Binding Now or Binding Later: The Performance of UDDI Registries

M. Brian Blake
Georgetown University
Washington, DC, USA
mb7@georgetown.edu

Amy Lynn Sliva
University of Maryland
College Park, MD, USA
asliva@umiacs.umd.edu

Michael zur Muehlen
and Jeffrey V. Nickerson
Stevens Institute of
Technology
Hoboken, NJ, USA
{mzurmuehlen,jnickerson}
@stevens.edu

Abstract

In service-oriented environments, the fluidity of the marketplace introduces changes in service offerings and subsequent connection failures for users still bound to outdated services. One focus of web services research is the real-time acquisition of new capabilities via service discovery using Universal Description, Discovery, and Integration (UDDI) registries. Another equally valuable usage of UDDI registries, as addressed in this paper, is the real-time assurance of service responsiveness. UDDI registries can be incorporated into business process execution routines to assure that the underlying services are active at operations time. In this paper, several UDDI modes of operation are evaluated through performance tests of different UDDI implementations. The results of the investigations can be applied as a decision aid for organizations to choose the most efficient utilization of UDDI for management responsiveness in their own service-oriented processes.

1. Introduction

Emerging web services standards are intended to speed the expansion of electronic marketplaces [6] [11]. These standards are promising because they will provide a way to increase the level of automation for contracting and purchasing. Their promise also brings about a new set of technical challenges. In order to automate the composition of varied services, the style of a program in a services environment needs to be different from a program in a conservative environment. Researchers have pointed out that programs need to bind late, or just-in-time [2][3]. Late binding is a technical necessity for simple business reasons; contracts will need to be filled in short amounts of time, and the openness of the marketplace will make it difficult to predict far ahead of time who will be able to best fulfill a particular request.

While late binding has been implemented within programming languages, it is more complex to implement in service environments, as performance is more likely to vary, transactions are more likely to be long, and the search criteria for services discovery are likely to be complex. In situations where services are not very reliable, or if their location changes frequently, late binding may be more desirable than in situations where the services are somewhat reliable and their location is fixed. Therefore, when late binding is significantly expensive, organizations must trade-off against the alternative: binding early, which helps mitigate the risk of poor performance at operations time.

In this paper, the major contribution is a method for evaluating operational modes, using UDDI technologies, to assure the validity of pre-existing service bindings at run-time. When new job requests are received, the organization can reaffirm the validity of all services prior to initiating the business process (*pre-process validation*) or the organization can choose to only acquire new service information only after a connection failure occurs (*connection-time validation*). We acknowledge that there are numerous variations to these operational modes, but these two modes represent two general notions, thus they are the focus of the following studies. Considering the overhead associated with concurrently accessing the UDDI registry and the fact that not all services will need new bindings, the studies in this paper suggest that an informed choice of operational mode can be made through an analysis of the local domain characteristics.

The paper proceeds in the following way. First we discuss related work in the area of UDDI. Then we describe the benchmarking experiments used to characterize the behavior of current UDDI implementations. In section four, we present the results of simulation-based experiments and discuss a decision support aid for configuring reliable business process execution systems.

2. Background and Related Work

Version 3 of the UDDI specification supports web services: its implementations are distributed on-line databases of web services descriptions [9]. Specifically, UDDI registries support the management of meta-information that describes particular web services. This meta-information is generally represented in the Web Service Description Language (WSDL) [21]. Service-oriented computing [16] supports the development of new business processes through the automatic or semi-automatic discovery and composition of web services (also referred to as services orchestration). UDDI can be viewed as the proposed discovery mechanism, the directory, for web services. Semantic web researchers point out that UDDI does not fully solve the semantic issues, and suggest ways of incorporating semantic description into UDDI [12][13].

Although researchers have investigated technologies that directly support the composition of web services (e.g. [7][5][17]) and there are a large number of studies that evaluate general database performance, there are far fewer that consider UDDI-specific registries. Miles et al. [15] measured the responsiveness of a UDDI registry with respect to developing personalized grid services. Metso [14] also conducted experiments that directly assess UDDI performance. Metso focuses on the performance as the UDDI repository stores an increasing number of web service descriptions. He discovered that the UDDI registry degrades considerably as the number of entries increase. Our study considers the degradation that occurs as the number of concurrent requests increases. In our work, this degradation is further used as a factor in determining the best integration of UDDI registries in business process management frameworks.

Adams, Gisolfi, Snell and Varadan [1] and Domanski [10] also see the UDDI registry as a potential source of performance problems, particularly those problems associated with message transmission and parsing. They suggest that web service applications cache information from UDDI registries to minimize UDDI requests. Such caching software can use the HTTP 200 error as a trigger to dynamically search for new information when web services information changes. This solution helps to minimize unnecessary traffic but does not investigate how the registry should be utilized for reliability. Chen, Liang-Tien, and Bu Sung [8] introduce a supporting architecture and implementation to UDDI frameworks. This architecture stores historical measurements on web services performance and combines them with UDDI functionality to predict and suggest the most efficient services as new external requests come in. This work complicates the UDDI implementation effort, but helps find the underlying services that perform better.

In another related study, just one UDDI implementation was evaluated [18]. The results were incorporated into a simulation to predict UDDI's scalability. In this paper, we extend the breadth and depth of the earlier studies by considering multiple concurrent UDDI registry implementations. In addition, these studies are used as decision support aids for organizations planning to develop systems that incorporate UDDI directory services to assure reliability.

3. Understanding UDDI Performance: A Prerequisite for System Analysis

In order to evaluate the usage of UDDI reliability scenarios, it is first important to understand the unique nature of these technologies. We experimented with two open source registries, jUDDI and Java Web Services Development Pack (JWSDP). jUDDI is a Java-based implementation of UDDI that was created to integrate effectively with the Tomcat web server [4]. JWSDP integrates with Tomcat as well but follows more closely Sun Microsystems' Java-based suite of tools [20]. jUDDI uses an underlying MySQL database, while JWSDP uses a Java proprietary database.

The purpose of the experimentation was to determine how UDDI implementations perform under regular conditions and under conditions of heavy concurrent requests. Ultimately, we incorporate the real measures of performance into simulation software that we used to evaluate modes of UDDI operation in Section 4. We focused on the most common functions, *inquiry* (read) and *publication* (publish). There were two experiments performed on a 1.5 Gigahertz, Pentium 4, Dell workstation with 1 Gigabyte of RAM. The first experiment was designed to determine the baseline performance of the UDDI frameworks by measuring the speed of common registry tasks, specifically the inquiry and publication functions, in an optimal environment (i.e., with no other registry traffic). The second experiment determined the performance of UDDI implementations under conditions of concurrent traffic.

In the first experiment, the publication and inquiry functions were measured under normal operations without additional traffic. The registry was populated with approximately 100 entries. The inquiry (read) function was executed 10 times sequentially and the average service time was recorded. Likewise, the average service time over 10 runs was recorded for the publication function. jUDDI's inquiry function executed at an average of 40 ms over 10 invocations, and JWSDP at 156 ms over 10 invocations (For JWSDP the first measure was not included considering the significant connection overhead on the first request. The publication function has an average service time of 453 ms for jUDDI and 247 ms for JWSDP over 10 invocations. In all tests, variance was less than 10% of the respective mean. We anticipated that the publication function would require more time based on the underlying database commit that must take place. These baseline measures for jUDDI inquiry and publication are displayed in Figure 1 and 2; Figure 3 and 4 contain the data for JWSDP inquiry and publish.

The second experiment examined the performance of the UDDI frameworks with an increasing concurrency of processes (i.e., large numbers of requests, large numbers of concurrent service changes, etc.). The same performance tests were run, this time with additional Java programs simulating concurrent registry operations. The purpose of these tests was to determine the relative performance of the registry functions in conditions of varying traffic. Specifically, the inquiry and publication functions were evaluated with other concurrent publication and inquiry traffic. There were 4 cases of concurrent traffic, 1 inquiry/second, 1 publication/second, 1 inquiry/second and 1 publication/second, and 2 inquiry/second and 2 publication/second.

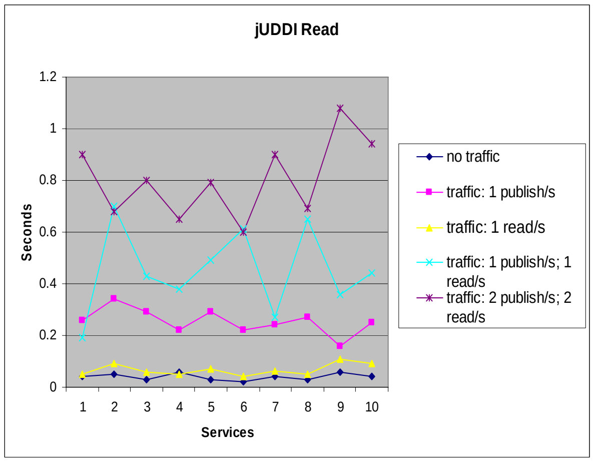

Figure 1. jUDDI Inquiry Benchmarks

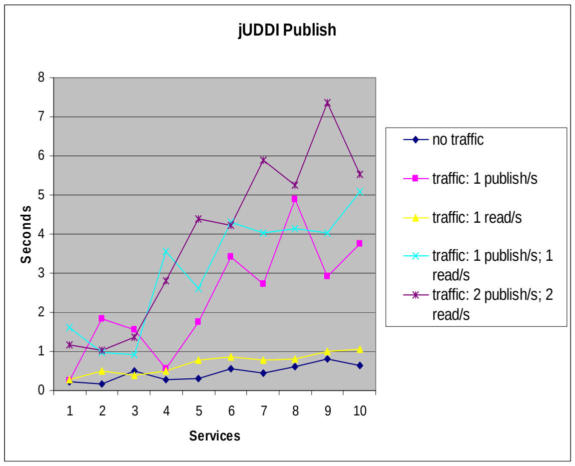

Figure 2. jUDDI Publish Benchmarks

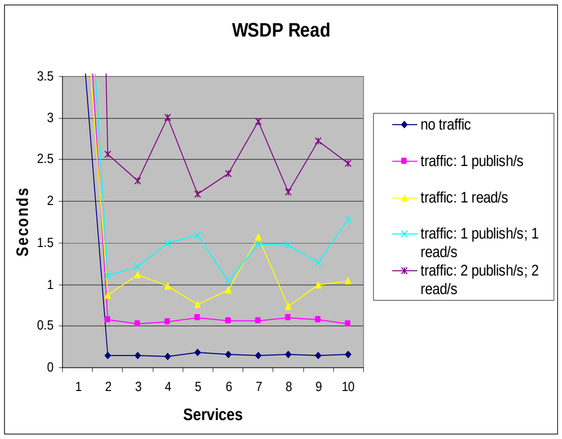

Figure 3. JWSDP Inquiry Benchmarks

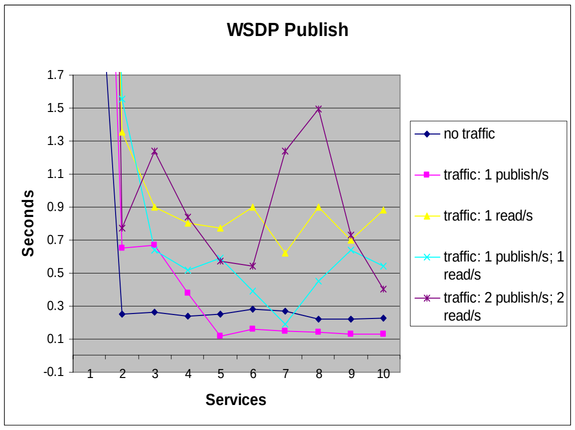

Figure 4. JWSDP Publish Benchmarks

The specific performance results are shown in Figures 1, 2, 3, and 4. Several trends and variations were determined from the initial experimentation. Trends across multiple UDDI registries are helpful in determining standard operational modes that may increase the general performance of *any* registry.

Variations in multiple registries help to determine attributes that can be determined and used only for certain registries in certain domains. Table 1 summarizes anticipated results while Table 2 and 3 summarize the findings useful to our optimization framework.

Table 1. Anticipated Results in UDDI Operations.

All registries perform significantly worse as more concurrent requests are delivered.
Performance fluctuates more severely as concurrent traffic increases.

Table 2. Findings: Trends Common to all Operations.

Concurrent heterogeneous traffic is more expensive than concurrent homogeneous traffic.
At certain times in the operation, heavier loads may perform better than lighter loads.

Table 3. Findings: Variations in the UDDI Operations.

A significant connection delay is associated with the first request in the JWSDP registry but not seen in the jUDDI registry.
Overall, jUDDI handles inquiries more efficiently than publications and JWSDP handles publications more efficiently than inquiries.
Publication traffic directed toward the jUDDI registry causes the registry to degrade as the number of requests continue over time at the maximum rate for one client. All other operations for both registries tend to perform consistently with no upward trend.

The most important result is the last one: the degradation of inquiry/publication service times as concurrent requests are executed.

4. Experimentation with Business Process Reliability Routines

We experimented with two general operational modes for incorporating UDDI into service-oriented business process management systems for reliability. The subsequent sections describe both operational modes in detail.

4.1 Pre-Process Validation and Connection-Time Validation

The first general model of operation for assuring reliability in business process management systems using UDDI is to configure the system to check all of the underlying services once a new job is received (pre-process validation). For this configuration, the system follows steps 1, 1a, and 2 (Figure 5). In step 1, a consumer requests a new job, the provider captures the job and starts a new business process. In step 1a, the provider confirms that all current services for the instantiated business process are still viable. This confirmation is based on the assumption that a service listed and valid in the provider's UDDI at the beginning of the job remains valid until the service is actually requested. The purpose of this step is not the discovery of new services, but just the validation of pre-established services. Finally, step 2 is the enactment of the services.

The second general model of operation is based on internal failure triggers during service invocation, rather than a preemptive viability check of all services. The operational mode for connection-time validation accesses the UDDI registry for services that have problems (i.e., after a connection error). This approach minimizes the need to search for the address of every service, but causes searches only for services that have changed locations. However, the service time for an individual service is increased because the new service time must include the time required to determine that a connection error has occurred. The process for connection-time validation is also illustrated in Figure 5, as the sequence of steps 1, 2, and 2a.

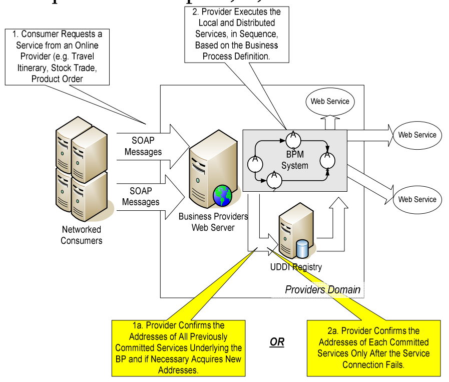
1. Consumer Requests a Service from an Online Provider (e.g. Travel Itinerary, Stock Trade, Product Order
2. Provider Executes the Local and Distributed Services, in Sequence, Based on the Business Process Definition.
Web Service
SOAP Messages
SOAP Messages
BPM System
Web Service
Web Service
Networked Consumers
Business Providers Web Server
UDDI Registry
Providers Domain
1a. Provider Confirms the Addresses of All Previously Committed Services Underlying the BP and if Necessary Acquires New Addresses.
OR
2a. Provider Confirms the Addresses of Each Committed Services Only After the Service Connection Fails.

Figure 5. Two Modes of Operation with UDDI.

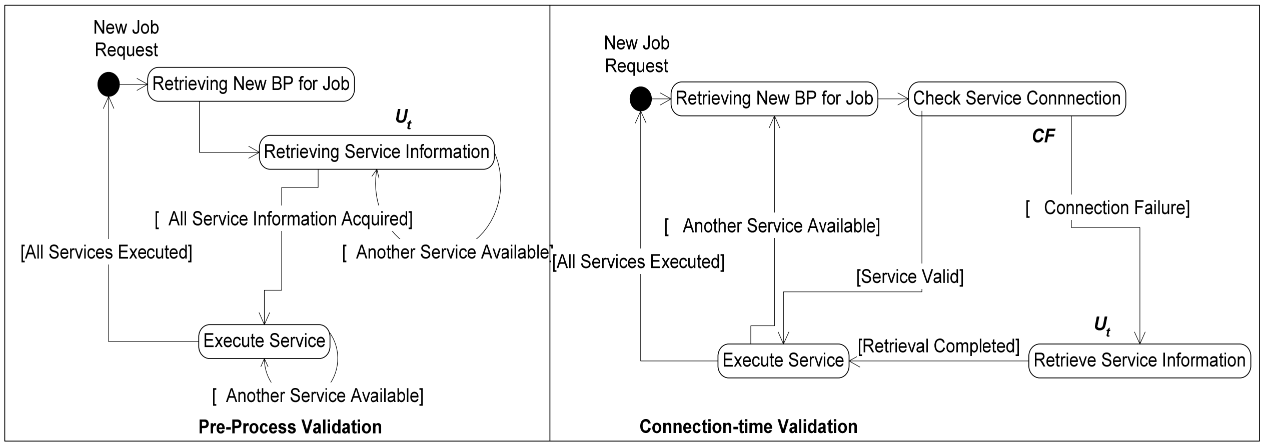
New Job Request
Retrieving New BP for Job
U_t
Retrieving Service Information
[All Service Information Acquired]
[All Services Executed]
[Another Service Available]
Execute Service
[Another Service Available]
Pre-Process Validation
New Job Request
Retrieving New BP for Job
Check Service Connnection
CF
[Connection Failure]
[Another Service Available]
[All Services Executed]
[Service Valid]
U_t
[Retrieval Completed]
Execute Service
Retrieve Service Information
Connection-time Validation

Figure 6. State Diagrams of Validation Modes.

4.2 Defining Service Times for Two UDDI Operational Modes

In evaluating multiple operational modes, this study is concerned with the service time specific to the UDDI operations. As such, we define the service time for a specific service inquiry request as STR. We have developed a specialized UDDI service time function that was created running performance evaluations on the UDDI implementation (which is discussed in Section 3). Therefore, at time, *t*, the UDDI function, U_t, generates the service time, ST_R , considering the sum of incoming UDDI requests, i_t, and the sum of active jobs, q_t, being processed concurrently on a specific UDDI registry. The service time assigns service times to the new requests while active requests are considered part of the working queue. The service time modules uses the behavior discovered based on concurrency for new requests and baseline operational performance measures for active jobs (Section 3).
The service time for the operation model for pre-process validation is defined as

$$ST_R = U_t(\sum i_t, \sum q_t)$$

The service time for the operation mode for connection-time validation is similar to the service time for pre-process validation. The only difference is that the service time for a connection failure, *CF*, must also be added to the total UDDI operation time. Therefore the new service time, ST'_R , can be defined as

$$ST_R = U_t(\sum i_t, \sum q_t) + CF$$

The operations of the business process management system can be generally modeled in a Unified Model Language (UML) statechart diagram. Figure 6 contains two statechart diagrams illustrating both modes of operations annotated with the expressions defined in the above relations.

4.3 Experimentation

Several experiments were executed using a service-oriented simulation framework created by the authors, and described in detail in [19]. The simulation software consists of three components. A traffic generation component which generates service-oriented process requests in various distributions. The simulation component processes the results used for decision support. The simulation component has an internal service time module, as described in Section 4.2, that calculates service time as a result of concurrent requests. The simulation component is configured with a variable amount of UDDI emulator components. These UDDI emulators are similar to queues, but their responsiveness are configured based on benchmarks from the UDDI performance experimentation.

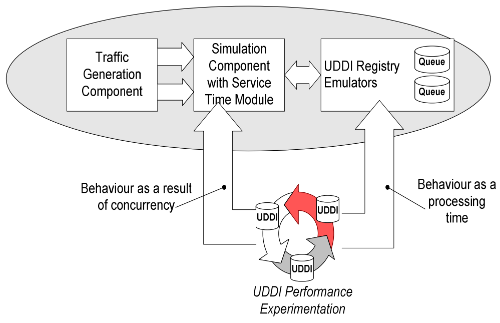
Traffic Generation Component
Simulation Component with Service Time Module
UDDI Registry Emulators
Queue
Queue
Behaviour as a result of concurrency
Behaviour as a processing time
UDDI
UDDI
UDDI
UDDI Performance Experimentation

Figure 7. Overview of the Simulation Software.

Experiments were executed to demonstrate situations where one operational model is generally more efficient than the other. In evaluating the aforementioned operational modes, each of the modes was exercised by varying three different aspects: traffic composition, percentage of service changes, and connection failure time. In the experiments, we considered a finite number of job requests (i.e., 100 job requests) independent of the traffic composition. The traffic composition was varied such that the first traffic scheme was consistently light and steady. In this scheme, 1 job request is transmitted for each time cycle. Another traffic scheme had a relatively high level of concurrency. Ten requests were transmitted at once regularly throughout the simulation. The final traffic was delivered using a Poisson distribution.

The percentage of service changes were varied independently. The overall percentages of services that change throughout the entire simulation were varied at 33%, 66%, and 100%. At 100%, all services change each time. Finally, the connection failure time was also varied relative to the UDDI service time. The connection time was varied to be 25%, 50% and 100% of the UDDI service time.

Considering the variations, both operational modes were simulated in five sets of experiments. In this paper, we do not present all of the numerical findings considering the fact that the actual numbers would vary depending on machine and process schema. However, in Figure 8, we display the findings generally comparing the merits of both operational modes. The non-shaded cells represent that the connection-time validation mode has the lowest average service time per business process. The pre-process validation mode has the lowest average service time in the shaded cells. Mixed cells are borderline cases.

5. Discussion

As anticipated, the factor that has the most impact on the choice of modes is the percentage of service changes. In all tests, when the percentage of service changes was less than 59%, the connection-time validation was most efficient. Although the exact percentage varied, typically, when the service changes were between 59% and 70% of all services, the pre-process validation mode was most efficient in completing business processes. In all experimental cases, the pre-process validation mode was more efficient when the percentage of changes were greater than 70%.

The experimentation also suggests that the connection-time validation mode performs better when the traffic is steady. Steady traffic is more beneficial to the connection-time mode than highly concurrent and Poisson-distributed traffic. With steady traffic, the connection-time mode has reduced concurrency across processes. Steady or concurrent traffic does not significantly improve the pre-process validation mode because, in this mode, the service requests are sent in concurrently regardless of the traffic composition. Finally, when the connection failure time is high, the percentage of service changes required to make the pre-process validation mode more efficient than the connection-time validation mode is reduced.

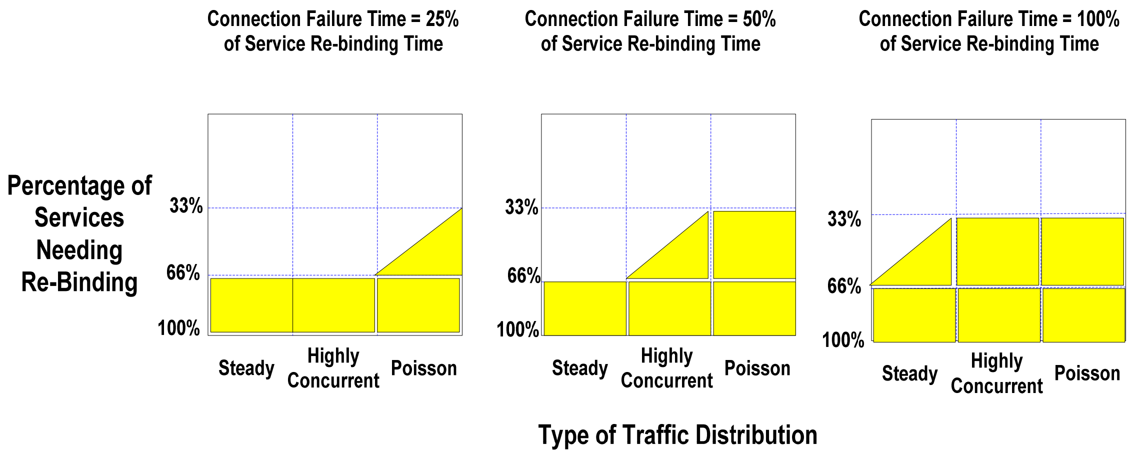

Figure 8. Decision Matrix (*Shaded quadrants represent areas where pre-process validation should be instituted*)

6. Conclusion

In this paper, we present a method for evaluating modes of operation in a service-oriented computing environment. In this approach, we benchmarked current UDDI implementations and incorporated the resulting performance measures into software simulation. Using this simulation, we evaluated, side-by-side, two modes of operation that support the use of UDDI to assure the reliability of services in a business process enactment scenario. Results show that understanding the amount of requests, the overhead associated with connection failures, and the nature of the traffic can define the strategy that organizations might take when using UDDI registries in this fashion. An innovation in this work is the methodology for combining these factors to make operational decisions.

Acknowledgements

The investigations in this paper were supported by the Computing Research Association's (CRA) Collaborative Research Experience for Women (CREW) in Undergraduate Computer Science and Engineering fellowship. In addition, the authors wish to acknowledge the support by the National Science Foundation under grant number 0326309.

References

[1] Adams, H., Gisolfi, D., Snell, J., & Varadan, R. (2004). Best Practices for Web services. IBM. Accessed (2006) http://www-106.ibm.com/developerworks/webservices/library/ws-best1/

[2] Andrade, L., & Fiadeiro, J. (in print). Composition Contracts for Service Interaction. *Journal of Universal Computer Science.*

[3] Andrade, L., Fiadeiro, J., Gouveia, J., Koutsouko, G., & Wermelinger, M. "Coordination for Orchestration". *Proceedings of the 5th International Conference on Coordination Languages and Models*. 2002

[4] Apache. (2004). jUDDI, http://ws.apache.org/juddi/jUDDI, Apache Project.

[5] Blake, M.B. "Coordinating Multiple Agents for Workflow-Oriented Process Orchestration". *Information Systems and E-Business Management*, 1(2).

[6] Benatallah, B., Dumas, M., Sheng, Q. Z., & Ngu, A. H. H. "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services". *Proc of the IEEE 18th International Conference on Data Engineering*. 2002

[7] Benatallah, B., Sheng, Q., & Dumas, M. "The Self-Serv Environment for Web Services Composition". *IEEE Internet Computing*, 7(1), 40-48. 2003

[8] Chen, Z., Liang-Tien, C., & Bu-Sung, L. QoS-Aware and Federated Enhancement for UDDI. *Int. Journal of Web Service Research*, 1(2), 58-85. 2004

[9] Clement, L., Hately, A., Riegen, C. v., & Rogers, T. (2004). UDDI Version 3.0.2 (Technical Committee Draft, Dated 20041019). http://uddi.org/pubs/uddi_v3.htm

[10] Domanski, B. (2003). An Introduction to Web Services and Performance Issues. *zJournal*, December.

[11] Leymann, F., & Roller, D. (2002, 01 August 2002). Business processes in a Web services world. A quick overview of BPEL4WS. Retrieved 02-28, 2005, from http://www-106.ibm.com/developerworks/webservices/

[12] Massimo, P., Takahiro, K., Terry, R. P., & Sycara, K. "Importing the Semantic Web in UDDI". In C. Bussler & R. Hull & S. McIlraith & M. E. Orlowska & B. Pernici & J. Yang (Eds.), Lecture Notes in Computer Science (Vol. 2512). 2002

[13] Medjahed, B., Bouguettaya, A., & Elmagarmid, A. K. (2003). Composing Webservices on the Semantic Web. *VLDB Journal*., 12(4), 333-351. 2002

[14] Metso, J. Suitability of UDDI Registry for web-Pilarcos Architecture-Performance Measurements, Report C-2003-72: University of Helsinki, Finland. 2003

[15] Miles, S., Papay, J., Dialani, V., Luck, M., Decker, K., Payne, T., & Moreau, L. Personalized Grid Service Discovery. *Proceedings of the Nineteenth Annual UK Performance Engineering Workshop (UKPEW'03)*, University of Warwick, Conventry, England.2003

[16] Papazoglou, M. P., & Georgakopoulos, D. Service Oriented Computing. Communications of the ACM, 46(10). 2003

[17] Peltz, C. Web Services Orchestration and Choreography. IEEE Computer, 36(10), 46-52. October 2003

[18] Saez, G., Sliva, A.L. & Blake, M.B. "Web Services-Based Data Management: Evaluating the Performance of UDDI Registries". P*roceedings of the International Conference on Web Services, (ICWS 2004)*, San Diego, CA. 2004

[19] Sliva, A.L. (2005). "Characterizing Data Management Approaches for Service-Oriented Computing", Undergraduate Thesis, Department of Computer Science, Georgetown University

[20] Sun. (2006). Web Services Developer Pack (JWSDP). http://java.sun.com/webservices/jwsdp/index.jspRetrieved, from the World Wide Web:

[21] W3C. (2006). Web Services Description Working Group. http://www.www.org/2002/ws/desc/