Ad-UDDI: An Active and Distributed Service Registry

Zongxia Du[1], Jinpeng Huai[1], Yunhao Liu[2]

[1]School of Computer Science
Beihang University, Beijing, P. R. China
duzx@act.buaa.edu.cn,
[2]Dept. of Computer Science
Hong Kong Univ. of Science and Technology, Hong Kong
liu@cs.ust.hk

Abstract. In SOA (Service Oriented Architecture), web service providers use service registries to publish services and requestors use registries to find them. The major current service registry specifications, UDDI (Universal Description, Discovery and Integration), has the following drawbacks. First, it replicates all public service publications in all UBR (Universal Business Registry) nodes, which is not scalable and efficient, and second, it collects service information in a passive manner, which means it waits for service publication, updating or discovery request passively and thus cannot guarantee the real-time validity of the services information. In this paper, we propose an active and distributed UDDI architecture called Ad-UDDI, which extends and organizes the private or semi-private UDDIs based on industry classifications. Further, Ad-UDDI adopts an active monitoring mechanism, so that service information can be updated automatically and the service requestors may find the latest service information conveniently. We evaluate Ad-UDDI by comprehensive simulations and experimental results show that it outperforms existing approaches significantly.

1 Introduction

Web services based on service-oriented architecture (SOA) provide a suitable technical foundation for interoperability and integration of applications [1, 2]. To make the web services accessible to users, service providers describe their interfaces with WSDL [3] and publish the description to service registries, so that service requestors may find them conveniently[4]. As a result, service registries play an important role in SOA. Most today's service registries comply with UDDI [5] (Universal Description, Discovery and Integration) specifications, whose initial focus was geared to working as UBR (Universal Business Registry), a master directory for all public web services. However, Su Myeon Kim et. al. showed their observations on public web services [6] on the monitoring result about UBR, in which only 34% of the Web Service (WS) are valid. Here a 'valid' WS (Web Service) means a WS with a URL where a WSDL file is retrievable. Furthermore, a large portion of the downloaded WSDL files are invalid due to syntax errors. On the other side, very few organizations update their service information after their first publication.

We have following observation on current UDDI service registry in SOA. First, it replicates all web service publications in all UBR nodes, which is not suitable for the

large number of services. Second, it collects service information in a passive manner, which means it waits for service publication, updating or discovery request passively. Consequently, the real-time validity of the service information is not guaranteed.

In this paper, we propose an active and distributed UDDI architecture called Ad-UDDI, which extends and organizes the private or semi-private UDDIs based on industry classifications. Further, Ad-UDDI adopts an active monitoring mechanism, so that service information can be updated automatically and the service requestors may find the latest service information conveniently. We evaluate Ad-UDDI by comprehensive simulations and experimental results show that it outperforms existing approaches significantly.

The rest of this paper is organized as follows. Section 2 presents an overview of related works. Section 3 introduces the design of Ad-UDDI. We show our experimental results in Section 4 and conclude this work in Section 5.

2. Related work

Flexible resource management is a key point for the collaboration between partners. Traditional centralized resource management framework have limitations both in their failure tolerance and scalability[7]. Recent years, there are more and more attention changed to the distributed framework[8, 9] for scalability and flexibility.

UDDI v3.0.2 released in 2004 recognizes the needs for multiple registries, as well as the interactions among registries [5]. Due to the large number of registries focusing on various interests, service publication and discovery becomes challenging. In addition, UDDI v3 provides subscription mechanisms to enable affiliate registry to obtain change information of a root registry, but there is no approach to get the real status of the services except waiting passively for the updating requests from providers.

In ADS (Advertisement and Discovery of Service Protocol) issued by IBM[10], service descriptions are collected by UDDI crawler rather than being manually published by providers. The design of crawler borrows the idea from the web search engine and sets the file, *svcsadvt.xml*, to the root of Web Server. When a crawler finds such a file, it collects the corresponding service information of the web site. However, when the web crawler goes ahead according to the hyperlink in the web page, there is no hyperlink information in the web service description. Therefore, the diffusing of crawler is much difficult. UDDIe [11] is an extended registry for web services, which exploits the lease time of each service to ensure the availability of service information in registries. However, the lease time and availability of service is dependent on the relationship established in advance between UDDIe and the service providers, and there is no method for checking the real availability of services.

MSWSDI [12] is a part of the ongoing METEOR-S [13] project. It is a scalable P2P registry infrastructure for semantic publication and discovery of web services. It employs an ontology-based approach to organize the registries and enable domain-based semantic classifications for all web services. Each of these registries supports semantic discovery of the web services. In MSWSDI, the relationship among the registries is managed based on a Registries Ontology. Because the Registries Ontology needs specific management and maintenance, the organization of the registries is not trivial. Authors in [14] proposed a federated architecture for P2P web-services, in which a federation for UDDI-enabled peer registries is employed in a decentralized

fashion. Service providers publish their services on a centralized UDDI and then join service syndication. Obviously, a single point of failure cannot be avoided. Also, no mechanism is designed for getting real status of services.

3. Design of Ad-UDDI

In this section, we introduce Ad-UDDI active monitoring mechanism and its distributed architecture.

3.1 Design of Active Monitoring

The validity of service information in registries is of great importance. However, due to the fact that few organizations update their published information in registries on time [6], a certain mechanism has to be applied to monitor the service status and update the information in registries automatically.

In this design, a registry server, called Ad-UDDI server, checks the real time status of services and collects the service information periodically. The state chart of the Ad-UDDI server, as shown in Fig.1, consists of three states, *Normal*, *Update*, and *Monitor*. In the *Normal* state, the Ad-UDDI server waits for periodically monitoring triggers or incoming requests. In the *Monitor* state, the Ad-UDDI server initiates a monitoring request to the service provider. In *Update* state, the Ad-UDDI server updates the service information in it based on the returned messages from providers.

Once triggered by a timer, the Ad-UDDI transfers from the *Normal* to the *Monitor* state and starts checking the real status of services. If the monitored service has not been updated yet, the Ad-UDDI returns to the *Normal* state triggered by a 'nonUpdate' message. If the monitored service is updated, the Ad-UDDI transfers from the *Monitor* state to the *Update* state, executes the information updating process. After that, the Ad-UDDI returns to the *Normal* state again. Another way, the Ad-UDDI in a *Normal* state transfers into the *Update* state if it is requested by the providers.

Figure 2 illustrates the interaction process of the active monitoring mechanism. The Ad-UDDI server sends a 'Monitor' message to a service provider periodically, containing the registered service name, service key and service version. The service provider checks each item in the 'Monitor' message with its own. To simplify the handling process and reduce the load, only service name, key and version are compared. If they are identical, a message of 'nonUpdate' is returned. Otherwise, new service information is sent to the Ad-UDDI server via a 'save_Service' message. On receiving a 'nonUpdate' message, the Ad-UDDI server terminates the present monitor thread. On receiving a 'save_Service' message, the Ad-UDDI server conducts the service updating process. If there is no message returned within given time period, the service is considered to be unavailable and the Ad-UDDI server will step into 'Update', claiming the unavailability of the service.

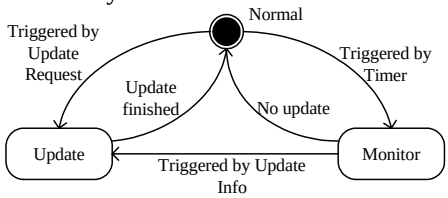

Fig.1. The statechart of Ad-UDDI

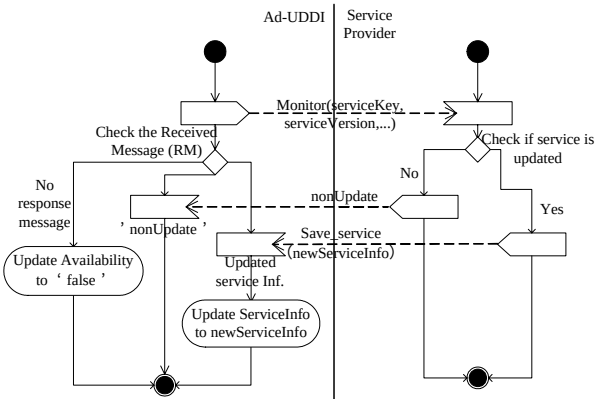

Fig.2. The interaction process of active monitoring

It is noteworthy that an unavailable service might be caused by a network failure, a temporal invalidation of the provider's server, or the undeployed service. Therefore, we should deal with the unavailable service based on the service monitoring strategies, instead of a simple deletion. In our implementation, monitoring strategy is often as follows: 1) service information is to be cancelled after 10 times of monitoring without any returned message; 2) on receiving a returned message, the Ad-UDDI updates the service information accordingly and resets the service as available; 3) on receiving a service discovery request, the Ad-UDDI server searches in available services only.

3.2 Design of Distributed Architecture

The Ad-UDDI adopts a three-layered structure of distributed service registry, as Fig. 3. The top layer is the root registry layer, in charge of managing the Ad-UDDI service information. The root is a special Ad-UDDI server, in which every Ad-UDDI server in the middle layer publishes its own information as a web service. In addition, we do not let this layer publish and monitor business services so as to reduce its work load. The middle layer is the business service registry layer, in which all Ad-UDDI servers are initiated following GICS (Global Industry Classification Standard) [15]. Normally, the business services belonging to a classification are registered in corresponding Ad-UDDIs and multiple industry classification services may be registered in one Ad-UDDI. The bottom is the service layer, in which every service publishes their information to one or more Ad-UDDI based on to their service type and industry classification.

The solids in Fig. 3 show the publishing relationship, such as business services publish their information to the corresponding Ad-UDDI and Ad-UDDIs publish their information to the root. The dash lines in the middle layer denote the neighboring relationship, such as Ad-UDDI 1, 2 and 4 have established the neighboring relationship according to their classification (Transportation). The dash lines in the bottom layer show the interaction relationship between services.

There are mainly five operations in such distributed architecture, including *adding* and *closing* of an Ad-UDDI, *Ad-UDDI neighbor maintenance*, *service querying*, and *service updating*.

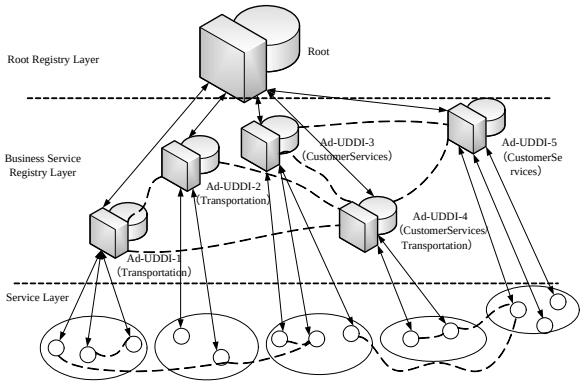

Fig.3. The distributed architecture

a) Adding a new Ad-UDDI

In case of adding a new Ad-UDDI, it sends its basic information to the root registry, and search in the root registry for other Ad-UDDIs in the same industry classification. The new Ad-UDDI then requests to establish neighboring relationship with existing same category Ad-UDDIs. When a request is granted, the two Ad-UDDIs record the other side's information. Finally, once the neighboring relationship is set up, the publishing and discovering of services are performed within the middle layer without accessing to the root. Therefore, while the root is a single point of failure, it does not impact the publishing and discovery of web services. In that case, only adding a new Ad-UDDI will be fail. The related interaction protocol is shown in Fig. 4.

b) Closing an Ad-UDDI

In case of closing an Ad-UDDI, the following four modes are possible in this design: 1) to close an Ad-UDDI directly, discarding all service stored without contacting the root registry; 2) discard all service information but inform the root registry of its unavailability; 3) transfer all service information to its neighbors before leaving without informing the root; 4) move all service information to neighbors, sends a closing request to the root registry, and waits for permission. Obviously, the complexities of above four modes increase in order. In our design, an Ad-UDDI might be closed by anyone of them. Although the fourth one is usually encouraged, the first mode is used when an Ad-UDDI fails to connect with the root registry center due to the network failure. Figure 5 illustrates the fourth mode interaction protocol.

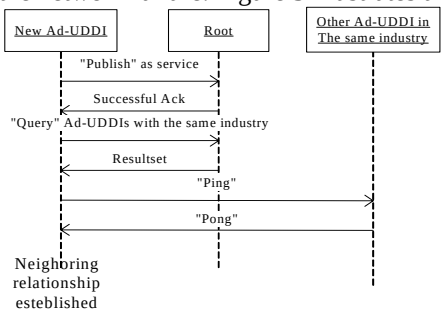

Fig.4. The interaction protocol of adding an Ad-UDDI

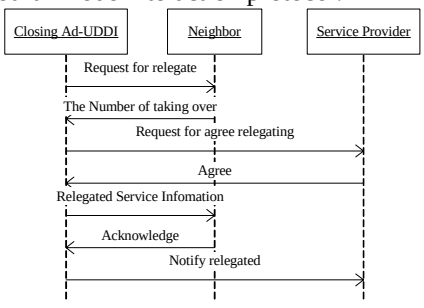

Fig.5. The interaction protocol of closing an Ad-UDDI

c) Neighbor Maintenance

Neighboring relationship among the Ad-UDDIs is established when a new member joins. When an existing member leaves, it is possible that it does inform its neighbors. In this design, we require the root registry center monitors the status of all Ad-UDDIs and broadcasts the updated information to all Ad-UDDIs in the same category using the subscription method in UDDI v3.

d) Service Querying

Each Ad-UDDI maintains the service information published in it and deals with the service query from service requestors. To improve the service querying efficiency, each Ad-UDDI caches the recent searching results. On receiving a service query, an Ad-UDDI looks up its cache repository. If the desired service is found, the Ad-UDDI returns the result to the requestor and terminates the query. If there is no target found, the Ad-UDDI goes on querying in local and neighboring repositories, and then stores the querying results into local cache after returning the results to the requestor.

e) Diffused Updating of Service Information

In this distributed structure, the updating of the service information is extended to all neighboring Ad-UDDIs whose local caches have cached related service information. This procedure is called the diffused updating of the service information.

With both the diffusing updating and the active monitoring mechanism, the statechart of the Ad-UDDI in Fig. 2 is extended into the one shown in Fig. 6. Having updated the service information locally, the Ad-UDDI broadcasts an updating message to its neighbors, so that the neighboring Ad-UDDIs can update corresponding information in their caches.

3.3 Implementation Experiences

The implementation of Ad-UDDI prototype server contains four repositories, i.e. the Local Service Information Repository (LSIR), the Local User Information Repository (LUIR), the Neighbor Ad-UDDI Information Repository (NAIR) and the Cached Service Information Repository (CSIR), as illustrated in Fig. 7. The LSIR and the LUIR are similar with those in UDDI servers. The NAIR and the CSIR are implemented purposely for the Ad-UDDI. The NAIR holds the information of neighboring Ad-UDDIs. The NAIR stores the neighbor's name, its access point, its industry classification, etc. The CSIR caches the service information which has been queried by requestors before. The major functional blocks to manage the information in the repositories are as follows.

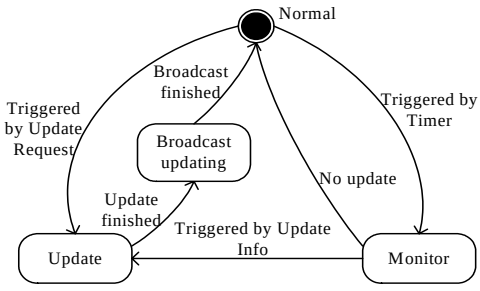

Fig.6. The extended statechart of Ad-UDDI

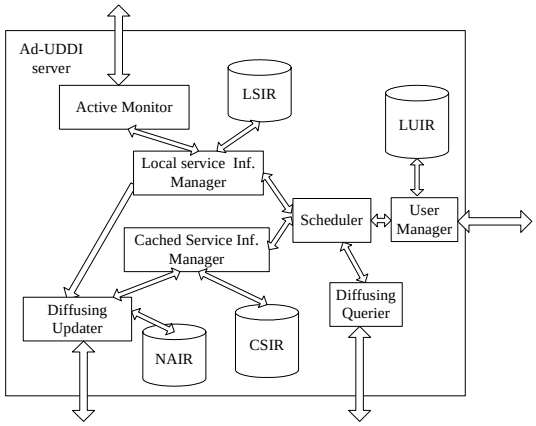

Fig.7. The architecture of Ad-UDDI server

User Manager manages the information of the service providers and requestors registered in current Ad-UDDI. It accepts registration requests from new users, updates the information for registered users, and implements access control.

Scheduler invokes managers according to requests (such as publishing / querying).

Local Service Information Manager publishes the service information to the local service repository, queries the service information in local repository and updates information in local repository.

Active Monitor connects the service providers who published their services in this Ad-UDDI, monitors the real-time service status, and updates the service information.

Cached Service Information Manager manages and maintains the CSIR, and caches the returned queries. On receiving a query requests, it searches in the CSIR for the matched service. It also guarantees the synchronization of the information. At last, it manages the cache size. When too much information is cached, the least recently requested ones will be deleted.

Diffusing Updater performs the information synchronization among the Ad-UDDIs. When the information of LSIR is changed, it propagates the information to the neighbors according to the information in the NAIR to update the cached service information of other Ad-UDDI servers. When updating requests come, it forwards the request to the Cached Service Information Manager for updating.

Diffusing Querier propagates the service querying requests to neighbors**.**

4. Performance Evaluation

To evaluate the performance of the Ad-UDDI approach, we coded a simulator using Java, in which a certain number of Ad-UDDIs, service providers and requestors are connected to form a mesh network to simulate the situation of Internet.

We use BRITE [16] to generate topologies up to 2,000 nodes with random connection. The network delay between every two nodes is calculated according to the shortest path along the physical network topology. Each service is remarked by its name, key, version, type, access point, etc. In each run, a number of services with diversified types are deployed into the network.

Each Ad-UDDI in the simulation is able to register the service information in several industries, while every industry classification can be registered into several Ad-UDDIs. We distribute the Ad-UDDIs into finite industries and publish the services into Ad-UDDIs based on their types. The root is a special Ad-UDDI node, which only registers the information of the Ad-UDDI services without receiving the publication of business services. On the other hand, we simulate UDDI as a centralized registry without active monitoring method and all services publish their information to it. In this section, we introduce our performance metrics, and then the simulation results.

4.1 Performance Metrics

The basic function of the Ad-UDDI is to find available web services matching requestors' demands. To better evaluate the Ad-UDDI design, we use the following metrics: *available rate*, *success rate*, *average response time*, and *total traffic cost*.

The *Available Rate* is defined as the ratio of the requests which successfully find desired and available services at the first return out of all requests. In real B2B environment, the service requestor tends to use the service information directly from the service registry, so the invalidity of discovered service information is very likely to cause the crash of B2B applications. Therefore, the available rate is an important metrics in B2B applications.

The *Success Rate* is defined as the ratio of the requests which successfully find desired and available services out of all requests.

The *Average Response Time* is defined as the average time elapsed from the issuance of a query till a desired and available service is found. If no appropriate service is found, the query ends after searching all candidate services which have the same service type with the request.

The *Total Traffic Cost* is defined as the traffic of messages incurred by queries and responses. The traffic of monitoring and updating for the Ad-UDDI is also considered.

4.2 Results

In the first simulation, we apply the active monitoring mechanism, where 1,000 services are distributed into randomly selected nodes. We set 10 Ad-UDDIs as the registries with 5 industry classifications. We generate 10,000 requests every 3 days to trace the evolution of the available rate of the queries. The results in Fig. 8 show that the available rate of information in the registry without active monitoring mechanism drops to a very low level after 30 days. With the help of AD-UDDI, the available rate stays in a relatively high level.

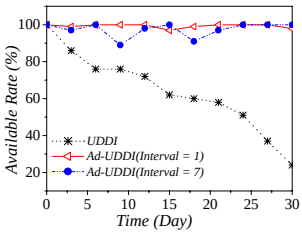

Fig.8. Available rate v.s. time with 1000 services

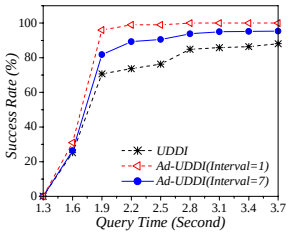

Fig.9. Success rate v.s. Query Time

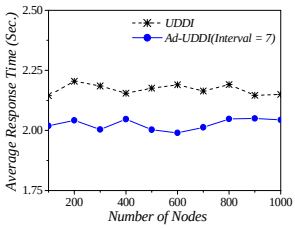

Fig.10. Response time v.s. number of services

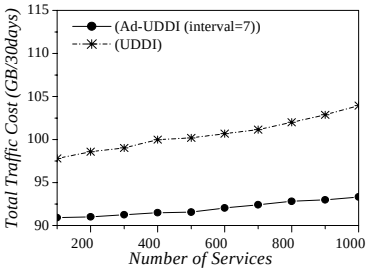

Fig.11. Total traffic cost v.s. number of services

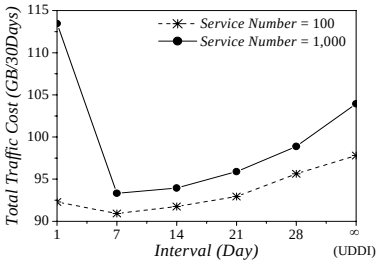

Fig.12. Total traffic cost v.s. interval

We implement the second simulation to analyze the response time distribution of the requests. The service number and the Ad-UDDIs number are the same as in the first simulation. We disperse 10,000 requests in 30 days and record their response time. Figure 9 plots the success rate against average response time. With an *interval* of active monitoring is 1 day, 96% requests get available services within 1.9 seconds. Without Ad-UDDI design, only 69% requests can get the available ones within such time period, and more than 15% requests never find available ones. Larger monitoring interval leads to longer response time, but smaller query overhead. Figure 10 plots the response time against system size. The results show Ad-UDDI design is scalable when the number of nodes increases.

We then explore the total traffic cost with different service numbers by recording the cost in 30 days with 10,000 requests. According to Fig. 11, the total traffic cost is slightly increased with larger number of services. With the same number of services, the query traffic with Ad-UDDI is much smaller than without active monitoring. In Fig. 12, we show the relationship between the total traffic cost and the monitoring interval with 100 and 1,000 services involved respectively. If we set the monitoring interval as 1 day, there will be a lot of monitoring cost. On the other side, without monitoring, we save the monitoring messages but more services have to be checked in order to find an available service, which means the traffic cost of queries will increase. There is an obvious trade-off between monitoring and query traffic.

Combined with Fig. 8, shorter interval between two monitoring process leads to higher available rate, but brings larger monitoring traffic cost, as shown in Fig. 12. We can conclude that the weekly monitoring is a good balance between available rate and the traffic cost.

5. Conclusion

Aiming at resolving the low validity of the public UDDI, we propose an active and distributed registry, Ad-UDDI, to provide available service information. In this design, the service information is distributed among multiple registries and thus the single point of failure and bottleneck in one public UDDI is reduced. In our approach, the root registry takes charge of managing the Ad-UDDI services without any business services, so the burden of root registry is lightened. The distributed architecture of Ad-UDDI may serve as a basic method of connecting the private or semi-private UDDIs. With the active monitoring mechanism, the real-time availability of the service information in the Ad-UDDI is significantly improved.

6. Acknowledgement

We thank the anonymous referees for their constructive comments. This work was supported in part by China NSFC 90412011, by Hong Kong RGC Grants DAG 04/05 EG01, and by Microsoft Research Asia.

References

1. M. Luo, M. Endrei, P. Comte, P. Krogdahl, J. Ang, and T. Newling. Patterns: Service_ Oriented Architecture and Web Services. http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246303.html. 2004.
2. D. Booth, H. Haas, and F. McCabe. Web Services Architecture. http://www.w3.org/TR/ws-arch/. 2004.
3. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language(WSDL) 1.1. http://www.w3.org/TR/wsdl. 2001.
4. Z. Du, J. Huai, Y. Liu, C. Hu, and L. Lei. IPR: Automated Interaction Process Reconciliation. in Proceedings of the International Conference on Web Intelligence (WI 2005). 2005.
5. L. Clement, A. Hately, C.v. Riegen, and T. Rogers. Universal Description Discovery & Integration (UDDI) 3.0.2. http://uddi.org/pubs/uddi_v3.htm. 2004.
6. S.M. Kim and M.-C. Rosu. A Survey of Public Web Services. in Proceedings of the 13th International Conference on the World Wide Web (WWW'04). 2004.
7. M. Cai and M. Frank. RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network. in Proceedings of the 13th International Conference on World Wide Web (WWW'04). 2004.
8. W. Hong, M. Lim, E. Kim, J. Lee, and H. Park. GAIS: Grid Advanced Information Service based on P2P Mechanism. in Proceedings of IEEE International Symposium on High Performance Distributed Computing 2004 (HPDC 2004). 2004.
9. L. Xiao, X. Zhang, and Z. Xu. On Reliable and Scalable Peer-to-Peer Web Document Sharing. in Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002). 2002.
10. W. Nagy, F. Curbera, and S. Weerawaranna. The Advertisement and Discovery of Services (ADS) protocol for Web services. http://www-128.ibm.com/developerworks/library/ws-ads.html?dwzone=ws. 2000.
11. A. ShaikhAli, O.F. Rana, R.J. Al-Ali, and D.W. Walker. UDDIe: An Extended Registry for Web Service. in Proceedings of Symposium on Applications and the Internet Workshops (SAINT Workshops 2003). 2003.
12. K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller, METEOR–S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. Journal of Information Technology and Management, 2005.
13. A.A. Patil, S.A. Oundhakar, A.P. Sheth, and K. Verma. Meteor-s: Web Service Annotation Framework. in Proceedings of the 13th International Conference on World Wide Web (WWW 2004). 2004.
14. M.P. Papazoglou, B.J. Kramer, and J. Yang. Leveraging Web-Services and Peer-to-Peer Networks. in Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAiSE2003). 2003.
15. GICS Structure and Sub-Industry Definitions. http://www.msci.com/equity/GICS_map2005.xls. 2005.
16. A. Medina, A. Lakhina, I. Matta, and J.W. Byers. BRITE: An Approach to Universal Topology Generation. in Proceedings of the 9th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'01). 2001.