# Software Service Selection by Multi-level Matching and Reinforcement Learning

Rajeev R. Raje, Snehasis Mukhopadhyay, Sucheta Phatak, Rashmi Shastri, and Lahiru S. Gallege

Indiana University Purdue University Indianapolis, Indianapolis IN 46202, USA
{rraje,smukhopa}@cs.iupui.edu
http:www.cs.iupui.edu/~rraje

**Abstract.** The software realization of distributed systems is typically achieved as loose coalitions of independently created services. The selection of such services, to act as building blocks of a distributed system, is a critical task that requires discovery and matching activities. This selection task is generally based on simple matching techniques and without any notion of customization. This paper presents a method to achieve the service discovery process using the principles of multilevel matching based on multi-level specifications and customization based on reinforcement learning techniques. In this method, services are selected dynamically using an on-line performance-based reinforcement feedback. In contrast to methods which require the services to actually carry out a task before being selected, in the method proposed in this paper, service selection is carried out using only specification matching, thereby eliminating a large amount of redundant computation. Experimental results are presented in the context of a information classification system. These experiments demonstrate that a high degree of performance can be achieved at a much reduced computational cost using only multi-level specification-matching based reinforcement feedback signals.

**Keywords:** software services, multi-level specifications, discovery, classification, reinforcement learning, acquaintances.

## 1 Introduction

The selection of relevant software services is a necessary and critical step in the creation of distributed systems that are composed of independently developed and deployed services. The task of discovering and selecting such services for a specific query is carried out typically by a discovery system (DS). There have been many attempts of creating discovery systems such as, Jini [1], UPnP [2], SLP [3,4], UDDI [5], CORBA Trader [6], MDS [7], Ninja [8], and WSPDS [9], which have been classified as first-generation DS by [10]. A majority of these services carry the task of discovery by using a centralized publication mechanism and matching by using attribute-value pairs. In this paper, we present an approach to the discovery and selection of relevant services, for a particular query, using the principles of customization and multi-level matching. These two features make the discovery process more comprehensive and efficient than the prevalent options.

The current discovery systems do not contain the notion of customization. Thus, they typically carry out an exhaustive search to identify appropriate services for a given query. As these current alternatives are limited in size and scope of the service search space, such an exhaustive search is still feasible. However, if the notion of a large-scale service bazaar is to be realized over a wide area network, the exhaustive approach will not be feasible due to the performance issues. Hence, there is a need to carry out selective search using the principles of customization. Customization, when added to the discovery mechanism will identify relevant services for a given query without the expense of an exhaustive search over a network. This could be achieved by storing the history of the previous service discoveries. Customization can be achieved by incorporating profiling techniques that use the concepts of machine learning (e.g., reinforcement learning [11]). The challenges associated with customization are related to the nature, size, and levels of these profiles, the entities to be profiled, and the exact learning techniques used in updating these profiles. The reward and penalty techniques used in such reinforcement learning can be realized by maintaining a history about the queries propagated.

A monolithic technique based on attribute and value pairs for matching, as used by a majority of the current approaches, is clearly not sufficient to identify the most appropriate services for a given query, as it does not not differentiate between multiple similar services for a given query. As compared to this approach, matching based on multiple levels such as the type, the semantic contract, synchronization constraints, QoS values, and temporal attributes will allow the selection of the most appropriate (or relevant) services and/or also rank them using the outcome of the match for a given query. Thus, multi-level matching is a way to compare two software services and can help to determine whether one service can be substituted for another service or if one service can interact with the other service. An implicit requirement to support such a multi-level match is the presence of a multi-level specification that formally describes many facets (e.g., programmatic, semantic, QoS, etc.) of a service. The exact mechanisms for describing such a multi-level specification of a service and associated matching operators depend on the nature of a particular facet. For example, the type hierarchy can be used during the syntactical matching, while numerical operators can be used for the matching of QoS values. These definitions of matching operators help to capture the notions of generalization, specialization, substitutability, sub-typing, and interoperability of software services.

One machine learning approach for agent (or software service)[1] selection (also called acquaintance learning), in the context of document classification, is described in [12]. It uses use a vector space model, term frequency-inverse document

[1] In this paper, we have used the term *agent* interchangeably with the term *service*, as agents also offer specific services. In particular, we restrict ourselves to agents offering document classification services in this paper.

frequency method, with various and disparate document collections to produce classification a gents with varying vocabularies that classify new documents by similarity to generated centroids. If an agent generates a null vector the document is unclassified, but might be classified by an agent with a different vocabulary. An 81 term Computer Science vocabulary was broken into nine disjoint subvocabularies creating agents that attempt to classify their own document sets, and time permitting, try to assist other remote agents. In the multi-opinion model all remote agents try to classify unclassified documents but as the number of agents available increases a saturation point is reached where more agents result in a small incremental increase in successful classification while response time increases linearly with the number of agents. Thus, proper selection of a small number of remote agents could achieve high performance at low response time and could be achieved by creating a small acquaintance list for each agent using a reinforcement learning algorithm called Pursuit Learning algorithm. On the basis of quickest return, or highest similarity value return, a best acquaintance is chosen and given a positive ranking weight which will modify the probability that its future choice will result in a reward. Algorithm performance compared to four best off line chosen agents resulted was 39% better than a random selection and 363% better than a worst four performance [12].

The limitation of the approach described above is that all agents need to carry out their task of document classification to generate the performance-based reinforcement signal, although only a small number are selected in the overall task execution. This leads to a large amount of duplicate, redundant computation. In contrast, in this paper, we propose that the reinforcement signals should be generated by multi-level specification matching, rather than actual execution results of all the contacted agents. Since such matching is substantially faster and less computationally expensive than the tasks for which the agents are designed, the overall discovery system will be computationally much more efficient. However, a critical question remains regarding the relative accuracy of such specification matching-based reinforcement learning of agents, since specifications are merely proxies of the capabilities of each agent. We argue that, due to the inherent robustness and noise tolerance of probabilistic reinforcement learning methods and due to the fact the discovery relies only on correct rank-ordering of the agents (rather than requiring very accurate values of the performance measures), such approximate matching-based reinforcement learning may result in near optimal discovery system, albeit with a much lower cost than performance-based discovery systems. Indeed, in this paper, we present experimental studies related to the same information classification domain as that reported in [12], to validate this claim.

## 2 Related and Past Work

There are many efforts at designing discovery systems in the domain of service-oriented architecture. For example, Jini [1], UPnP [2], SLP [3,4], UDDI [5], CORBA Trader [6], MDS [7], Agora [13], Ninja [8], and WSPDS [9] use the attribute-value pairs, which are used in the matching process. DReggie [14], Structural ontology matching techniques [15,16], various UDDI enhancements [17,18,19,20] and SemB-UDDI[21] perform matching by the use of semantic ontology and markup languages such as DAML [22]. Such a combination does improve the matching performance over the basic attribute-value matching. GloServ [23], CDBMS [24] and OCTOPOS [25] use an hierarchy-based approach to matching. An enhancement of GloServ [26] is achieved by adding to it an ontology-based matching.

There have also been some efforts of designing DS in Cloud Computing. For example, the Cloud Service Discovery System [27] finds relevant Cloud services by using a simple Cloud Ontology, a frequency analysis, and similarity measures. [28] provides architecture for the cloud services to perform service selection with adaptive performances and minimum cost. The service selection algorithm still has the limitation of performing the basic keyword-based matching. [29] indicates the discovery process based on simple attribute matching provided by the experimental platform of Amazon EC2. All these efforts employ fairly minimal approaches to describe the services that are deployed in clouds and hence, their matching semantic tends to be simplistic. None of these approaches use customization and multi-level matching that is beyond the basic attribute-based and the ontology-based semantic matching schemes.

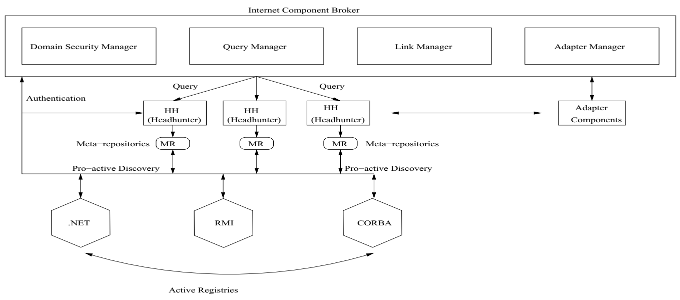

**Fig. 1.** The UniFrame Resource Discovery Service (URDS)

## 2.1 UniFrame Resource Discovery Service (URDS)

UniFrame [30] provides an environment for a seamless inter-operation of heterogeneous distributed services. A service in UniFrame is described by a multi-level specification, called UMM specification, which is an enhancement of the multi-level contract principle introduced in [31]. The UMM specification for a service is primarily made up of syntactic, semantic, synchronization, QoS, and deployment levels. Hence, in UniFrame the services are deployed on the network with their multi-level specifications. The URDS provides the infrastructure that supports the publication, deployment, and detection of the services. The URDS is a pro-active and hierarchical discovery system. The URDS architecture [32,33,34], shown in Figure 1, consists of: a) Headhunters (HH), b) Active Registries (ARs), and c) Internet Component Broker (ICB). Each headhunter is associated with a meta-repository, which serves as a local store. Services, when implemented, are registered with the active registries using their UMM specifications. Once services are registered, the task of a headhunter is to discover them pro-actively, store their UMM specifications in its meta-repository, and perform a matching between the UMM specifications and any incoming queries issued by the user. The headhunters use multi-level matching (i.e., matching all the levels of the UMM specifications while responding to a specific query to identify relevant services. Headhunters may collaborate with each other if a requested service is not available in their local stores. The active registries are enhanced versions of the native registries of various implementation models (e.g., Java-RMI) with which services are registered. These enhancements allow Active Registries to listen and respond to multi-cast messages from the headhunters and have introspection capabilities to discover the services along with their specifications. The Internet Component Broker comprises of the Query Manager (QM), the Domain Security Manager (DSM), Adapter Manager (AM), and the Link Manager(LM). The Domain Security Manager serves as an authorized entity, which handles the secret key generation and distribution and enforces group memberships on other entities in URDS. The purpose of the Query Manager is to dispatch a user's query to the available headhunters. These headhunters then return lists of services matching the search criteria expressed in the query. The Link Manager serves to establish links with other Internet Component Brokers for the purpose of creating a federation and to propagate queries received from the Query Manager to other linked Internet Component Brokers. The Adapter Manager allows interoperability between heterogeneous services that may be needed to create a distributed system. In this paper, we have used the URDS (mainly the headhunter and its multi-level matching techniques) for the Multi-agent Classification System along with the principles of customization.

## 2.2 Reinforcement Learning

Originally motivated by mathematical psychology models of animal and child learning, reinforcement learning refers to the ability of an agent to learn long-term optimal behavior through the use of a reinforcement, i.e., an on-line performance feedback from a teacher or environment. The reinforcement, in turn, may be qualitative, infrequent, delayed, or stochastic. There is a rich body of reinforcement learning literature encompassing a wide array of learning models and algorithms. One of the earliest models of reinforcement learning is called a learning automaton [35] where the agent attempts to learn the optimal action from a finite set using reward/penalty reinforcement from a stationary teacher/environment with unknown reward probabilities. The learning problem is formulated as updating the agent's action probabilities on the basis of trials consisting of an action performed and the reinforcement received. While a wide variety of model-based and model-free learning algorithms has been proposed for a learning automaton with different asymptotic convergence properties, a popular model-free algorithm is the so-called $L_{RI}$(Linear Reward-Inaction) algorithm described by

$$p_i(k+1) = p_i(k) + \alpha r(k)(1 - p_i(k))$$

$$p_j(k+1) = p_j(k) - \alpha r(k) p_j(k)$$

where $p_i(k)$ is the agent's probability of choosing action $a_i$ at trial $k$ , $a_i$ is the action chosen at trial $k$ , $r(k)$ is the reinforcement received (with $r(k) = 0$ signifying penalty, and $r(k) = 1$ signifying reward) , and $\alpha > 0$ is the learning step-size. The idea is to increase the probability of the chosen action linearly if a reward is received (while reducing the other action probabilities) and not to change the action probabilities if a penalty is received. The $L_{RI}$ algorithm has been shown to be $\epsilon$-optimal, i.e., the asymptotic probability of converging to the optimal action can be made as close to 1 as desired by choosing a sufficiently small step-size $\alpha$. While the $L_{RI}$ algorithm belongs to the class of model-free learning algorithm (since it does not maintain or use any estimate of the environmental reward probabilities), there are also algorithms that are model-based. The Pursuit Learning Algorithm [35], for example, maintains and updates estimates of the reward probabilities, and adjusts the action probabilities by a step size in the direction of the unit vector that represents the current estimate of the optimal action. The Pursuit Learning Algorithm has also been proved to be $\epsilon$-optimal.

It is quite clear that some form of machine learning technique needs to be used for the selection of remote services (i.e., acquaintance learning) to deal with uncertainty and/or dynamic changes in the environment. The advantage of reinforcement learning over other machine learning approaches (such as supervised learning and unsupervised learning) is that the former is inherently an on-line learning method, where the reinforcement data required for learning is generated during operation, thereby avoiding a large effort in off-line data generation and collection. Further, the reinforcement feedback signal needs to be only a qualitative and noisy indicator of how well the agent is performing, rather than a labeling of the data (as in supervised learning) to indicate the "correct" action. These advantages are particularly relevant in open, dynamic services environments where existing services may be removed and new services may be added at unknown instants of time. Reinforcement learning offers the possibility of adapting to these dynamic changes, by making use of the reinforcement feedback, without a large effort (and/or delay) in retraining of the agent. Hence, we have used the reinforcement learning technique in our proposed methodology for the design of a multi-agent classification system.

## 3 Methodology

Based on the work reported in [12], we propose a Multi-agent Classification System with specification matching. In this system, we propose the use of active registry and headhunters of the URDS for choosing the correct agent for classification of an incoming document. Multiple classification agents will register their

UMM specifications with the active registries and are eventually discovered by the headhunters. The headhunter in our experiments maintains an acquaintance list made up of top three classification agents. This list is dynamically updated using the reinforcement learning with the help of previous collaboration history. Unlike [12], the headhunter maintains its acquaintance based on matching of agent specifications with the clients query. For example (more details of our approach are provided in next section) in one scenario, the headhunter chooses the members of the acquaintance list by selecting the top three agents decided by the matching of their specifications against the incoming query and then the top acquaintance with highest matching score is chosen for the task of classification. Reward in the reinforcement learning algorithm is based on the matching score. Top acquaintance gets reward of 1 and all other acquaintance get reward of 0. Their Action Probability, and Estimated reward probability are updated by the formulas given above (and given in [12]). Headhunter now updates its acquaintance list by choosing the agent based on its action probability. To explore the learning we use following algorithm for choosing the first acquaintance:

`SortedActionProbability` holds the action probability values of all the agents in an ascending order.

```
Rand = RandomNum(0,1);
Sum = SortedActionProbability [0];
i=0;
While(Rand<Sum)
{
  Sum=Sum+SortedActionProbability[i++];
}
ChooseAgentAtIndex[i];
```

Other acquaintances are the agents with highest Estimated Reward Probability. Thus by using matching technique for choosing the acquaintance, we reduce the classification cost and final classification is only done by the top matching acquaintance. This technique is useful when the specification of an agent or service clearly describes the agent. If the specification is detailed enough then we can correctly identify the best service for that query. Next section analyzes the experimental results of this system. Similar to experiments described in [12], our experiments implement the pursuit learning algorithm for learning automata that works in feedback with environment. For each action, the environment provides reward or punishment reinforcement with some probability. We have designed a model-based pursuit algorithm that maintains the estimate of the action probabilities, which are unknown to classification agents. A query from the client is forwarded to a headhunter. Each headhunter maintains an acquaintance list and keeps track of each agent's action probability. It matches the user query with registered UMM specifications of agents. Whenever a successful match is found, the headhunter rewards and increases the action probability of the selected agent. Initially, all the agents have an equal probability and eventually each agent will be tried and explored.

We have implemented two different techniques for matching of agent UMM specification to queries. The first technique is called as the explicit matching. In this technique, the queries contain a domain and sub domain of the document to be classified. Every agent when registered with the headhunter provides a detailed UMM specification which is extended to include its domain of classification. Hence in this approach, the headhunter matches the query domain along with other parts of the UMM specification and decides the matching score. This technique assumes that the clients have some idea about the classification domain. On the other hand, if the clients do not have any idea about the domain of the document they want to classify, then the explicit domain matching will not work. Hence, we use another method wherein the headhunter selects the acquaintances in a random manner initially. In both these cases, after the initial selection, the future selections of agents use the reinforcement learning technique.

## 4 Experimentation

We conducted a number of experiments to test the effectiveness of the proposed approach. The first set of experiments did not use the reinforcement learning algorithm used and the headhunter would just select any random agent for classification or would send the document to all the agents for classification. This formed as the base cases for comparing the classification performance. The next sets of experiments involved the use of acquaintance lists of headhunters selected by using the explicit (i.e., the direct description of classification domain) and implicit (i.e., using the thesaurus) matching techniques. Our experiments included ten agents with overlapping thesauri and ten sets of documents from a single sub domain. The results and analysis of experiments are provided below.

### 4.1 Experiments without Reinforcement Learning Algorithm

Experiment 1: In this experiment, all documents were sent to all classifier agents so that the resulting classification success rate was 100%. However, since all classifier agents attempted to classify all documents, this is by far the most expensive method computation and communication-wise.

Experiment 2: In this experiment, documents were sent to three classifier agents chosen at random. The classification success rate was observed to be only 40%, but the computation time required was less than the previous experiment.

Experiment 3: In this experiment, documents were sent to only one randomly chosen classifier agent. The success rate in classification was observed to be only 20% in this case but it was faster than the previous two cases.

### 4.2 Experiments with Reinforcement Learning Algorithm Using Explicit Matching

Experiment 1: Classification achieved by selecting the Top Acquaintance

In this experiment, the top acquaintance is selected by a headhunter for classification queries by means of the matching the document domain, along with other aspects of the UMM specification of agents. As the acquaintance list is

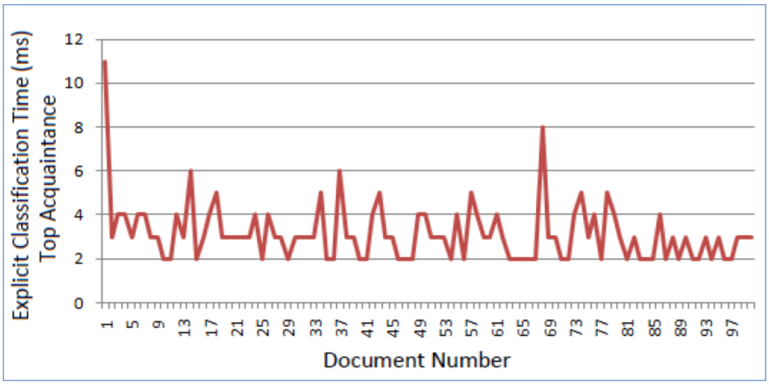

**Fig. 2.** Explicit Classification by Selecting the Top Acquaintance – Classification Time

constantly updated, using the reinforcement algorithm, during each classification only the top one is chosen. The performance (processing time), and the classification success rate are shown Figures 2 and 3 respectively. In Figure 2, the X-axis denotes the 100 documents (10 sets of 10 documents) and the Y-axis indicates the processing time (in milli-seconds) taken by the top acquaintance. After the initial period, the classification time saturates a value that is between 2 and 4 ms. The X-axis in Figure 3 indicates the set number (each set consisting of 10 documents), while the Y-axis indicates the number of successful document classifications for each set. From Figure 3, it can be seen that the classification performance quickly reaches the 100% level (i.e., 10 successful classifications in each set). This is hardly a surprise, as the top acquaintance which is most suitable for classification, is learnt by the headhunter quickly. When compared with the non-learning based exhaustive technique (Section 4.1), this scheme provides a comparable performance with a lot less computation (i.e., only one agent classifying the documents versus all 10 agents classifying the documents).

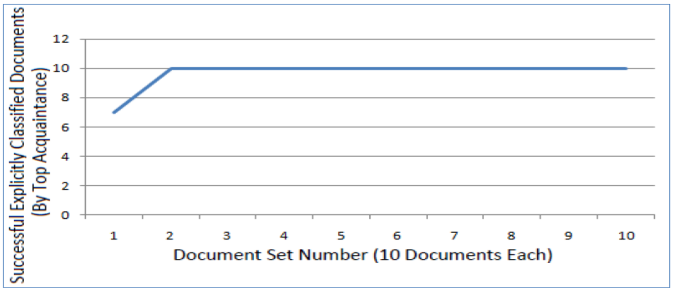

**Fig. 3.** Explicit Classification by Selecting the Top Acquaintance – Classification Success

Experiment 2: Classification done by Randomly chosen acquaintance.

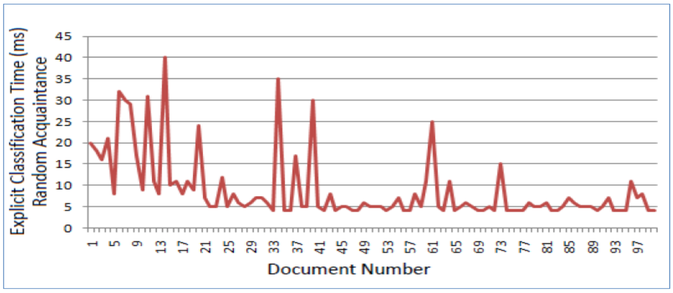

**Fig. 4.** Explicit Classification by Selecting a Random Acquaintance – Classification Time

In this case, an agent is selected by the headhunter from the acquaintance list in a random manner. Figures 4 and 5 show the performance of this scheme. The X and Y axes indicate the same entities as in Figures 2 and 3. As seen from Figure 4, the classification time varies randomly an in some cases is similar to the one in Figure 2, and the classification performance (Figure 5) is poorer than the case with the selection of the top-acquaintance (Figure 3). This is also expected, as the random selection does not guarantee the selection of the top acquaintance.

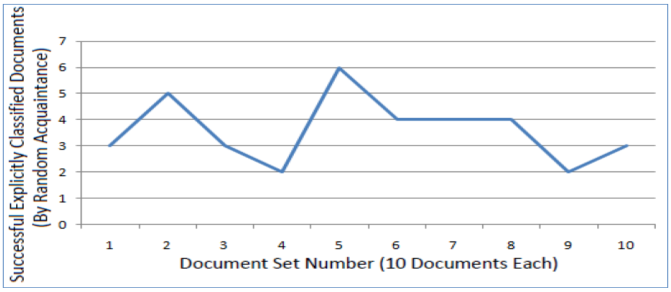

**Fig. 5.** Explicit Classification by Selecting a Random Acquaintance – Classification Success

## 4.3 Experiments with Reinforcement Learning Algorithm Using Implicit Matching

Experiment 1: Classification achieved by selecting the Top Acquaintance

In this experiment, the top acquaintance is selected by a headhunter for classification queries in a random manner initially. The selected agent gets a chance to

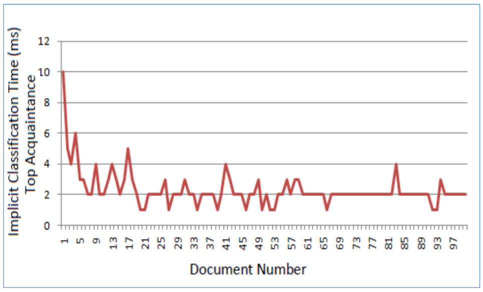

**Fig. 6.** Implicit Classification by Selecting the Top Acquaintance – Classification Time

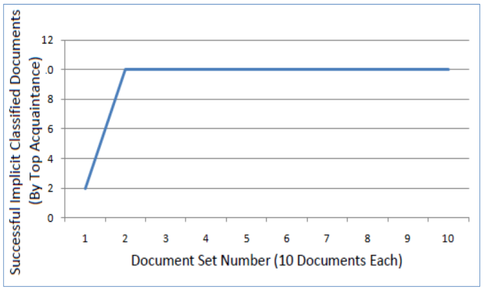

**Fig. 7.** Implicit Classification by Selecting the Top Acquaintance – Classification Success

classify the document and if it is successful, it gets a reward (as indicated above by the reinforcement algorithm), else it gets a penalty. The future iterations always use the top acquaintance during the selection process. The performance (processing time), and the classification success rate are shown Figures 6 and 7 respectively. In Figure 6, the X-axis denotes the 100 documents (10 sets of 10 documents) and the Y-axis indicates the processing time (in milli-seconds) taken by the top acquaintance. After the initial period, the classification time saturates a value that is close to 2 ms. The X-axis in Figure 7 indicates the set number (each set consisting of 10 documents), while the Y-axis indicates the number of successful document classifications for each set. From Figure 7, it can be seen that the classification performance quickly reaches the 100% level (i.e., 10 successful classifications in each set). This is hardly a surprise, as the top acquaintance is learnt by the headhunter quickly.

Experiment 2: Classification achieved by selecting Randomly chosen Acquaintance.

In this case, the initial acquaintance list is randomly selected and an agent is also selected by the headhunter from the acquaintance list in a random manner. Again, the selected agent, if successful, gets a reward, else gets a penalty and the list is updated accordingly. Figures 8 and 9 show the performance of this scheme. The X and Y axes indicate the same entities as in Figures 6 and 7. As

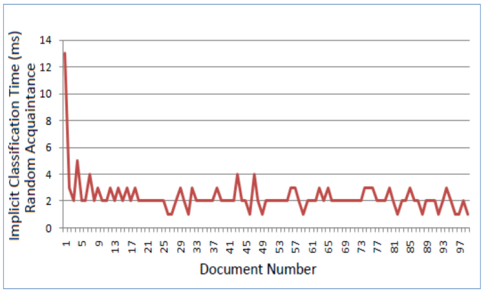

**Fig. 8.** Implicit Classification by Selecting a Random Acquaintance – Classification Time

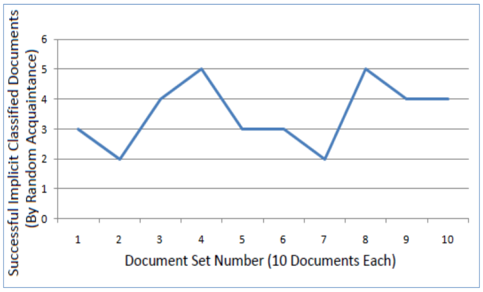

**Fig. 9.** Implicit Classification by Selecting a Random Acquaintance – Classification Success

seen from Figure 8, the classification time is similar to the case with the selection of the top acquaintance (Figure 6) but the classification performance (Figure 9) is poorer than the case with the selection of the top-acquaintance (Figure 7). This is also expected, as the random selection does not guarantee the selection of the top acquaintance.

## 4.4 Experiments Involving Fault Situations

Here to simulate the fault situations, during the classification process, the top acquaintance agent is deleted from the system after every 10 document classifications and after every next 10 passes a new agent is added to the system. The results are presented below Figures 10 and 11. As seen from Figure 10, the classification time does increase when the top acquaintance agent is deleted and it does reduce as the system learns the new top acquaintance. However, the classification performance (as shown in Figure 11) does not exhibit a predictive pattern. This can be attributed to the randomness of documents and the inclusion of a random agent into the system.

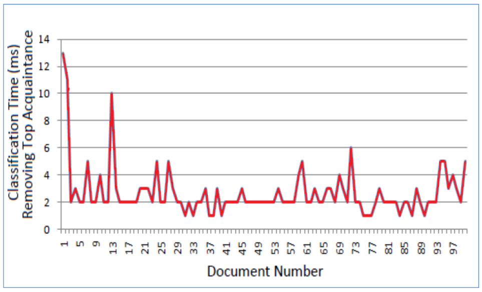

**Fig. 10.** Classification while Deleting the Top Acquaintance – Classification Time

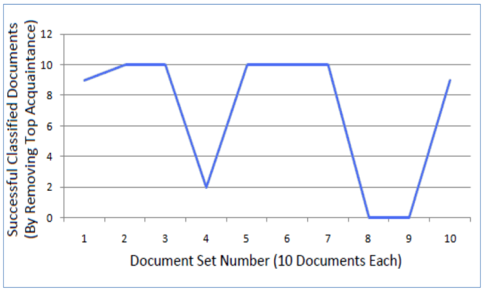

**Fig. 11.** Classification while Deleting the Top Acquaintance – Classification Time

### 4.5 Summary of Experimental Results

The above experiments indicate that the learning-based algorithm outperforms the discovery without learning (except the case where the documents are sent to all the agents) when the classification success is chosen as the metric of performance. When classification done by all classifiers, and with no learning, the classification success is 100% but it is at the cost of additional computation. In the case of explicit and implicit matching, with learning based algorithm, the scheme which uses the top acquaintance for classification outperforms the scheme where randomly selected acquaintances are used in the classification. The scheme involving top acquaintances for classification achieves a comparable classification success with the exhaustive case (i.e., on learning but the usage of all the agents in classification). Thus, incorporation of learning, in addition to multi-level matching, delivers a comparable performance while selecting appropriate agents for document classification.

## 5 Conclusion

In this paper, we have presented a method for learning of preferred acquaintance by using specification matching for classification agents. This is useful where agents are not only distributed in nature but they randomly join and leave the system. Our experiments show that the learning based on specification matching not only reduces cost of sending document to all classifier, but also gives a good classification performance. We show that, with little overhead of maintaining model based pursuit learning algorithm and specification matching, the service discovery system converges fast and provides near optimal solution as far as the classification success is considered.

## References

1. Sun Microsystems. Jini Specifications V2.0, `http://wwws.sun.com/software/jini/specs/`
2. UPnP Organization. UPnP Home Page (2005), `http://www.upnp.org`
3. Kemp, J., St. Pierre, P.: Service Location Protocol for Enterprise Networks. Wiley and Son Inc., ISBN 0-47-3158-7
4. OpenSLP Organization. OpenSLP Home Page (2005), `http://www.openslp.org`
5. UDDI Technical White Paper (2000), `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`
6. Object Management Group. Trading Object Service Specification (2000), `http://www.omg.org/docs/formal/00-06-27.pdf`
7. Globus Toolkit (2007), `http://www.globus.org/toolkit/`
8. von Behren, J., Brewer, E., Borisov, N., Chen, M., Welsh, M., MacDonald, J., Lau, J., Culler, D.N.: A Framework for Network Services. In: Proceedings of USENIX Annual Technical Conference (2002)
9. Banaei-Kashani, F., Chen, C., Shahabi, C.: WSPDS: Web Services Peer-to-peer Discovery Service (2004), `http://infolab.usc.edu/DocsDemos/isws2004_WSPDS.pdf`
10. Dabrowski, C., Mills, K., Quirolgico, S.: A Model-based Analysis of First-Generation Service Discovery Systems. Technical report, NIST Special Publication, 500-260 (October 2005), `http://w3.antd.nist.gov/pubs/SP500_260final.pdf`
11. Thathachar, M., Sastry, P.: A New Approach to the Design of Reinforcement Schemes for Learning Automata. IEEE Transactions on System Man Cybernetics 15, 168–175 (1985)
12. Mukhopadhyay, S., Peng, S., Raje, R., Palakal, M., Mostafa, J.: Multi-Agent Information Classification Using Dynamic Acquaintance Lists. Journal of the American Society for Information Science and Technology 54(10), 966–975 (2003)
13. Seacord, R., Hissam, S. and Wallnau, K. Agora: A Search Engine for Software Components. Technical report, Carnegie Mellon University, CMU/SEI-98-TR-011, ESC-TR-98-011 (1998)
14. Chakraborty, D., Perich, F., Avancha, S., Joshi, A.: DReggie: A Smart Service Discovery Technique for E-Commerce Applications. In: Proceedings, 20th Symposium on Reliable Distributed Systems (October 2001)
15. Di Martino, B.: Semantic web services discovery based on structural ontology matching. In: Proceedings of IJWGS (2009)
16. Lin, C., Wu, Z., Deng, S., Kuang, L.: Automatic Service Matching and Service Discovery Based on Ontology. In: Jin, H., Pan, Y., Xiao, N., Sun, J. (eds.) GCC 2004. LNCS, vol. 3252, pp. 99–106. Springer, Heidelberg (2004)

17. Paolucci, M., Kawamura, T., Payne, T., and Sycara, K. Importing the Semantic Web in UDDI. In: Workshop on EBusiness and Semantic Web (2001)
18. Kawamura, T., De Blasio, J.-A., Hasegawa, T., Paolucci, M., Sycara, K.: Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Registry. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 208–224. Springer, Heidelberg (2003)
19. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic Matching of Web Services Capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 333–347. Springer, Heidelberg (2002)
20. Colgrave, J., Akkiraju, R., Goodwin, R.: External Matching in UDDI. In: Proceedings of IEEE international Conference on Web Services (2004)
21. Aguilera, U., Abaitua, J., Diaz, J., Bujan, D., Ipina, D.: Semantic Matching Algorithm for Discovery in UDDI. In: Proceedings of International Conference on Semantic Computing (2007)
22. DARPA. The DARPA Agent Markup Language (2006), `http://www.daml.org/`
23. Arabshian, K., Schulzrinne, H.: GloServ: global service discovery architecture. In: Mobile and Ubiquitous Systems: Networking and Services, pp. 319–325 (2004)
24. Skouteli, C., Samaras, G., Pitoura, E.: Concept-based discovery of mobile services. In: MDM 2005: Proceedings of the 6th International Conference on Mobile Data Management, pp. 257–261. ACM, New York (2005)
25. Gu, T., Qian, H., Yao, J., Pung, H.: An architecture for flexible service discovery in OCTOPUS. In: ICCCN, pp. 291–296 (2003)
26. Arabshian, K., Dickmann, C., Schulzrinne, H.: Ontology-Based Service Discovery Front-End Interface for GloServ. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 684–696. Springer, Heidelberg (2009)
27. Taekgyeong, H., Kwang, M.: An Ontology-enhanced Cloud Service Discovery System. In: Proceedings of International Multiconference of Engineers and Computer Scientists (2010)
28. Zeng, W., Zhao, Y., Zeng, J.: Cloud service and service selection algorithm research. In: Proceedings of ACM/SIGEVO Summit on Genetic and Evolutionary Computation (2009)
29. Rajiv, R., Liang, Z., Xiaomin, W., Anna, L.: Peer-to-Peer Cloud Provisioning: Service Discovery and Load-Balancing. In: Proceedings of CoR (2009)
30. Indiana University Purdue University Indianapolis. UniFrame Project (2010), `http://www.cs.iupui.edu/uniFrame`
31. Beugnard, A., Jezequel, J., Plouzeau, N., Watkins, D.: Making Components Contract Aware. IEEE Computer 32(7), 38–45 (1999)
32. Siram, N.: An Architecture for the UniFrame Resource Discovery Service. Master's thesis, Indiana University Purdue University Indianapolis, Department of Computer and Information Science (2002)
33. Siram, N., Raje, R., Bryant, B., Olson, A., Auguston, M., Burt, C.: An Architecture for the UniFrame Resource Discovery Service. In: van der Hoek, A., Coen-Porisini, A. (eds.) SEM 2002. LNCS, vol. 2596, pp. 20–35. Springer, Heidelberg (2003)
34. Raje, R., Gandhamaneni, J., Olson, A., Bryant, B.: MURDS: A Mobile-Agent-based Distributed Discovery System. In: Taniar, D. (ed.) Encyclopedia of Mobile Computing and Commerce, Hershey, USA, vol. 1, pp. 207–212 (2007)
35. Narendra, K.S., Thathachar, M.A.L.: Learning Automata: An Introduction. Prentice-Hall (1989)