Quantitative Agent Service Matching

Xiaocheng Luan, Yun Peng, and Timothy Finin
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250, USA
{xluan1, ypeng, finin}@csee.umbc.edu

Abstract. The ultimate goal of service matching is to find the service provider(s) that can perform tasks of given description with the best overall degree of satisfaction. However, service description matching solves only part of the problem. Agents that match a given service request description may vary greatly in the level and/or quality of services that they can perform and an agent may have strong and weak areas in its advertised service space. In this work, we take a quantitative approach in which the broker agent (match maker) considers performance rating an integral part of an agent's capability model, captures an agent's strong and weak areas through the interaction with the agents, and refines the agent's capability model with the information gathered. The broker agent also dynamically builds a service distribution model that can provide vital information for determining the degree of match in cases of non-exact matches. An experimental system has been designed and implemented using OWL-S as the upper service ontology and the result statistics show significant advantage over other major levels of brokers.

1. Introduction

As the software agent technology quietly infiltrates into the computer software industry and the increasing demand for a semantic web, service matching is playing an increasingly important role. With OWL becoming a W3C recommendation, this trend will continue in an even faster pace. This research is an effort to address some of the service matching issues.

There has been intensive research in the area of service matching in the past years. However, there are still issues that have not been adequately addressed. Service matching is usually characterized merely as matching the requested service description against that of the advertised services to find the (best) matches. While this characterization is correct by itself, it directly lends itself to two issues: (1) Are these agents that advertise services with identical descriptions created equal? (2) Are these advertised service descriptions accurate with respect to their true capabilities, even if the agents are "honest"?

We know too well that agents are not created equal, even if they advertise the same services. There bound to be good performers and bad performers, and it would not be unusual for an agent to have strong and weak areas. When an agent posts a service advertisement, it is advertising a family of services in the service space as defined by the advertised service description. For example, a travel service advertised by a travel agency may cover services from international and domestic flight ticket reservation, rental car reservation, to hotel reservation, etc. Agents that advertise travel services may or may not actually offer all the services in the travel service space and each agent may have its strong and weak areas within the space(s) of its offerings. Finding out exactly what an agent has to offer or finding out its strong and weak areas is therefore just as important as telling good performers from bad performers.

The distribution of the services within a service domain is an important factor in estimating the overall strength of an agent's capability with respect to a specific service description, For example, it might be that 60% of the travel services are in the area of airline ticket reservation, 30% in the area of rental car reservation, and only 10% is about hotel reservation. For overall

rating on travel services, an agency doing well only in the airline ticket reservation sector will likely get a better rating than an agency that does well only in the hotel reservation sector, since it can satisfy 60% of the services requested.

To address these issues, this work takes a quantitative approach to agent service matching as well as to the agent capability modeling and refinement. A service provider agent's capability model is mainly built based on the service consumer agents' experiences and the knowledge from the domain ontology. Service matching is a two phase process. In the first phase, logical matching is performed to find out the service providers whose advertised services match the requested service description. In the second phase, each of the matches from the first phase is scored based on the service provider agent's capability model and the service distribution. The resulting score is not a overall of that service provider agent, but an overall score of the agent with respect to the specific requested service description. In other words, the final score is the probability that the service provider agent can successfully perform tasks with the given requested service description.

Some agents might be smart enough to learn about the other agent's capabilities but it is a heavy burden for individual agents and it may not be very effective since an individual agent's experience is usually limited. In the real world, we deal with these situations by reading consumer reports and by asking about other people's experiences. The consumer reports may incorporate the experiences of thousands of consumers and testing results, it can therefore help consumers make the right choices. We believe similar approaches should work for the agent world, too. A framework can be established in which service consumer agents can voice their opinions on the services they received and then the agents' capability model can be built and refined based on the feedback information and the knowledge from the domain ontology. The broker agent is in an especially "convenient" position to take this responsibility. The broker agent takes service advertisements as well as service match requests and therefore knows the service demands and the service offerings. Broker 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. Capturing and modeling agents' capabilities is consistent with the broker's ultimate goal, that is, to recommend the right service provider(s) to the service consumer agents. Moreover, since the broker agent recommends the service providers, it is legitimate and proper for the broker agent to receive feedback from the service consumer agents, or for the broker agents to check with the consumer agents, "how is my recommendation?"

In the rest of this article, we will first briefly discuss some of the related works, and then introduce the agent capability modeling and the service matching algorithms, after which we will discuss the experimental implementation, analyze the statistical results, and draw a conclusion.

2. A Brief Background and Related Work

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. The service matching process has evolved from the early stage, simple KQML-performative based service matching to the more advanced syntactic and semantic based matching. With the OWL becoming a W3C recommendation and the progress in OWL-S (DAML-S), more and more research on service matching are shifting (or adapting) to the OWL/OWL-S framework.

OWL, Web Ontology Language, is "a language for defining structured, Web-based ontologies which enable richer integration and interoperability of data across application boundaries" ([35]). It is machine-readable and supports incremental and distributed definition of ontologies. OWL became a W3C recommendation in February 2004. OWL-S "is an OWL-based Web service ontology, which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous,

computer-interpretable form" Under this framework, a service description has three key components: a service profile that tells "what the service does", a service model that describes "how the service works", and a service grounding that specifies details on how to access the service.

In [30], the authors discussed desiderata and algorithms for service matching in the DAML-S (the predecessor of OWL-S) framework. The basic idea is that an advertised service profile matches a requested service profile if (and only if) for each input parameter of the advertised profile there is a matching input parameter in the requested profile; and that for each output parameter in the requested profile there is a matching output parameter from the advertised profile. At parameter level, there are three levels of matches based on the relationship between the types of the pair of matching parameters: exact match, plugIn match, and subsumes match.

Many (if not most) of the research on reputation management is in the context of electronic marketplaces. One of the reputation mechanisms described in [39] 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 described in [26] is very interesting, too, but we do not have the space to introduce it in more details.

In comparison to the existing work, we consider the "trust" issue as orthogonal to our work and focus on the agent capability modeling. We consider performance as an integral part of an agent's capability model as well as a key factor in service matching

3. The Agent Capability Model and Service Matching

This work is independent of any upper service ontology but we will use the OWL-S (DAML-S) framework for the purpose of discussion and experimental implementation. Under this framework, a service is described by a service profile (or profile, in short), the major components of which are the input parameters, the output parameters, the preconditions, and the effects (post conditions). We think that at least some of the preconditions and some of the conditions in the conditional outputs and conditional effects are outside the scope of service matching and therefore we will focus on the input and output parameters.

Here is an example profile description 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", which is the type the parameter takes. It's the main factor in matching parameters of two service descriptions.

The general interaction is as follow: the service provider agents advertise the services that they can (and intend to) offer; a service consumer agent requests the broker agent to recommend (service provider) agent(s) that can perform services with the given description. The broker agent matches the requests against the advertisements it has received and recommends the ones that it thinks would best meet the requests. A service consumer agent may optionally provide feedback to the broker agent on the service rendered, which may include information like the exact description of the service performed (may or may not be the same as that in the service matching request or in the service advertisement) and the satisfaction rating.

To simplify the discussion, we define a few terms. A class (or type) c2 *satisfies* another class (or type) c1 if and only if c2 $\supseteq$ c1 (that is, c2 = c1 or c2 is a super-class/ super-type of c1). An input parameter p2 (of one service description) *satisfies* a (corresponding) input parameter p1 (of another service description) if and only if the type of p1 satisfies the type of p2. An output parameter q2 (of one service description) *satisfies* a (corresponding) output parameter q1 (of another service description) if and only if the type of q2 satisfies the type of q1. A **service space** *S* is defined by 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 s falls in the service space *S* (P, I O) if and only if (1) the profile type (class) of s is satisfied by P; (2) for each output parameter p of s, there is an output parameter p' in O such that p' satisfies p; and (3) for each input parameter q in I, there is an input parameter q' of s such that q' satisfies q. 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 description.

A match is an exact match if the requested service space is exactly the same as the advertised service space, in other words, any service in the requested service space is also in the advertised service space, and vice versa; a match is a satisfied match if any service that falls in the requested service space also falls in the advertised service space, but not the other way around; a match is a subsumes match if any service that falls in the advertised service space also falls in the requested service space, but not the other way around.

3.1 The agent capability model

The agent capability model has two major components: a service distribution model and a set of rating entries. The domain ontology provides the basis and the concept hierarchy for the construction and update of both components. The service distribution component is not specific to any agent or advertised service, but rather it is common to all the agents within the service domain. The service distribution model is constructed and updated through the interaction with the agents in the domain. Service distribution model has an important role to play in the computation of an agent's performance rating during the service matching as well as in the rating entry management.

For each advertised service, there is a set of rating entries that are constructed and updated with the service feedback and/or other evaluation information such as settings specified by a human administrator. In other words, a set of rating entries is used to organize the evaluation information (e.g., feedback) of an advertised service and to characterize the service provider with respect to this advertised service. Each rating entry has a description that defines the service space

of the entry. The basic rule is that an evaluation (e.g., feedback) will be applied to a rating entry if the entry's service space is the most specific of all the entry service spaces of the advertised service that the evaluation is about. Therefore, a rating entry reflects the cumulative satisfaction rating on the agent with respect to the entry's description. For example, for a travel service advertised by some agent, it may have rating entries like flight ticket reservation entry, international travel service entry, etc.

Initially, a new rating entry may be created each time a feedback with a unique description is received (even though it may be applicable to some existing entries). When the number of entries exceeds a certain limit, a group of more specific entries may be combined into a more general rating entry. The service distribution is a big factor here. More entries should be dedicated to areas in which more service instances are realized to maximize the accuracy for the most frequently used subset. Similarly, more entries should be dedicated to areas in which the agent's performance varies the most so as to capture the subtleties. When a feedback is applied to a rating entry, the rating of the rating entry will need to be updated. Algorithms like EWMA (Exponentially Weighted Moving Average) may be used to recalculate the new rating for an entry but specific algorithms may be chosen to suit the need of a specific domain. As a result, an agent's capability model is refined when a new rating entry is created, or when a rating entry is updated. The performance rating of an agent with respect to a specific given description (which may or may not be identical to the description advertised) can then be computed based on the rating entries. Since service distribution is a factor too, an agent's performance rating may change even when the rating entries remain the same. More details will be described in the service matching section.

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 performance with respect to the specific service description 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.

3.2 Service matching

In this work, the service matching process has two phases. The first phase performs logical match to find all the candidates, that is, all the advertised services whose service descriptions logically match the requested service description are considered as candidates and will participate in the second phase of matching. The second phase attempts to quantitatively evaluate the candidates and picks the candidates that have higher probabilities of success. It's this second phase that is the focus of this work.

3.2.1 Logical match

This phase deals with the issue of whether two service descriptions match. It basically compares the two given service descriptions, an advertised service description (or adv, in short) and a requested service description (or req, in short), to see if the adv matches the req. There is a match if (1) the adv profile type matches the req profile type; (2) for each input parameter in adv, there is a matching input parameter in the req; and (3) for each output parameter in req, there is a matching output parameter in the adv. A parameter can be used at most once in the matching. Why each input parameter in the adv has to be matched by one in the req? Because that is what the service provider requires (or needs) in order to perform the task. Similarly, each output parameter in the req must be matched by one in the adv because that is what the service consumer wants to be accomplished.

Two remaining issues are how to match (or pair) the parameters in a given pair of adv and req descriptions, and what is considered a match (between two parameters). The second issue is simple. There are four potential levels of parameter matches, namely exact match, satisfied match, subsumes match, and intersection match. With respect to whether and how a parameter p2 (of an adv) matches another parameter p1 (of a req):

Exact match if p2.type = p1.type; otherwise,
Satisfied mach if p2.type $\supset$ p1.type; otherwise,
Subsumes match if p2.type $\subset$ p1.type; otherwise,
Intersection match if p2.type $\cap$ p1.type $\neq \phi$.

These are just potential levels of matches since some may prefer to disable subsumes match and intersection match, which are just partial matches.

The first issue, that is, how to match (or pair) the parameters in a given pair of adv and req, requires a closer look. As discussed earlier, the main factor in parameter matching is the parameter type. But in a service description multiple parameters may have the same type. For example, in the flight ticket service, there is a departure airport and an arrival airport, both of which may have type Airport. It's tempting to match parameter names, e.g., departureAirport matches departureAirport, and arrivalAirport matches arrivalAirport. Given the distributed nature of OWL, some parameters of a service profile may not have been explicitly defined in the common domain ontology. It is possible to require that all the parameters of a profile be explicitly defined but we think that requirement is too strong, especially in large, open agent systems or other similar Internet based services. Our approach to parameter pairing is a two-step one.

In the first step, we pair the parameters (from adv and req) that match both in terms of their types and in terms of their names. 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 match the (remaining) parameters. As we shall see below that the parameter pairing problem can be transformed into a bipartite matching problem.

In graph theory, a bipartite graph is an undirected graph in which each of its vertices falls in one of two sets, the left set L, or the right set R. There can only be edges between vertices of different set, and each edge may be assigned a weight. The maximum cardinality maximum weight bipartite matching problem is about finding the maximum matching M that has the maximum weight sum. We will not get into too much detail here and it suffices to say that there are (tractable) algorithms that can find the maximum cardinality maximum weight matching. Now let us see how to transform the parameter pairing problem into the bipartite matching problem. Let the parameters from the adv form the node set L and the parameters from req form the node set R. Then draw an edge between each pair of parameters whose types match (a parameter may be used more than once at this point). We can assign weights to the edges (potential matches) such that exact matches are favored over satisfied matches, satisfied matches favored over subsumes matches, etc. Then we can use the maximum cardinality maximum weight bipartite matching algorithms to find the maximum matching with maximum weight. If every parameter that needs to be matched is matched, we have a match. Otherwise, the adv does not match the req.

3.2.2 Score the matches

Now we have all (may not need to find all, if too many) the matches (candidates), we need to estimate the probabilities that they will succeed in performing the requested services. We think in many (if not most) cases finding the best match is of paramount importance. As discussed earlier, the broker builds agent capability models in the form of rating entries and service distribution

model, based on which we can estimate for each candidate match the capability of the agent (that advertised the service) with respect to the specific request.

In choosing algorithms for estimating the service provider's performance based on the set of rating entries, we would like to see the algorithms to (at least) have the following properties: (1) entries that are "closer" to the request description should have more influences; (2) the overall rating must be smaller than the largest entry rating and must be larger than the smallest entry rating; (3) with the addition of an entry with rating larger than the current overall rating, the new overall rating should not decrease and it should increase if the relevancy of the entry is not zero; the effect should be reversed when such an entry is removed; and (4) with the addition of an entry with rating smaller than the current overall rating, the new overall rating should not increase and it should decrease if the relevancy of the entry is not zero; the effect should be reversed when such an entry is removed. One of the simplest algorithms with these properties (proof omitted due to space constraint) is the weighted sum algorithm and we will use a variant of the weighted sum algorithm. Here the weight is the relative "relevancy" (or degree of match) of an entry towards the requested service description. That is, a rating entry whose description matches better with the requested service description will have more influence in estimating the provider's performance with respect to the requested service description. The overall rating R_n of an agent (with respect to the request) of n rating entries is given by:

$$R_n = p_{max} \sum_{i=0}^{n} (r_i * (p_i / \sum_{i=0}^{n} p_i))$$

Where r_i is the rating value of the i^{th} entry; p_i is the relevancy of the i^{th} entry (the degree of match between the entry and the requested service); and p_{max} is the maximum relevancy among all the entries (for the advertised service).

Now the key issue becomes how to calculate the relevancy, that is, the degree that a rating entry description matches the requested service description. Since each service description defines a service space, the degree that an entry matches the req is therefore the probability that an instance in the requested service space falls in the service space defined by the rating entry:

$$P(s \in \text{entry\_space} \mid s \in \text{req\_space}) = |(\text{entry\_space} \cap \text{req\_space})|/|\text{req\_space}|$$

Where entry_space is the service space defined by a rating entry; req_space is the requested service space. A pair of vertical bars gets the cardinality of the enclosed set.

In the case when a rating entry is more general than the request, that is, when entry_space $\supset$ req_space, the degree of relevancy should be 100% according to this probability model. However, some discount factor may be applied in cases like this (when appropriate) to reflect the "common sense" that a provider of a general service may not be as good as more specialized providers in the specific areas.

4. Experimental Evaluation

In this experimental implementation, OWL is chosen as the representation language for service descriptions and OWL-S is chosen as the upper service ontology. We extended OWL-S with constructs for supporting service feedback and we defined a simple domain ontology for flight ticket reservation service. This prototype is implemented in Java and the OWL inference engine we used is a variant of DAMLJessKB. DAMLJessKB is a DAML inference engine. 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 semantic; 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.

DAMLJessKB is a great tool but to better support our implementation, we rewrote DAMLJessKB into **daml4jess**, which was then slightly modified to support basic OWL inference. As discussed in the previous sections, the service matching has two phases. The logical matching is performed in the Jess engine while the match scoring modules are mainly implemented in Java. The agent capability modeling modules are mainly implemented on the Java side, too. The overall design is illustrated in figure 1.

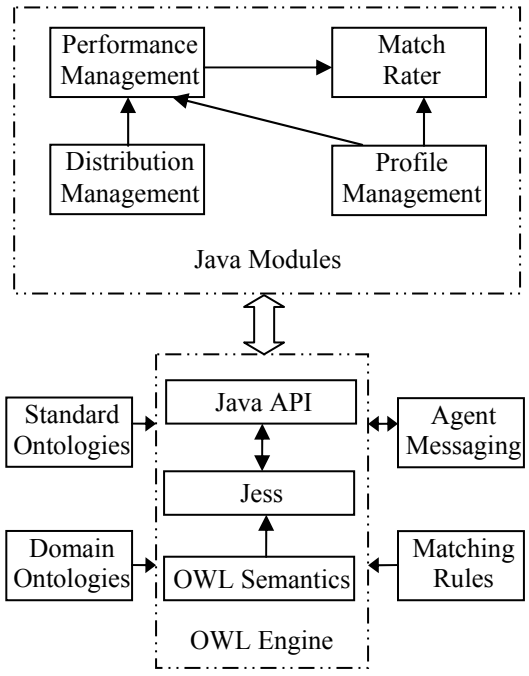

Figure 1 Prototype Design

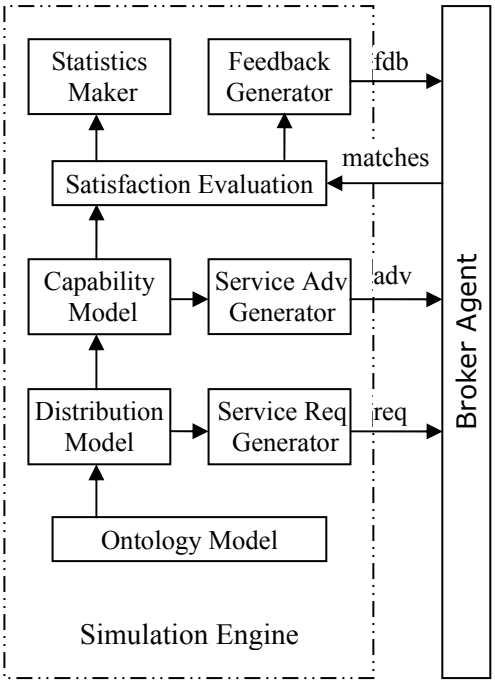

Figure 2 Evaluation framework

4.1 An Evaluation Framework

An evaluation framework has been designed and implemented to simulate the agent interactions so as to evaluate the ideas and the algorithms discussed in this work. As illustrated in figure 2, the simulator builds a service distribution model based on which the service advertisements, the service matching requests, and the capability model of the service provider agents are generated. Please note that the distribution model and the agent capability model generated here are not available to the broker agent. They are used by the simulator for generating service sequences as well as for generating satisfaction ratings for the matches recommended (by the broker agent). The advertisements and requests are sent to the broker agent and recommended matches (if any) will be sent back. The simulator evaluates the matches and assigns satisfaction ratings (based on the provider's capability model) to the services chosen, and then a feedback will (optionally) be generated and sent back to the broker. Through the interactions, the broker agent attempts to capture the service distribution and build the capability model for the service provider agents based on which more accurate service matching can be achieved.

To evaluate the performance of the broker agent, that is, the algorithms discussed in this work, we defined and implemented four types of broker agents (just for the purpose of this evaluation), types 1 – 4. **Type-1 broker** is a basic broker that performs logical match only to check if an advertised service matches a requested service description. If it is a match, it does not say how

well (or of what degree) the match is. **Type-2 broker** performs logical match only but it tells you the level of the match, e.g., exact match, plugIn match, subsumes match, etc, as described in [30]. **Type-3 broker** is our broker. Besides performing logical match, it considers performance an integral part of an agent's capability model. It learns through the interaction with the agents and builds the capability models for the service provider agents. **Type-4 broker** is an ideal broker that "knows" exactly which service provider agent will perform the best with regard to the given service request.

4.2 The Result Statistics

To collect the data for analysis, we "advertised" 50 services to the broker agent in the area of flight ticket reservation service, and then 1000 service matching requests were made (to the broker agent). The same procedures were applied to each of the four types of broker agents both for the case when subsumes match is turned off and for the case when subsumes match is turned on. When subsumes match is turned off, a match at subsumes level will not be considered a match. The intersection match level is not implemented here because it is lack of basis for comparison and it is of limited interest.

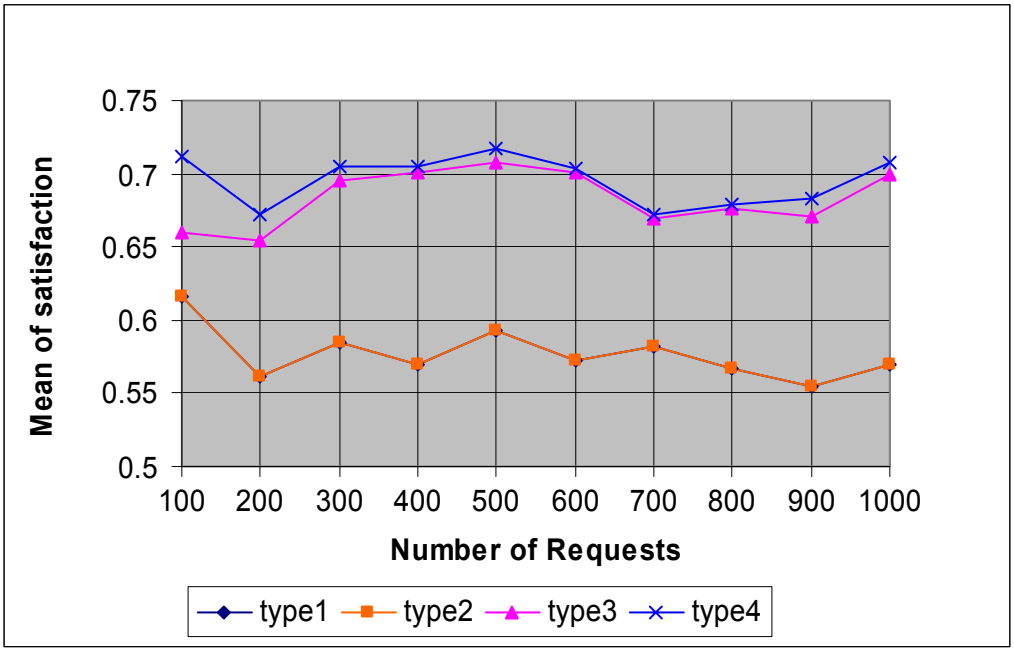

Figure 3 Comparison of broker performance with subsumes match turned off

Figure 3 shows the comparison of the four different types of brokers on the mean of satisfaction ratings. In this trial, the subsumes-match is turned off, that is, only exact matches and satisfied matches are qualified as matches. Under this condition, the type-1 broker and the type-2 broker perform exactly the same (the curves overlap), which is not a surprise. Just like a type-1 basic level broker, a type-2 broker does not try to tell the difference between the exact matches and the satisfied matches. In the first 100 requests, our broker (type-3) performs a bit better than the type-1 and type-2 brokers but it is about 10% below the ideal broker (type-4), which is significant. In the second and the third 100 requests, our broker appears to have learned quite a bit about the agents' capabilities and its performance is catching up with the ideal level broker. Starting from the fourth 100 requests, our broker is almost as good as the ideal level broker and it performs better than the type-1/type-2 brokers by at least 10-15%.

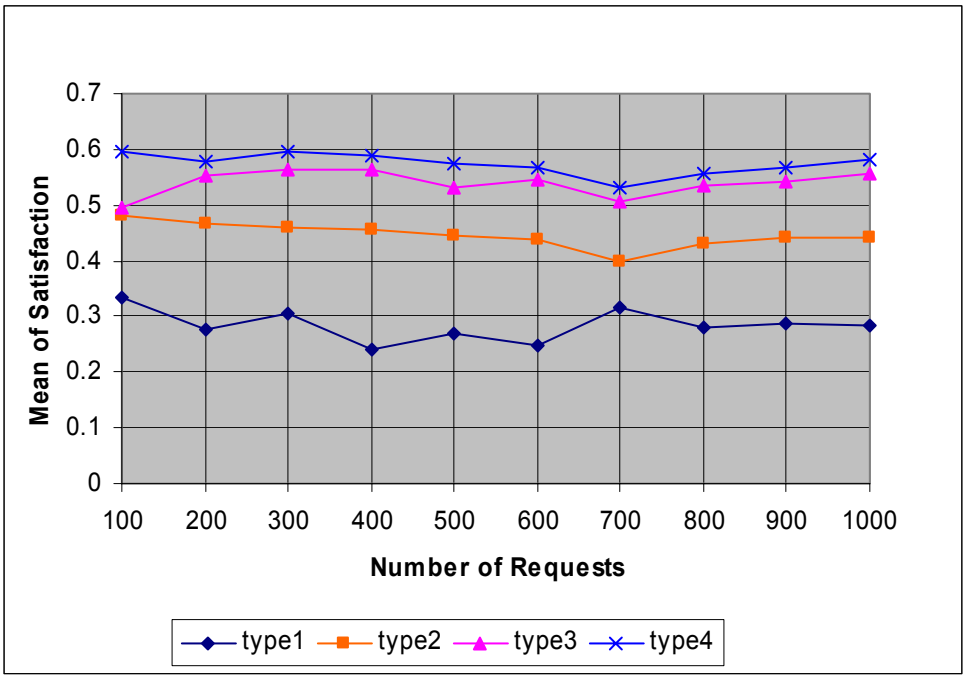

Figure 4 Comparison of broker performance with subsumes match turned on

Figure 4 shows the case when subsumes match is turned on, that is, a subsumes-level match will be considered a match. The type-1 broker performs far worse than the other three types of brokers because it does not tell the differences among different levels of matches. Our type-3 broker performs similarly as the type-2 broker in the first 100 requests. However, after 300 requests, our broker performs at least 10-15% better than the type-2 broker. Moreover, the performance of type-3 broker is very close to the type-4 broker, the ideal broker.

With the subsumes-level match turned off, 672 requests (out of the 1000 requests) are fulfilled, that is, have matches recommended by the broker agents. With the subsumes-level match turned on, 987 requests (out of 1000) are fulfilled. Please also note that the absolute numbers in the charts are not meaningful because the "intrinsic" agent capability ratings are generated randomly. It is the relative performance with respect to each other (especially with respect to the ideal broker) that matters.

5. A Summary and Discussions

In summary, we think that performance should be an integral part of an agent's capability model in addition to the descriptions of the services that it can provide. An agent would usually have strong and weak areas in its advertised service spaces and the capture of such strong and weak areas would be just as important as telling good performers from bad performers. The adoption of a formally defined, machine readable domain ontology would help realize these goals. Moreover, such a domain ontology would enable the broker agent to capture the service distribution, which may help determine the degree of match. With all these factors considered, it is no longer accurate to characterize "service matching" as matching logical descriptions of the service profiles, but rather, service

matching should be characterized as a process to find out the service provider agent(s) that have the best probability of success in performing tasks of given descriptions.

We think the experimental results demonstrate that the quantitative approach discussed in this work enables the broker agent to establish the capability model of the service provider agents and can significantly improve the quality of the service matching of the broker agent.

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. 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
5. daml4jess, http://www.csee.umbc.edu/~xluan1/daml4jess.
6. DAMLJessKB, http://edge.mcs.drexel.edu/assemblies/software/damljesskb/damljesskb.html.
7. DAML specification, http://www.daml.org/, October 2000.
8. daml.org, OWS-1.0 (document set), http://www.daml.org/services/owl-s/1.0/
9. http://www.daml.org/2001/03/reference.html.
10. http://www.daml.org/services/
11. DAML-S: A DAML for Web Services, White paper, SRI, http://www.ai.sri.com/daml/services/daml-s.pdf
12. 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.
13. Decker, K, Williamson, M and Sycara, K, Matchmaking and Brokering, 1996. Downloaded from the site of cs.cmu.edu.
14. 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
15. FIPA 97 Specification Part 1, Agent Management
16. FIPA 97 Specification Part 2, Agent Communication Language
17. 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
18. Genesereth, M. R. and Singh, N. P., A Knowledge Sharing Approach to Software Interoperation. Stanford Logic Group Report Logic-93-12
19. 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.
20. Gruber, T. R., A Translation Approach to Portable Ontologies. *Knowledge Acquisition*, 5(2):199-220, 1993.
21. Jess, http://herzberg.ca.sandia.gov/jess/.

22. 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
23. McIlraith, S., Son, T.C. and Zeng, H. `Semantic Web Services, *IEEE Intelligent Systems*. Special Issue on the Semantic Web. To appear, 2001.
24. Michalski, R. S., Carbonell, J. G., Mitchell, T. M., *Machine Learning, An Artificial Intelligence Approach*, Tioga Publishing Company
25. 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.
26. Lik Mui, Mojdeh Mohtashemi, Cheewee Ang. A probabilistic Rating Framework for Pervasive Computing Environments. The Oxygen Workshop, 2001.
27. 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.
28. Collaborative Brokering. Technical report, MCC, INSL-093-97. Abstract.
29. 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.
30. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, Katia Sycara; "Semantic Matching of Web Services Capabilities." Proceedings of the 1st International Semantic Web Conference (ISWC2002)
31. 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.
32. 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
33. Singh, N. P., A Common Lisp API and Facilitator for ABSI, version 2.0.3 Stanford Logic Group Report Logic-93-4
34. Sycara, K., Lu, J. and Klusch. M. Interoperability among Heterogeneous Software Agents on the Internet. CMU-RI-TR-98-22.
35. W3C, OWL Web Ontology Language (document set), http://www.w3.org/TR/2004/REC-owl-features-20040210/
36. Weinstein, P. and Birmingham, W.P., Comparing concepts in differentiated ontologies. *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management* (KAW'99).
37. Wickler, G. CDL: An Expressive and Flexible Capability Representation for Brokering. gw@itc.it.
38. Web Services Description Language (WSDL) 1.1, January 23, 2001, Microsoft Corporation, http://msdn.microsoft.com/xml/general/wsdl.asp
39. Giorgos Zacharia, Alexandros Moukas and Pattie Maes, Collaborative Reputation Mechanisms in Electronic Marketplaces. *Proceedings of the 32nd Hawaii International Conference on System Sciences*, 1999.