Learning in the Broker Agent

Xiaocheng Luan[1], Yun Peng[2], and Timothy Finin[2]

[1] Aquilent Inc. 22215 Overview Lane
Boyds, MD 20841, USA
xluan1@cs.umbc.edu
[2] Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250, USA
{ypeng,finin}@cs.umbc.edu

Abstract. Service matching is one of the crucial elements in the success of large, open agent systems. While finding "perfect" matches is always desirable, it is not always possible. The capabilities of an agent may change over time; some agents may be unwilling to, or unable to communicate their capabilities at the right level of details. The solution we propose is to have the broker agent dynamically refine the agent's capability model and to conduct performance rating. The agent capability model will be updated using the information from the consumer agent feedback, capability querying, etc. The update process is based on a concept of "dynamic weight sum system", as well as based on the local distribution of the agent services. We assume that the agents in the system share a common domain ontology that will be represented in DAML+OIL, and the agent capabilities will be described using DAML-S.

1 Introduction

Finding the right agent(s) for the right task is critical in achieving agent cooperation in large, open agent systems. A popular approach to this problem is to use a broker agent (or in general, the middle agents) to connect the service provider agents and the service consumer agents, via service matching. Typically a broker agent recommends service providers based on the capabilities/services advertised by the service provider agents themselves. The matching scheme has evolved from the early age, simple KQML performative based matching [17], to syntax and semantic based matching [27]; from returning yes/no exact matches to returning matches with probabilities [32]. However, there are still issues that need to be addressed. The ability to learn is one of the key properties of the agents, therefore, the capabilities of an agent, both in terms of what it can do and how well it can do it, will likely to change over time. Moreover, the advertised capability information may not be always accurate - some agents may be unwilling or unable to advertise their capability information at sufficient level of details, some might unknowingly advertise inaccurate information, while others might even purposefully provide misleading information of their capabilities. Even when the capability information is “accurate”, the agent selected may not be able to provide quality service(s).

We have similar problems in the real world: we don't know whether the colorful, fancy, and even convincing commercials are true or not. There is no perfect solution to this real world problem - people have to learn their lessons. People can learn from their own experience - if you bought a bottle of milk from a super market, but the milk was sour, you will be less likely to buy milk from that store again. People can also learn from their friends' experience. While this kind of learning is very helpful, it's usually insufficient, because an individual usually has a limited social circle and therefore, the experience is limited, both in terms of the variety of experience and the number of occurrences. That is why there are the consumer reports. Consumer reports are created using the information from the manufacture's specification, the consumer's feedback, and their test results on the products. It provides guidance for consumers to choose the right products. We believe that this consumer reports approach should work in the agent world, too.

A particular agent can certainly try to learn which agents can provide good services for it. However, its contact with other agents is usually limited, not to mention that it usually has its own specialized work to perform. Therefore, to have each agent to perform the learning would create a huge burden both for the agent itself and for the agent developer(s). The broker agent, however, typically interacts with many (if not all) of the agents in the system, and therefore is the ideal candidate for collecting and summarizing the agents' experience and composing the “consumer reports” for the other agents. This new task is consistent with its ultimate goal, that is, to provide the best recommendations to the service consumer agents. By following a brokering protocol, the broker agent will not only collect the information advertised by the service provider agents, but will also learn from the experience the consumer agents have about their service providers. It can also interrogate (query) a service provider agent to get more detailed information on the services it can provide. Moreover, the broker agent can dynamically capture the local probabilistic distribution of the agent services and use this information to assess the probability of a service match.

For the same reason that an agent's capability (description) may change over time (e.g., through learning), the significance of a piece of feedback data may also change over time. For example, recent feedback data might be considered “more important” than the earlier feedback data. To address this problem, we model the system as a dynamic, weighted sum system. When new data come in, new weights are generated for them and the weights for the data obtained earlier will be recomputed based on a pre-specified pattern or trend, so that the total weights still sum to 1. There is a family of weight sequence functions that is of special interest - the sequence functions that have the “incremental property”. When a new weight sequence is generated due to the increase in the number of data samples, the new “total result” can be computed based on the previous total result and the new data (and of course, the new weights), without re-computing the whole thing.

Finally, our approach goes beyond the simple notion of a “reputation server” in that it discovers and refines a complex, symbolic model of a service provider's capability and performance.

The rest of this article is organized into three sections. In Section 2, we briefly introduce the related work in the area, as well as the technologies that will be used in this work, such as DAML+OIL and DAML-S. In Section 3 we discuss the refinement of an agent's capability model as well as the performance rating on the agents. We conclude the paper with the discussions on some related issues in Section 4.

2 Related Work and a Background

The area of agent service matching has been intensively researched in the past years because of its significance to the success of an agent system. In the early time, Agent Based Software Interoperation (ABSI) architecture [17]), a special kind of agent in the system, called the facilitator, is responsible for content-based message routing (basically based on the KQML performative in a message). This is essentially a KQML performative-based service matching, that is, the capability of an agent is described by what KQML performatives it can handle. More recent brokers usually support semantic based service matching, like the broker agent in the InfoSleuth Agent Architecture [27]. The SIMS information mediator [1] does more than simple service matching, it provides access and integration of multiple sources of information. When no direct mapping can be found, it can extend the search through concept generalization and specialization.

An interesting work on service matching is the LARKS (Language for Advertisement and Request for Knowledge Sharing) [32]. LARKS is an agent capability description language developed at CMU. It describes an agent's service by specifying the context, the data types, the input and output variables, and the input and output constraints. It also has a slot for the definition of the concepts used in the description. The matchmaking scheme in LARKS is fairly flexible. There are five filters, each of which 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 syntactically match; while the “semantic matching” checks if the input/output constraints of a pair of descriptions are logically matched. Based on the need of a specific application domain, these filters can be combined to achieve different types/levels of matching.

The work in [33] compares concepts in differentiated ontologies. Differentiated ontologies are (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.

Most of the research/work on reputation management is in the context of electronic marketplaces. In [37], the author 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 described in the paper is more interesting. It models the 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 [25] 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".

In comparison to the existing researches, our proposed approach allows the broker agent to refine the capability model of an individual agent, and to provide performance rating. Moreover, the performance ratings of an agent are considered an integral part of an agent's capability model. This is consistent with the DAML-S service ontology, in which a qualityRating attribute is defined in the service profile. The broker agent also approximates the probabilistic distribution of the agent services by capturing the local distribution. In this work, the ontology and the service description will be represented with DAML+OIL and DAML-S, respectively.

DAML+OIL is the result of the joint effort by the US DARPA Agent Markup Language project and the EU Information Society Technologies Program (IST). It is a semantic markup language for Web resources. It builds on earlier W3C standards such as RDF and RDF Schema, and extends these languages with richer modeling primitives. DAML+OIL provides modeling primitives commonly found in frame-based languages [7]. Therefore, we think it is suitable for use in ontology definition, manipulation, and reasoning. With DAML+OIL, one can define classes and properties, specify property restrictions, etc.

DAML-S is a web service ontology built on top of DAML+OIL. It is still an ongoing work at the DAML program. It supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their services in unambiguous, computer-interpretable form [8]. It describes a service in terms of "service profile", "service model", and "service grounding". The service profile tells what the service does; the service model tells "how the service works"; the service grounding specifies how the service can be accessed. DAML-S could facilitate the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring.

3 Capability Model Refinement and Performance Rating

As mentioned earlier, finding the right agent(s) for a given task is critical in achieving agent cooperation in large, open agent systems. Typically a broker agent recommends service providers based on the capabilities or services advertised by the service providers themselves. But there are issues yet need to be addressed. For example, given the adaptive nature of the agents, the capabilities of an agent are likely to change over time; the advertised capability may not be always accurate, and that an agent with the right capability description may not provide quality services. In this work, we propose to extend the capability of the broker agent, i.e., to assign broker agent with new responsibilities of refining the capability description of individual agents, and conducting performance rating, based on the feedback and other collected information.

To simplify the problem, but without lose of generality, we make the following assumptions:

- All the agents (including the broker agent) in the system share a common domain ontology. Agents with different ontologies could work together through ontology translation and/or semantic resolution (For example, [29]).
- We assume a cooperative environment, in which all of the agents are cooperative.
- We consider security and privacy issues orthogonal to what we discuss here.

3.1 Basic Model of a Multi-agent System (MAS)

In our model of agent system, there are three types of agents: service provider agent, service consumer agent, and broker agent.

Service provider agent, or *service provider*, in short, is an agent that has certain capabilities of providing certain services, such as some piece of information or computing power. See also *service consumer*.

Service consumer agent, or *service consumer,* in short, is an agent that consumes the service(s) provided by other (service provider) agents. An agent can be a consumer of one service, and a provider of another service at the same time.

A broker agent, or *broker*, in short, is a middle agent that can recommend service providers (to the service consumers) based on the information it has. Generally service providers advertise their services to the broker agent, and service consumers can ask the broker agent what service providers can provide the services they need. In this work, broker agent has two more tasks to do: to refine the capability model of individual service providers, and to conduct performance rating based on the information it collects. However, an agent is not required to go through a broker agent to find a service provider.

Advertise refers to the process that an agent voluntarily tells other agents, e.g., the broker agent, about its capability information. It is similar to the “advertise” performative in KQML.

Recommend refers to the process that an agent (typically a broker agent) tells another agent what agents can perform certain tasks, usually in response to such a request.

Matching or *Matchmaking* refers to the process that the broker agent tries to match a recommendation request against its knowledge about the capabilities of the agents to find some agent(s) that can provide the service requested.

The term *recommendation request* and the term *matching request* are sometimes used interchangeably to refer to a request for recommendation of agents that can provide certain services.

3.2 The (Advisory) Brokering Protocol

To enable the broker with its new capabilities, the other agents as well as the broker agent itself need to follow an advisory brokering protocol, so that the individual agents can provide, and the broker agent can collect, useful information for refining the capability description and for conducting performance rating. The protocol is advisory because an agent without the knowledge about the protocol should at least be able to work with other agents in the system. However, it may not be able to take the full advantage of the broker's capability, as well as to contribute to the success of the broker agent. Note that the protocol described here is conceptual, in the sense that it may be implemented in various ways, e.g., as KQML performatives, or at service ontology level.

The protocol has the following (communicative) acts:

1. <advertise> If an agent wants other agents to know about its capabilities or services it can provide, it can send an <advertise> message (with its capability description) to the broker agent.
2. <request-for-recommendation> If an agent needs some services and wants to know who (what agents) can provide such services, it can send a <request-for-recommendation> message (with the service description) to the broker. The broker agent can respond with <recommend> if it can find some suitable service providers, or <sorry>, if it can't make an appropriate recommendation.
3. <recommend> The communicative act that the broker agent will use to recommend service providers (if any) in response to a <request-for-recommendation>
4. <follow-on-recommend> The broker agent (if it chooses to) may notify an agent about the availability of a new (better) service provider for a service that was previously requested.
5. <feedback> An agent can voluntarily, or in response to a request, send <feedback> to the broker agent about some previously recommended service providers.
6. <request-for-feedback> The broker agent can ask an agent how well a previously recommended agent works. The agents asked are encouraged to give a timely response.
7. <capability-query> An individual agent is expected to be able to answer queries such as “Can you do...?”, “what are your capabilities”, etc. The response to such a query can be a <advertise>, <confirm> or <disconfirm>.
8. <confirm> and <disconfirm> An individual agent can confirm or disconfirm about a capability query.

If an individual agent does not comply with the brokering protocol, it does not affect the others. As long as an agent observes the first three items, i.e., <advertise>, <request-for-recommendation>, and <recommend>, it can work with the broker, although it might not be able to take the full advantage of the broker's learning capability.

3.3 An Illustration

The refinement of an agent's capability model and the agent performance rating are based on the information the broker collected from the agents. The information may come from various channels, such as:

- The information voluntarily advertised by a service provider agent
- The feedback from a service consumer agent about some service providers
- The result of a capability query
- The local distribution of the agent services (will be explained later)
- Feedback/ratings (if any) provided by the human users
- The domain knowledge (e.g., the domain ontology)
- If the broker also performs task brokering[1], then the requests and the results are useful, too (there are some privacy/security issues, which are outside the scope of this work).

Now we use an example to illustrate the process. Consider selling televisions as a service with three sub-service classes: selling traditional TV, selling HD-ready TV, and selling HDTV. Suppose that agent A advertised that one of the services it can provide is selling TV. Then agent B requests the broker to recommend some agent that sells HDTV and agent C asks for a traditional TV seller. But unfortunately, nobody has advertised for HDTV or traditional TV so far. Suppose the best the broker can do is to recommend agent A to both B and C. Moment later, agent B came back said "No, I'm disappointed with A" (Feedback on HDTV, not TV). But C came back and said, "The service provided by A is great!". Given all that information, the broker figured that although agent A advertised for selling (all kinds of, by default) TVs, it looks like its strength is in selling traditional TV service, and its selling HDTV service may not be that good. To take advantage of the brokering protocol, the broker agent can ask A if it really sells HDTVs. If A confirms, it won't help much to raise its reputation, but if A disconfirms, the broker agent can then just disqualify it for any future HDTV requests. Why agent A wants to reply, then? There could be two reasons: one is to be cooperative, the other is to avoid bad reputations - claiming that you can do something you can't do would adversely affect your reputation. The broker agent can prove, disprove, or refine its belief about an agent through more feedback from other agents, and make use of this knowledge in the future matching processes.

From this example, we can see that the refinement is based on the advertised capability description, guided by the domain knowledge (e.g., selling HDTV is a sub-class of selling TV service), using the information collected from various channels. At the same time as the capability description is refined, the broker agent will also assign ratings on the various aspects (or properties) of the service provided by certain agents. Therefore, capability description refinement and performance rating are really two results of the same process, or alternatively, one may view the performance rating as part of the capability description model.

[1] Task brokering refers to the process that the broker receives a query, finds an appropriate agent, and forwards the query to that agent. When finished, the result is sent to the broker, and the broker forwards the result to the requested agent.

Now we use a similar example to illustrate how the information on local service distribution can help in achieving better matching results. It's certainly the best if we know the exact probabilistic distribution of the various agent services, but that is not something always available. By local service distribution, we mean the distribution of the services seen so far. Suppose that through the course of matchmaking service, the broker discovers the following: 85% of the advertisements/requests are about traditional TV, 8% are about HD-ready TV, and the rest (7%) are about HDTV. Suppose that the only advertisements on selling TVs are the one from agent B, which advertised “selling traditional TVs”, and the one from agent C, which advertised “selling HDTVs”. If agent A requests a recommendation on “selling TV” service, then with the knowledge of the local service distribution, the broker would be able to recommend the traditional TV seller (agent B) over the HDTV seller (agent C), assuming all the other factors equal. Five years later, the distribution of the three sub service classes might change to 30%, 20%, and 50% respectively. The broker agent will then be able to dynamically capture the changes in the probabilistic distribution of services, and makes appropriate recommendations accordingly.

On the other hand, while most of the TV sellers (those who advertise that they sell TVs) sell traditional TVs, not that many TV sellers sell HDTVs. So based on the probabilistic distribution, the broker agent would be more confident to recommend a TV seller if the request is about traditional TV, while it would be less confident (to recommend a TV seller) if the request is about HDTV. When computing the probabilistic distribution, we consider both how many sub classes a service class has, and the frequency of a service being referenced (requested/advertised).

In large, heterogeneous agent systems, while exact service matches are always desirable, it's not always possible to find exact matches. The approach introduced here should help achieve more accurate partial matching.

3.4 Agent Capability Refinement Model

Now, let's take a closer look at the agent capability refinement model. A service has a set of direct super (parent) services (empty if top-level service) and a set of direct sub (child) services (empty if leaf level service). Among other information in an agent's capability description, there is a set of ratable features or properties (when no ratable features are specified, then there is one default ratable feature, which is the capability description itself). An importance vector can optionally be defined to specify the (relative) importance of each ratable feature.

The refinement is performed along three lines: service specialization, service generalization, and service performance rating. Specialization and/or generalization will be performed in situations when no (exact) match could be found for a requested service, or some (exact) matches could be found but the ratings on these services are not good. If some parent service p of the requested service can be found, and the rating r of p is good, we may assume that the agent could perform the requested service with similar ratings as r - basically we specialize the service that the agent can perform. Or on the other hand, if an agent can be found that can perform most (if not all) of the direct sub services of the requested service, we could estimate how well the agent can perform the requested service based on ratings on the sub services. The assumption or estimation (or belief, in one word) that the agent can perform the requested service with a certain rating can be further refined based on future feedback or capability queries. Specialization and generalization, together with performance rating, give the broker agent deeper insight in what an agent can do, and how well it can do it.

Performance rating is orthogonal to specialization and generalization. In order to discuss how the performance rating will be conducted and updated, we first need to introduce our abstraction of the problem and the concept of weight function.

The Dynamic Weighted Sum System. Our abstraction of the problem is to compute the weighted-sum of a set of dynamically obtained data samples. "Weighted" because the capability of an agent may change over time (e.g., through learning) and therefore, data samples obtained at different times usually carry different significance. The number of data samples may increase as new samples are obtained. The weights for these data samples are modeled as a weight sequence, whose length increases as new data samples are obtained. The data samples can be of any type (scalar, vector, matrix, or whatever) as long as it meets certain requirements as discussed below. The goal here is to dynamically and systematically assign a weight for each new data sample, adjust the weight for the existing data samples (implicitly or explicitly), and then compute the new weighted-sum, preferably in an incremental way.

A **weight sequence** is a sequence W_n $\{w_1, w_2, w_3, w_4 \ldots w_n\}$ of n (n>0) real numbers w_i such that $0 <= w_i \leq 1$ for all $1 \leq i \leq n$ and that $\sum_{<i=1,\, n>}\{w_i\} = 1$ (the sum of w_1 to w_n)

In this context, a **weight function** is a function *f* that, given a natural number n, can generate a weight sequence $\{f(n, 1), f(n, 2) \ldots f(n, n)\}$ of length n. Note that a weight sequence or a weight function is independent of any data sets, although it could be associated with a data set. A weight function is said to be **incremental** if, when a new data sample is obtained and a new weight sequence is generated for the increased data set, the new weighted sum can be computed incrementally based on the current weighted sum.

Why is this property of incrementality so important? First, you don't have to keep all the data samples around. Second, in contexts like broker learning, information about individual agents is obtained dynamically, and that information should be taken into account as soon as possible. If everything is recalculated from all the data obtained since the very start, the cost may be prohibitively high.

In order to study the dynamic properties of some weight functions, we need to define the concept of a dynamic weighted sum system.

Definition 1:
A 4-tuple (*f*, X, +, *) is said to be a **DWS (Dynamic Weighted Sum) System** if:

- ➢ *f* is a weight function with range $d \subset R$ and $d = [0, 1]$. (R is the set of real numbers)
- ➢ X is a set of sample data (whose size may increase over time)
- ➢ + is the addition operator
- ➢ is the multiplication operator

Such that, with $x, y, z \in X$, and $a, b, c \in d$, the following hold:

- ➢ + and * on d are just the same as what/how they are defined on R.
- ➢ Closure: $x + y \in X$
- ➢ Associative law: $(x + y) + z = x + (y + z) = x + y + z$
- ➢ Distributive law: $a * (x + y) = a * x + a * y$, $(a + b) * x = a * x + b * x$.
- ➢ Commutative law: $a * x = x * a$

Theorem 1:
Suppose *f* is the weight function of a DWS system (*f*, X, +, *). If for any given n (n>0) and i (1< i <= n),

$$f(n+1, i)/f(n, i) = c(n)$$

holds, where c(n) is a function of n, *f*(m, j) is the jth element in the (generated) weight sequence of length m, then *f* is incremental in the framework of the DWS system.

Proof:
Suppose the current size of X is n. Let $\{w_1, w_2, w_3, \ldots, w_n\}$ be the weight sequence generated by *f*(n, i), and $\{w'_1, w'_2, w'_3, \ldots, w'_n, w'_{n+1}\}$ be the new weight sequence generated by *f*(n+1, i) when the new data sample x_{n+1} is acquired. Let S_n be the weighted sum, we have:

$$S_n = \sum_{<i=1,\, n>}\{x_i{*}w_i\}.$$

With the latest acquired data sample x_{n+1}, let S_{n+1} be the new weighted sum, we have:

$$\begin{aligned} S_{n+1} &= \sum_{<i=1,\, n+1>}\{x_i{*}w'_i\} \\ &= \sum_{<i=1,\, n>}\{x_i{*}w'_i\} + x_{n+1}{*}w'_{n+1} \\ &= \sum_{<i=1,\, n>}\{x_i{*}w_i{*}c(n)\} + x_{n+1}{*}w'_{n+1} \\ &= c(n) * \sum_{<i=1,\, n>}\{x_i{*}w_i{*}\} + x_{n+1}{*}w'_{n+1} \\ &= c(n) * S_n + x_{n+1}{*}w'_{n+1} \end{aligned}$$

Therefore, the result S_{n+1} can be computed from S_n, so *f* is incremental.

End of proof.

Corollary 1:
Suppose *f*(n, i) is the weight function of a DWS system (*f*, X, +, *). For any given n = N, and any 1< i <= N, if *f*(N+1, 1)/*f*(N, i) = c holds, where c is a constant, then *f* is incremental.

Proof:
It follows directly from theorem 1.

So, how to create/find a weight function that is incremental? As a first step, let's find out how we can find weight functions that can generate weight sequences of any given length. One solution is to construct weight functions from other mathematical sequence functions, e.g., the natural sequence function (a function that generates the natural sequence {1, 2, 3…}). Let b(n, i) be a positive sequence function for sequence {b(n,1), b(n,2) … b(n,n)} of length n, where 1 <= i <= n, and b(n, i)>= 0. Let T(n) =

$\sum_{<i=1, n>}\{b(n, i)\}$ be the sum of the sequence b(n, i). Now we construct a new sequence function *f*(n, i) of length n as follow:

$$f(n, i) = b(n, i)/T(n),\ 1 <= i <= n, \text{ and } x/0 = 0 \text{ for any number } x.$$

It's not difficult to show that 0 <= *f*(n, i) <= 1, and that $\sum_{<i=1, n>}\{f(n, i)\} = 1$. Therefore, *f*(n, i) is a weight function. In this context, b is called the **base function**; T is called the **sum function** of b; and *f* is the **constructed weight function** with base function b.

Theorem 2:
For a positive sequence function b(x, i), if b(n+1, i) = b(n, i) holds for any given n (n>0) and any i (1<=i<=n), then the constructed weight function *f*(x, i) with base function b is incremental in any DWS system (*f*, X, +, *).

Proof:

$$\begin{aligned} f(n+1, i)/f(n, i) &= (b(n+1, i)/T(n+1)) / (b(n, i)/T(n)) \\ &= (b(n+1, i)/ (b(n, i)) * (T(n) /T(n+1)) \\ &= T(n) /T(n+1) \end{aligned}$$

The last step above is because b(n+1, i) = b(n, i). From theorem 1 we know that the weight function *f*(n, i) is incremental.

End of proof.

An Example Weight Function. Evidently, theorem 2 can lead to the discovery of some weight functions that are incremental in the framework of a DWS system. As an example, let's look at the weight function constructed from the geometric sequence function. The geometric sequence function (of length n) can be written as follow:

$$b(n, i) = r^{i-1}, \text{ where } 1 <= i <= n, \text{ and r is called the “ratio term”}.$$

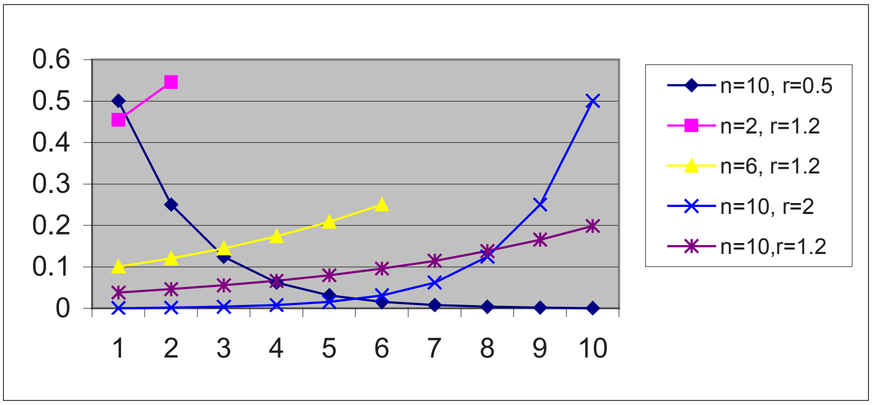

Fig. 1. Geometric Sequence Based Sequence Functions

When $|r| \neq 1$, the sum function is:

$$T(n) = (1 - r^n)/(1-r)$$

Therefore, the weight function with base function b is:

$$f(n, i) = b(n, i)/T(n) = r^{i-1}/((1- r^n)/(1-r)) = (r^i - r^{i-1})/(r^n - 1)$$

Since $b(n+1, i) = b(n, i) = r^{i-1}$, it follows from theorem 2 that this geometric sequence based weight function is incremental. Figure 1 shows a family of weight functions with different length "n" and ratio "r". Here are some more observations:

1) For two geometric sequences with ratio r1 and r2. If $r1 = 1/r2$, their corresponding weight functions are symmetric along $x = n/2$, that is, $f_{r1}(n, i) = f_{r2}(n, n-i+1)$.
2) The sum function T(n) converges to $1/(1-r)$ when $0 <= r < 1$. Therefore, the minimum possible value for $f(n, 1)$ is $1-r$, and is therefore independent of n.
3) When $0 <= r < 1$, the weights decrease monotonically. When r is small, the weights of the first few examples decrease sharply and then turn almost flat. Moreover, the first few weights totaled almost to 1.0. But when r is closer to 1.0, the curve becomes quite flat.
4) Given (1), the conclusions in (2) and (3) are also applicable to the case of $r > 1$, but need to be mirrored along n/2.

Rating Computation. Finally, we can discuss the computing of the performance ratings. A ratable feature of a service is the smallest unit for rating. The rating of a service is therefore a vector of ratings for its ratable features.

Suppose the current rating on some service (of some agent) x is R(n), which is the combined effect from the previous n ratings. We just received a rating p on x and we want to compute the new rating R(n+1) as the combined result of the n+1 ratings received so far. If the weight function f is incremental, we can compute R(n+1) as follow:

$$R(n+1) = c(n)*R(n) + p*f(n+1, n+1),$$

where c(n) is some function of n, and is decided by f. On the other hand, if f is not incremental, we need to re-compute the whole thing:

$$R(n+1) = \sum_{<i=1, n>}\{p_i*f(n+1,i)\} + p*f(n+1, n+1),$$

where p_i is the ith rating received. If the overall rating on a service is needed, it can be computed from the feature ratings, optionally weighted by an importance vector (for features), that is,

$$r = R' * I,$$

where R' is the transposed rating vector, and I is the importance vector.

Rating Propagation. But things won't stop there. When the rating(s) on a service is updated, the changes propagate up as well as down the service hierarchy. The degree of effect on the services up and/or down the hierarchy is governed by the local service distribution, which is captured dynamically. If the service whose rating is to be updated through propagation (the target service) has common ratable features with the propagation origin service, then the set of common features will be updated. Otherwise, the target service itself is considered to be the only feature. Here we only discuss the case of tree-shaped hierarchy. In more general hierarchies like the DAG hierarchy, propagation can performed in a similar fashion.

Upward propagation refers to the process that when the rating on a service x is updated, the rating on its parent service p will also be updated. This update propagates upward along the service hierarchy, to the most general service a specific agent has to offer. Suppose x accounts for α percent of p. Let R_p and R'_p be the ratings of p before and after the update, respectively; let R'_x be the rating of x after the update. We have the upward propagation rule:

$$R'_p = (R_p + \alpha * R'_x)/(1 + \alpha)$$

Therefore, the more percentage x is of p, the more p will be affected.

Downward propagation refers to the process that when the rating on a service x is updated, the ratings on its child services will be updated accordingly. The update propagates along the service hierarchy all the way down to the most specific service the agent has to offer. Suppose c is one of the child services of x and accounts for β percent of x. Let R_c and R'_c be the ratings of c before and after the update, respectively. Then we have the downward propagation rule:

$$R'_c = (R_c + \beta * R'_x)/(1 + \beta)$$

Therefore, the more percentage c is of x, the more c will be affected.

In general (for both downward and upward propagation), the further a propagation target is from the origin, the less it will be affected by the propagation. How much a propagation target will be affected depends on the local distribution of the services.

4 Discussions

This paper presents a framework for an adaptive service broker that learns and refines a model of a service provider's performance. The system design and implementation are on the way. However, significant additional issues still remain. One issue that will need to be addressed is the situation when the child services of a service do not strictly disjoint with one another. This might be addressed using the local service distribution information. Next is the fairness issue. Although we believe that in general the broker agent can improve the quality of service matching through learning, the ratings on specific services may not always be "accurate". Over or under estimate the capability of any agent is unfair to that agent, and is unfair to the other agents as well. The problem might be lessened if the "bad" agents were given some chances - but that might compromise the quality of the matching service. We will also explore other formal methods for rating update and propagation. We believe that the security issue and the privacy issue are orthogonal to what we've discussed here.

One of the ideas behind this work is the law of locality. The more frequently a subset of the agents' capability is referenced (e.g., asked for recommendation), the more likely this sub-set will be referenced again later. The good news is, the more frequently the subset is referenced, the more likely the detailed information on this subset will be obtained through learning. Therefore, the approach captures the temporal locality. Moreover, the (spatial) distribution of the agent services is dynamically captured and used in computing agent service ratings.

Although we choose the DAML framework, we think the approach proposed here should work with other languages/frameworks, too. For example, LARKS can be extended with a set of (domain-dependent) rating slots and a new type of filer(s) that can be used to handle the new slots. Then what we discussed here should apply.

References

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

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

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

[4] Harry Chen, Anupam Joshi, Tim Finin. Dynamic Service Discovery for Mobile Computing: Intelligent Agents Meet Jini in the Aether. *The Baltzer Science Journal on Cluster Computing*. March 2001 (Volume 3, No. 2).

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

[6] DAML specification, http://www.daml.org/, October 2000.

[7] http://www.daml.org/2001/03/reference.html.

[8] http://www.daml.org/services/.

[9] DAML-S: A DAML for Web Services, White paper, SRI, http://www.ai.sri.com/daml/services/daml-s.pdf.

[10] Decker, K, and Sycara, K and 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.

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

[12] Dellarocas C, , 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.

[13] Dennis D. Berkey, Paul Blanchard. Calculus, Third edition, (Instructor's Preliminary Edition) Saunders College Publishing, 1992.

[14] FIPA 97 Specification Part 1, Agent Management.

[15] FIPA 97 Specification Part 2, Agent Communication Language.

[16] Friedrich, H., Kaiser, M., Rogalla, O. and Dillmann, R., 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.

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

[18] Gruber, T. R., 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.

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

[20] Yannis Labrou, Tim Finin, Benjamin Grosof and Yun Peng, Agent Communication Languages, in *Handbook of Agent Technology*, Jeff Bradshaw, ed., MIT/AAAI Press, 2001.

[21] 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.

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

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

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

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

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

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

[28] Nwana, Hyacinth S., Software Agents: An Overview. *Knowledge Engineering Review*, Vol. 11, No 3, pp.1-40, Sept 1996, Cambridge University Press, 1996.

[29] 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.

[30] Smith, Reid G., 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.

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

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

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

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

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

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

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