is then a two step process. In the first step, candidates are selected through the regular service description matching. In the second step, the performance of each candidate with respect to the specific request is estimated based on the agent capability model and then the candidates with the highest estimated performance ratings will be selected.

We think the experimental results demonstrated that the quantitative learning approach discussed in this work enables the matchmaker agent to establish the capability model of the service provider agents, with which the accuracy of service provider selection has been significantly improved. We think this will make a significant difference in the real world applications.

That being said, there are a few issues that may deserve further exploration in the future work.

1) Domain-specific service features as part of the agent capability model

   The OWL-S Profile represents two aspects of the functionality of the service: the information transformation (represented by inputs and outputs) and the state change produced by the execution of the service (represented by preconditions and effects). But services may have domain-specific features. Airline services, for example, may have such features as meal quality, friendliness, etc. Such features can be defined as another type of parameters (in addition to the IOPEs) that are ratable, that is, ratings may be assigned with respect to specific features (in addition to the overall satisfaction rating). This can easily be incorporated into the agent capability model by extending the rating entry structure with features.

2) Multiple middle agents

   There can be various reasons why you need multiple middle agents in a given agent system. For example, using multiple middle agents can prevent from the problem of single point of failure and can support a larger agent population. Another advantage is that each middle agent may focus on a subset of the overall

service space and become highly specialized in the area. Issues that need to be addressed include (but not limited to) how the middle agents are organized, for example, hierarchical or peer-to-peer; how the middle agents interact with each other; and how to find the middle agent that is "specialized" in the area of the request.

3) Swapping to the disk

   The learning model is relatively memory intensive and it will not be unusual if the middle agent run out of memory with a large agent population. Studying the strategies of swapping the agent capability models to the disk may be essential to ensure the scalability of such an agent system.

4) Precondition and effect parameters.

   The precondition and effect parameters are largely unconsidered in the agent capability modeling, but with the progress on SWRL (Semantic Web Rules Language) at W3C, the role of preconditions and effects in agent capability modeling deserves a closer look.

With the research in these areas we hope to take the method discussed in this dissertation a step closer to the practical applications.

9 References and Bibliography

[1] S. Ahuja, N. Carriero and D. Gelernter, Linda and Friends. *IEEE Computer*, vol. 19, no. 8, August 1986, pp. 26-34.

[2] S. Christain Albright, Wayne L. Winston, and Christopher Zappe. Data Analysis for Managers (Second Edition), Thomson Brooks/Cole, 2004.

[3] Y. Arens, C. Chee, C. Hsu, H. In, and C. A. Knoblock, Query Processing in an Information Mediator.

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. The Description Logic Handbook. Cambridge University Press, 2002

[5] A. L. C. Bazzan, Evolution of Coordination as a Metaphor for Learning in Multi-Agent Systems, In G. Weiss, editor, (LNAI 1221), Distributed Artificial Intelligence Meets Machine Learning, pages 117-136. Springer Verlag, 1997

[6] Tim Berners-Lee, James Hendler and Ora Lassila, The Semantic Web, Scientific American, May 2001.

[7] Jose C. Brustoloni, Autonomous Agents: Characterization and Requirements, Carnegie Mellon Technical Report CMU-CS-91-204, Pittsburgh, Carnegie Mellon University, 1991.

[8] C. Byrne and P. Edwards, Refinement in Agent Groups, in G. Weiss, and S. Sen, editors, (LNAI 1042), Adaptation and Learning in Multi-Agent Systems, Pages 22-39. 1995

[9] W. Cohen, A. Borgida, and H. Hirsh, Computing Least Common Subsumers in Description Logics. Proceedings of the National Conference on Artificial Intelligence - AAAI 92, pp 754-760, 1992

[10] David Carmel, and Shaul Markovitch, Oppoent Modeling in Multi-Agent Systems, in G. Weiss, and S. Sen, editors, (LNAI 1042), Adaptation and Learning in Multi-Agent Systems, Pages 40-52. 1995

[11] Daniel D. Corkill, Collaborating Software: Blackboard and Multi-Agent Systems & the Future. In *Proceedings of the International Lisp Conference*, New York, New York, October 2003.

[12] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms. The MIT Press.

[13] Daml.org, OWL-S Release 1.0. http://www.daml.org/services/owl-s/1.0/

[14] Daml.org, Reference description of the DAML+OIL (March 2001) ontology markup language, http://www.daml.org/2001/03/reference.html.

[15] R. Davis and R. G. Smith, Negotiation as a metaphor for distributed problem solving, Artificial Intelligence, 20(1), 63109, January 1983

[16] K. Decker, K. Sycara, and M. Williamson, Middle-Agents for the Internet. In IJAI-97 International Joint conference on Artificial Inelligence, Nagoya, Japan, 1997.

[17] K. Decker, K. Sycara, and M. Williamson, M, Modeling Information Agents: Advertisements, Organizational Roles, and Dynamic Behavior. Working Notes of the AAAI-96 workshop on Agent Modeling, AAAI Report WS-96-02. 1996.

[18] K. Decker, M. Williamson, and K. Sycara, Matchmaking and Brokering, 1996. Downloaded from the site of cs.cmu.edu.

[19] C. Dellarocas, Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. Proceedings of the 2nd ACM Conference on Electronic Commerce, Minneapolis, MN, October 17-20, 2000

[20] FIPA.org, http://www.fipa.org

[21] FIPA 97 Specification Part 1, Agent Management, http://www.fipa.org

[22] FIPA 97 Specification Part 2, Agent Communication Language, http://www.fipa.org

[23] Stan Franklin and Art Graesser, Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

[24] Ernest Friedman-Hill, Jess the Rule Engine for the Java. http://herzberg.ca.sandia.gov/jess/.

[25] H. Friedrich, M. Kaiser, O. Rogalla, and R. Dillmann, Learning and Communication in Multi Agent Systems, In Weiss, G., editor, (LNAI 1221), Distributed Artificial Intelligence Meets Machine Learning, pages 259-275. Springer Verlag, 1997

[26] M. R. Genesereth and N. P. Singh, A Knowledge Sharing Approach to Software Interoperation. Stanford Logic Group Report Logic-93-12.

[27] Warren Gilchrist, Statistical Forecasting, John Wiley & Sons, 1977.

[28] T. R. Gruber, The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases, in Allen, J. A., Fikes, R., and Sandewall, E. (Eds), Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference. San Mateo, CA: Morgan KaufMann, 1991.

[29] T. R. Gruber, A Translation Approach to Portable Ontologies. Knowledge Acquisition, 5(2):199-220, 1993.

[30] Pan Gu and Anthony B. Maddox. A framework for distributed reinforcement learning. In Gerhard Weiss and Sandip Sen, editors, Adaption and Learning in Multi-agent Systems: IJCAI'95 Workshop, Montreal Canada, August 1995, Proceedings, pages 85-96. International Joint Conference of Artificial Intelligence, Springer, 1995.

[31] Thomas Haynes, Sandip Sen, Dale Schoenefeld, and Roger Wainwright, ``Evolving a Team", in the AAAI Fall Symposium on Genetic Programming, in Cambridge, MA, November 10 - 12, 1995.

[32] James Hendler, Tim Berners-Lee, and Eric Miller, "Integrating Applications on the Semantic Web," Journal of the Institute of Electrical Engineers of Japan, Vol 122(10), October, 2002, p. 676-680.

[33] Ming-Yang Kao, Tak-Wah Lam, Wing-Kin Sung, Hing-Fung Ting. A Decomposition Theorem for Maximum Weight Bipartite Matchings, SIAM Journal on Computing, Volume 31, Number 1, pp. 18-26.

[34] M. Klusch and K. Sycara, "Brokering and Matchmaking for Coordination of Agent Societies: A Survey." In Coordination of Internet Agents, A. Omicini et al. (eds.), Springer, 2001.

[35] Joe Kopena, DAMLJessKB, http://edge.mcs.drexel.edu/assemblies/software/damljesskb/damljesskb.html.

[36] Y. Labrou, Semantics for an Agent Communication Language. PhD Dissertation, University of Maryland, Baltimore County.

[37] Yannis Labrou and Tim Finin, A Proposal for a new KQML Specification. TR CS-97-03, February 1997, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County.

[38] P. Lambrix and J. Maleki. Learning Composite Concepts in Description Logics: A First Step. Proceedings of the 9th International Symposium on Methodologies for Intelligent Systems - ISMIS 96, LNAI 1079, pp68-77, 1996

[39] B. Lenzmann and I. Wachsmuth, Contract Net Based Learning in a User-Adaptive Interface Agency, In G. Weiss, editor, (LNAI 1221), Distributed Artificial Intelligence Meets Machine Learning, pages 223-241. Springer Verlag, 1997

[40] Lei Li and Ian Horrocks. A Software Framework For Matchmaking Based on Semantic Web Technology. WWW2003, May 20-24, 2003, Budapest, Hungary. (http://www2003.org/cdrom/papers/refereed/p815/p815-li.html)

[41] Xiaocheng Luan, Yun Peng, and Timothy Finin, Learning in the Broker Agent. In W. Truszkowski, C. Rouff, M. Hinchey (Eds), LNAI 2564, Innovative Concepts for Agent-Based Systems, pages 106 – 121.

[42] Pattie Maes, "Artificial Life Meets Entertainment: Life like Autonomous Agents," Communications of the ACM, 38, 11, 108-114.

[43] David L. Martin, Adam J. Cheyer, and Douglas B. Moran, *The Open Agent Architecture: A Framework for Building Distributed Software Systems.* Applied Artificial Intelligence, vol. 13, no. 1-2, pp. 91-128, January-March 1999.

[44] MCC, Collaborative Brokering. Technical report, MCC, INSL-093-97. Abstract.

[45] Joe McDonald, "Mean, Variance, and Standard Deviation", http://joemath.com/applets/stats

[46] S. McIlraith, T. C. Son, and H. Zeng, `Semantic Web Services, IEEE Intelligent Systems. Special Issue on the Semantic Web. To appear, 2001.

[47] Kurt Mehlhorn, Stefan Naeher. LEDA, A Platform for Combinatorial and Geometric Computing, Cambridge University Press, 1999.

[48] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Machine Learning, An Artificial Intelligence Approach, Tioga Publishing Company

[49] Hao-Chi Wong and Katia Sycara "A Taxonomy of Middle-agents for the Internet" In Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS'2000)

[50] Lik Mui, Mojdeh Mohtashemi, Cheewee Ang, A probabilistic Rating Framework for Pervasive Computing Environments. The Oxygen Workshop, 2001.

[51] Lik Mui, P. Szolovitz, and C. Wang, Sanctioning: Applications in Restaurant Recommendations based on Reputation, Proceedings of the Fifth International Conference on Autonomous Agents, Montreal, May 2001.

[52] Engineering Statistics Handbook, *NIST/SEMATECH e-Handbook of Statistical Methods*, http://www.itl.nist.gov/div898/handbook/index.htm

[53] Marian Nodine, William Bohrer, Anne Hee Hiong Ngu, Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth. In Proceedings of the International Conference on Data Engineering, 1999

[54] M. Nodine and B. Perry, Experience with the InfoSleuth Agent Architecture, To appear in Proceedings of AAAI-98 Workshop on Software Tools for Developing Agents, 1998.

[55] N. Ono and K. Fukumoto, A Modular Approach to Multi-Agent Reinforcement Learning, In Weiss, G., editor, (LNAI 1221), Distributed Artificial Intelligence Meets Machine Learning, pages 25-39. Springer Verlag, 1997

[56] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara; "Semantic Matching of Web Services Capabilities."Proceedings of the 1st International Semantic Web Conference (ISWC2002)

[57] Christos H. Papadimitriou, Kenneth Steiglitz. Combinatorial Optimization, Algorithms and Complexity. Prentice-Hall, 1982.

[58] Yun Peng, Nenad Ivezic, Youyong Zou, and Xiaocheng Luan. Semantic Resolution for E-Commerce. To appear in the Proceedings of the First Workshop on Radical Agent Concepts. Greenbelt, Maryland. September 2001.

[59] E. Plaza, J.L. Arcos, F. Martín (1997); Cooperative Case-Based Reasoning. Lecture Notes in Artificial Intelligence. Springer, Vol. 1221, pp.180-201. (IIIA-RR-97-01),

[60] N. P. Singh, A Common Lisp API and Facilitator for ABSI, version 2.0.3 Stanford Logic Group Report Logic-93-4

[61] Reid G. Smith, The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, IEEE Transactions on Computers, Vol. C-29, Pages 1104-1113, 1980

[62] K. Sycara, J. Lu, and M. Klusch, Interoperability among Heterogeneous Software Agents on the Internet. CMU-RI-TR-98-22.

[63] UDDI, Universal Description, Discovery, and Integration, http://www.uddi.org/.

[64] W3C, Web Ontology Language (OWL), http://www.w3.org/2004/OWL.

[65] W3C, Semantic Web, http://www.w3.org/2001/sw/

[66] W3C, Web Services Activity, http://www.w3.org/2002/ws/

[67] P. Weinstein and W. P. Birmingham, Comparing concepts in differentiated ontologies. Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management (KAW'99).

[68] G. Weiss, Adaptation and Learning in Multi-Agent Systems: Some Remarks and a Bibliography, in Weiss, G., Sen, S. editors, (LNAI 1042), Adaptation and Learning in Multi-Agent Systems, Pages 1-21. 1995

[69] G. Wickler, CDL: An Expressive and Flexible Capability Representation for Brokering. gw@itc.it.

[70] Microsoft Corporation, Web Services Description Language (WSDL) 1.1, January 23, 2001, Microsoft Corporation, http://msdn.microsoft.com/xml/general/wsdl.asp

[71] Michael Wooldridge and Nicholas R. Jennings, Intelligent Agents: Theory and Practice, The Knowledge Engineering Review 10 (2), 115 – 152, 1995

[72] Giorgos Zacharia, Alexandros Moukas and Pattie Maes, Collaborative Reputation Mechanisms in Electronic Marketplaces. Proceedings of the 32nd Hawaii International Conference on *System Sciences*, 1999.

[73] Yuanyuan Zhou, James F. Philbin, and Kai Li, The Multi-Queue Replacement Algorithm for Second Level Buffer Caches Proceedings of the 2001 USENIX Annual Technical Conference, June 25 - 30, 2001, Boston, Massachusetts, USA