Binding- and Port-Agnostic Service Composition using a P2P SOA

Dominik Dahlem, David McKitterick, Lotte Nickel, Jim Dowling, Bartosz Biskupski, and René Meier

Distributed Systems Group, Trinity College Dublin
{dominik.dahlem, david.mckitterick, nickell, jim.dowling, biskupski, rmeier}@cs.tcd.ie

Abstract The assumption of the availability of port information at design time of service compositions in Service-Oriented Architectures (SOAs) is not valid for an increasing number of hosts on the Internet that do not have a public, static IP address. Existing workflow engines do not support services deployed on such hosts, as service invocations require the availability of port information defined either in concrete WSDL definitions or within a deployment descriptor of the BPEL workflow engine. This paper presents a workflow engine that supports runtime look-up of service endpoints based on a P2P middleware. Using a service identifier based on a DHT identifier, Service Proxy objects that encapsulate port information are downloaded over the structured P2P network from the host where the service is deployed. A Service Proxy delegates service invocations to an abstract protocol adaptor framework that uses dynamic invocation mechanisms to provide a protocol-independent execution of remote services, e.g., over GIOP/IIOP or SOAP. This allows us to specify binding- and port-agnostic service compositions in BPEL using abstract WSDL and our service identifiers. To validate our approach, we extended the ActiveBPEL workflow engine to support the discovery and consumption of services using our P2P middleware and the abstract protocol adaptor.

1 Introduction

One goal of Service-Oriented Architectures (SOAs) is to integrate and compose services that are deployed on heterogeneous middleware paradigms, e.g., CORBA, RMI, and J2EE. Web Services standards have been introduced to provide a language and platform-independent way of describing and invoking services on these platforms, and techniques for the composition of Web Services have emerged, such as the Business Process Execution Language (BPEL) [1,2]. Approaches have been developed to assist in the design and deployment of service compositions, such as semantic [3] or model-driven [4] techniques that help search the space of available services. However, they are based on middleware that requires the availability of binding and port information at deployment time.

In this paper, we present a strategy where service endpoints are looked up at runtime by an adapted workflow engine to support the binding to services deployed on hosts that do not have a public, static IP address. Additionally, the presented client-side programming model is unaware of the actual interaction protocol and thus supports late binding using a protocol adapter framework configured at runtime. This allows us to support the deployment of services on hosts with DHCP-allocated IP addresses and, using Relay Peers, hosts that reside behind a Network Address Translation (NAT) Gateway.

The SOA and workflow engine described in this paper have been designed as part of the Digital Business Ecosystem (DBE) project[1], which aims at providing a SOA based on a P2P architecture for use by small-to-medium sized enterprises (SMEs). We identified the following requirements for the DBE SOA:

- Support for port-agnostic service compositions using service addressing and discovery mechanisms that do not require higher cost public IP addresses;
- Dynamic invocation mechanisms to promote loose coupling between consumers and the service invocation protocol;
- Enable service providers to advertise their services independent of the underlying paradigm (RMI, CORBA, J2EE, etc.) without wrapping the service into a Web Service container;

This paper focuses on service composition using BPEL which is built on top of Web Services Definition Language (WSDL) [5]. Abstract WSDL and BPEL together can enable service compositions to be defined independent of binding and port details. We implemented a Service Proxy framework that allows binding and protocol information for services to be encapsulated into objects that can be downloaded at runtime to access the service. Service Proxy objects use an Abstract Protocol Adaptor (APA) that enables the decision on which protocol to use to be delayed until runtime when the Service Proxy object is downloaded. Service Proxy objects are located using a service identifier that is based on a Distributed Hash Table (DHT) identifier in a structured P2P network. The P2P network is based on Bamboo [6] and all peers active in the system, including those without public, static IP addresses, can be located using the DHT identifier. Service identifiers can be discovered in a Service Registry. Finally, we extended an open source BPEL workflow engine, ActiveBPEL [7], to integrate it with our implementation of the P2P network, Service Proxy and APA frameworks.

2 Background

Service composition is a quite general term which is often used in relation to Web Service composition, because Web Services can be viewed as a service integration architecture covering many standards, languages and frameworks. BPEL [2] has become the accepted standard language for executable business processes defined by WSDL interfaces. It defines a process-centric model for the formal specification of the behaviour of business processes based on the interaction of the executable process and its partners [8]. A key aspect of BPEL is the notion of

[1] http://www.digital-ecosystem.org

partner links. Partner links describe the relationship between two services at the interface level, by providing both a link to services that are invoked by the process and also to clients which are invoking the process. The BPEL process itself knows only about partner links and is unaware of the underlying dependency on the concrete WSDL bindings and service endpoints. The BPEL specification does not require a static declaration of port-specific data, because an endpoint reference element allows for explicit dynamic assignment of service endpoints during the execution of a BPEL process [9]. Implementations of existing BPEL engines require concrete port and binding definitions to be available at deployment time, usually SOAP bindings and static endpoints. BPEL engines like BPWS4J [10] and ActiveBPEL [7] do not support the internal assignment or selection of service endpoints within their implementations when executing BPEL workflows so therefore this assignment must be defined by the BPEL developer.

Service discovery and selection have already been recognised and tackled by several research groups by introducing semantics into the process of designing and executing BPEL workflows. The METEOR-S project [11,12,13,14] provides tools to enable automatic service discovery and selection at deployment time based on constraints, which then uses the BPWS4J engine to execute static BPEL workflows. Service compositions are specified in BPEL but the required Web Services are described in service templates [11]. Service templates define a semantic description of a service [15] as well as constraints for service selection like user's preferences for service partners, Quality of Service (QoS) requirements, and service dependencies. An enhanced Universal Description, Discovery and Integration (UDDI) registry is queried with the service templates for matching service descriptions [16]. Matching service descriptions are further analysed if they meet the specified constraints [13]. Finally the best matching service is selected and with late-binding an executable BPEL process generated at deploy-time. Another approach to overcome the static definition of services in a BPEL workflow comes from work done by the OWL-S research community [17]. They introduce a Semantic Discovery Service (SDS) that serves as a dynamic proxy between the BPWS4J engine and services [18]. Instead of invoking requests on statically bound services, the BPWS4J engine routes requests to the SDS. Services are described, advertised, and discovered via an OWL-S service profile. The SDS finds a matching service (or chain of services) and invokes the endpoint at runtime. Upon receiving a reply from the partner the SDS forwards the response back to the BPWS4J engine. Synthy [19] decouples a Web Service composition into logical and physical design stages. The logical stage establishes a relationship based on semantic annotations between candidate services for a composition while the physical stage is responsible for creating one or more concrete BPEL workflows based on QoS parameters relevant for this workflow. Additionally, Synthy adds a fault-tolerant level to workflow execution. Multiple workflows can be created at the physical design stage and deployed with the runtime engine. These executable workflows can be ranked according to QoS metrics. In case of a runtime failure in a workflow the runtime engine can select the next ranking workflow to be executed.

However, these projects do not address the look-up of endpoint information at runtime for a known service. There has been some existing work on the Web Services Invocation Framework (WSIF) [20,21] to enable delaying the decision on which binding to use until the service is executed, although it requires that multiple bindings are specified in advance with one of them chosen at runtime. Consequently, WSIF does not address the issue of services deployed on hosts that do not have a public IP address. The next section introduces our approach based on a structured P2P network and Service Proxies.

3 DBE SOA and the Workflow Engine

3.1 The DBE SOA

This section gives a brief overview of how service proxies, identified by a persistent and unique ServiceID, are discovered and downloaded using a structured P2P network. The DBE SOA provides two services to enable service registration/discovery and service binding, respectively. Firstly, a Service Registry is used to register and discover ServiceIDs along with their associated abstract WSDL. Secondly, a structured P2P network based on a Distributed Hash Table (DHT) architecture [6] is used to download a Service Proxy that encapsulates binding and port information for the service.

The Service Registry is currently a centralised component that provides service registration and look-up operations for both abstract WSDL and ServiceIDs for deployed service instances. BPEL developers can use the Service Registry to discover service instances to use in a service composition. We are in the process of developing a decentralised Service Registry based on an unstructured P2P network, described in [22].

The DHT architecture supports the deployment of services on hosts that do not not have a public IP address by providing a globally unique DHT identifier that is used to logically identify hosts in the DBE SOA, including those that reside behind NAT Gateways or have a DHCP allocated IP address. Peers that reside behind a NAT Gateway join the DHT by discovering a Relay Peer[2] that acts as a virtual server for their DHT identifier [24], relaying all messages to and from the NAT-restricted peer. Relay peers must have open static IP addresses.

The ServiceID contains two parts: the DHT identifier that identifies the peer that hosts the service and a local service identifier used by an application server on the peer. The ServiceID can then be used at runtime to discover the host for a service, enabling the Service Proxy object containing the actual binding information to be downloaded.

The Service Proxy Once a consumer has discovered a ServiceID in the Service Registry, it can be used to download a Service Proxy from the service provider's

[2] The discovery of Relay Peers is outside the scope this paper, but systems such as Skype use extensive caching of Relay Peer IP addresses and monitoring of performance [23].

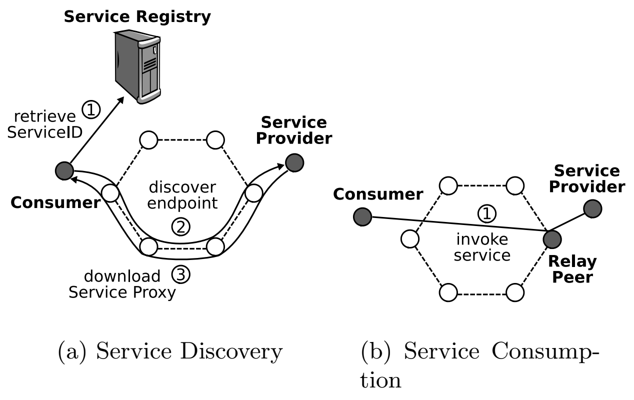

Figure 1. Discovering and downloading a Service Proxy using a Structured P2P Network

application server. The request for the Service Proxy is routed over the structured P2P network using the ServiceID to the peer hosting the service. If the peer hosting the service is NAT-restricted, the request is forwarded to its destination by a Relay Peer, see Figure 1 (a). The Service Proxy is deployed in the provider's local application server using a local, persistent unique identifier generated by the application server[3] and the endpoint information for the service, i.e., either the IP address of the peer itself or the IP address of its Relay Peer if it is a NAT-restricted peer. When the application server receives a request for a ServiceID, it returns a locally deployed Service Proxy for the service if found. A Service Proxy is a Java object that encapsulates the configuration information for services. Within this information the port-specific data, mainly the current endpoint address for a service, is found and can be used to interact with a service, see Figure 1 (b). The introduction of a Relay Peer into an invocation path introduces new failure modes to the transport protocol, meaning that not all invocation protocols are suitable for services deployed on NAT-restricted peers. For example, the use of SOAP RPC could result in lost replies if the Relay Peer unexpectedly fails.

The Service Proxy also specifies the invocation protocol and other binding related information which are supported by the service provider. A service provider can deploy a service with more than one protocol, i.e., more than one Service Proxy, enabling the consumer to choose the binding mechanism most appropriate for their environment. A Service Proxy Framework was implemented to provide a simple API for service consumers to download and handle proxies within the DBE environment.

[3] We use the Sun Servent application server, see http://swallow.sourceforge.net

3.2 Abstract Protocol Adaptor

Middleware often ties the deployed services and the interacting clients to their respective invocation protocols, such as CORBA's GIOP/IIOP or Java's RMI. The emergence of XML-based Web Services standards, such as WSDL and SOAP, help overcome cross-platform limitations. WSDL is not exclusively tied to the SOAP protocol. It enables the specification of multiple bindings, for example for EJBs, JMS, and IIOP.

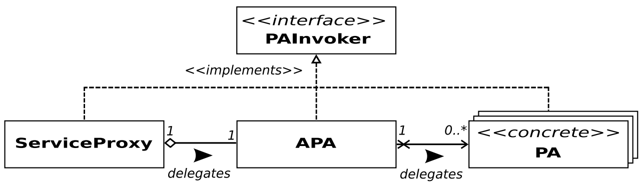

Figure 2. The APA Class Diagram

The Abstract Protocol Adaptor (APA) is a client-side programming model that provides a dynamic invocation interface, similar to that provided by WSIF, to make calls on a generic RPC invocation interface called `PAInvoker` (depicted in Figure 2). The APA framework can be integrated into any component, which requires a protocol-independent communication mechanism, such as a workflow engine. The `APA` promotes loose coupling between clients and the protocol used to access a services, as the protocol is declared or discovered at runtime rather than at compile time. The `APA` is configured at invocation time by a Service Proxy object with the necessary endpoint, the required invocation protocol, and possibly additional configuration. The Service Proxy exposes the same interface as the `APA` , and delegates invocations to the `APA` with all configuration values. The `APA` is ignorant of protocols and simply delegates the invocation to the respective concrete protocol adaptor `PA` at runtime. If a service is implemented to support a single invocation protocol, e.g., SOAP, then the client application will need a compliant SOAP protocol adaptor to create the SOAP calls.

The `APA` can be extended to implement any number of protocol adaptors that can create the necessary setup for a dynamic invocation of a remote service. So far, the following three protocol adaptors have been implemented, SOAP, kSOAP and object serialisation over HTTP. The SOAP `PA` supports the invocation of remote Web Services which expose SOAP endpoints. The kSOAP `PA` supports Web Service communication from mobile clients to kSOAP compliant services.

3.3 Adapting the ActiveBPEL Workflow Engine

The DBE SOA, Service Proxy, and APA frameworks have been integrated into an open source BPEL workflow engine, called ActiveBPEL. The ActiveBPEL organisation [7] has implemented a pluggable workflow architecture in Java that is fully compliant with the BPEL specification, version 1.1 [1]. The engine runs on a servlet container and uses the Axis Web Services container for communicating with other Web Services and clients. It supports SOAP bindings for service invocations. Taking advantage of the pluggable architecture, our work mainly involved integrating this engine with the structured P2P network and APA framework by substituting the default invocation handler with our customised one.

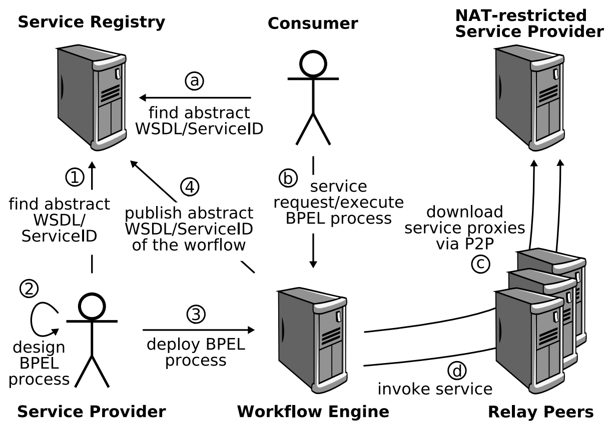
Service Registry
Consumer
NAT-restricted Service Provider
ⓐ find abstract WSDL/ServiceID
① find abstract WSDL/ ServiceID
④ publish abstract WSDL/ServiceID of the worflow
ⓑ service request/execute BPEL process
download service proxies via P2P ⓒ
② design BPEL process
③ deploy BPEL process
ⓓ invoke service
Service Provider
Workflow Engine
Relay Peers

Figure 3. Design, Deployment, and Consumption of a BPEL Process

The deployment process for our adapted workflow engine requires a set of abstract WSDL definitions for all the services aggregated within the BPEL process, see step 1 in Figure 3. These abstract WSDL definitions effectively provide general representations of the technical interface of each service without any information about deployment platforms, bindings to that platform, or the endpoint location of the service. Within the architecture of the DBE a service provider publishes their abstract WSDL definition to a Service Registry with a ServiceID attached. During the design stage of a BPEL process, abstract WSDL definitions are selected to represent a service within the process, see step 2 in Figure 3. The partner relationship between the BPEL process and the selected service interfaces are defined in the `EndpointReferences` (EPRs), specified by WS-Addressing [25], as part of the `PartnerLink` in the Process Deployment Descriptor (PDD) of the ActiveBPEL engine. The ActiveBPEL engine supports the dynamic binding to ports with the `dynamic` and `invoker` endpoint reference strategies. However this requires the BPEL developer to specifically include the assignments and related activities within the BPEL process and possibly implement some additional service to provide the updated endpoint information. In our approach, we avoid using explicit assignments within the BPEL process by declaring the ServiceID of the associated published service as a dynamic URI reference to the service in the `Address` element within the EPR of the

`PartnerLink` . This ServiceID is used to discover Service Proxy objects that contain the binding and port information of a service instance at runtime. The complete BPEL process, containing a reference to the abstract WSDL definitions and their ServiceIDs, can be deployed to our extended workflow engine, see step 3 in Figure 3. The deployment process also publishes the service's abstract WSDL and ServiceID to the Service Registry, see step 4. Consumers can then discover composed services in the Service Registry, see step *a*, and invoke them, potentially via a Relay Peer, see steps *b* to *d*.

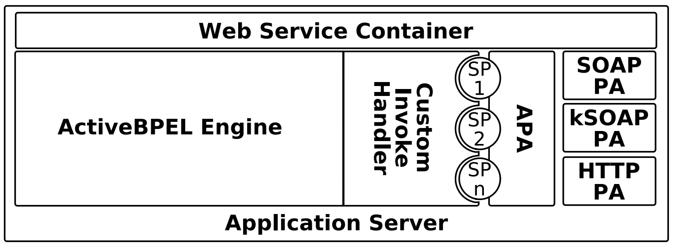

Figure 4. Modified ActiveBPEL Engine

The runtime process for our adapted workflow engine, illustrated in Figure 4, differs only in the behaviour of `Invoke` activities. The `Invoke` activity attempts to invoke an external service based on the abstract WSDL definition and the partner relationship. Our extension to the engine extracts the declared ServiceID for a service from the EPR instead of extracting an endpoint URL to bind to the service with SOAP, see workflow engine in Figure 3. Using this ServiceID, the Service Proxy is downloaded over the structured P2P network. Once a Service Proxy is located and downloaded, it is used to delegate the invocation of an operation to the protocol adaptor framework. The engine is unaware of the binding and port information for this dynamic invocation, as this information is encapsulated by the configuration capacity of the Service Proxy. This enables transparent consumption of services, independent of location, transport, protocol, and even data model.

4 Discussion

Our work has been evaluated by comparison with related work in the area of the execution of BPEL-defined service composition. We examine the performance of our approach for binding- and port-independent composition in relation to other approaches using the following criterion: support for abstract WSDL, interaction paradigm, endpoint discovery and binding mechanisms. The use of the abstract WSDL results in a reduced level of coupling between a service definition and service instance. The interaction paradigm, endpoint discovery, and binding mechanism enable a dynamic invocation approach and hide the underlying middleware paradigm. The results presented in Table 1 are based on our analysis of these approaches (see Section 2). Due to the lack of concrete information regarding the binding mechanisms used in Meteor-S, the assumption was made that the communication protocol is SOAP.

	DBE Engine	**ActiveBPEL**	**Meteor-S**	**SDS**
Abstract WSDL	yes	no	no[a]	no
Interaction Paradigm	Web Services	Web Services	Web Services	Web Services
Endpoint Discovery	Runtime P2P N/W	Design-time[b]	Deployment-time	Runtime
Binding Mechanism	Dynamic Proxy	SOAP	N/A	Dynamic Proxy

[a] Uses Extended WSDL

[b] Also allows for in-model dynamic assignment of endpoints at runtime as stated in the BPEL specification.

Table 1. Comparison of Workflow Engines

All workflow engines compared in this paper use the BPEL specification to define a service composition. Two workflow engines are used in the different approaches, ActiveBPEL and BPWS4J. BPWS4J provides the execution environment for both Meteor-S and SDS, while ActiveBPEL was extended to support dynamic endpoint look-up and binding.

The interaction paradigm for all approaches is based on Web Services standards and requires the services to be deployed in a Web Service container. With our approach a service provider can expose a service using any supported middleware paradigm by the protocol adapter framework and abstract WSDL to describe the interface of this service.

In terms of endpoint discovery, the ActiveBPEL engine offers the least flexibility. The aggregated Web Services of the service composition needs to be discovered at design time including the binding and port information. The Meteor-S project focuses on developer tools that allow binding of Web Services to an abstract process based on constraints and generate an executable process at deployment time [11]. Consequently, the resulting BPEL process is static with respect to the endpoints of the chosen Web Services. While SDS does allow for the runtime discovery of service endpoints, these endpoints are restricted to concrete WSDL port definitions that can only be used by a host with a public IP address. Our approach supports the discovery of endpoints at runtime using a Service Proxy and a service identifier that logically identifies a host in the structured P2P network.

The ActiveBPEL engine and the Meteor-S approach use the standard SOAP binding mechanism for the invocation of services. The SDS is itself a locally bound Web Service that acts as a single dynamic proxy between the workflow engine and the Web Services to be discovered. In our approach, the binding mechanism is not restricted to SOAP. Each service is associated with at least one Service Proxy that encapsulates a binding model for the service.

A feature of our approach to endpoint discovery and service binding is that services deployed on hosts with a non-static IP address can republish their Service Proxies on events such as a change in the host's Relay Peer or DHCP-allocated address. This helps increase robustness of the SOA.

5 Conclusions and Future Work

This paper described a P2P SOA architecture with explicit support for hosts without public, static IP addresses and a workflow engine based on our SOA that supports binding- and port-agnostic service composition. Service interfaces are described using abstract WSDL, and service consumption uses a Service Proxy and Abstract Protocol Adaptor framework that can be extended to any particular interaction paradigm supporting our dynamic invocation interface. Service compositions are defined using BPEL, and service endpoints are looked up at runtime using a service identifier, based on a DHT identifier, to download a Service Proxy over the structured P2P network. This approach supports hosts that reside behind a NAT Gateway by using a Relay Peer that acts as a virtual server for the NAT-restricted peer. We extended the open-source ActiveBPEL workflow engine in order to use the P2P SOA and the dynamic invocation framework, and compared this extension with other composition approaches based on BPEL. Our approach was shown to be more flexible for service providers to support hosts that do not have a public IP address and to be independent from the underlying middleware paradigm.

Issues that remain open for future research include a more flexible and generic execution of service compositions. Instead of downloading Service Proxies, a XML-based configuration file can be used to configure a middleware component that is responsible for the invocation. We are also working on a Relay Peer election algorithm and invocation protocols that better support the different failure modes introduced by Relay Peers in a service invocation path. In order to increase the robustness of our DBE SOA the next version will be based on a decentralised Service Registry available over an unstructured P2P network and consequently avoiding single points of failure. Finally, we are extending the workflow engine to support the intelligent selection of services at runtime for specific domains in the DBE.

Acknowledgements

The work described in this paper was partly supported by the "Information Society Technology" Programme of the Commission of the European Union under research contract IST-507953 (DBE).

References

1. BEA Systems, IBM, Microsoft, SAP AG, Siebel Systems: Business process execution language for web services. Version 1.1 (2003)
2. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Analysis of web services composition languages: The case of BPEL4WS. In: ER. Volume 2813 of LNCS. (2003) 200–215
3. Cardoso, J., Sheth, A.: Semantic e-workflow composition. Journal of Intelligent Information Systems **21**(3) (2003) 191 – 225
4. Orriëns, B., Yang, J., Papazoglou, M.: Model driven service composition. In Orlowska, M.E., Weerawarana, S., Michael P. Papazoglou, e.a., eds.: ICSOC. Volume 2910 of LNCS. (2003) 75 – 90
5. van der Aalst, W.: Don't go with the flow: Web services composition standards exposed. IEEE Intelligent Systems **18**(1) (2003) 72–76
6. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling Churn in a DHT. Proceedings of the 2004 USENIX Technical Conference, Boston, MA, USA (2004) 127–140
7. ActiveBPEL LLC: ActiveBPEL engine, open source BPEL server (2004-2005)
8. Srivastava, B., Koehler, J.: Web service composition - current solutions and open problems. In: Proceedings of Workshop on Planning for Web Services (ICAPS). (2003)
9. Cubera, F., Khalaf, R., Mukhi, N., Tai, S., Weerawarana, S.: The next step in web services. Communications of the ACM **Vol. 46**(10) (2003) 29–34
10. Curbera, F., Duftler, M.J., Khalaf, R., Mukhi, N., Nagy, W.A., Weerawarana, S.: BPWS4J. Published online by IBM at http://www.alphaworks.ibm.com/tech/bpws4j (2002)
11. Aggarwal, R., Verma, K., Miller, J., Milnor, W.: Constraint driven web service composition in meteor-s. In: Services Computing, 2004 IEEE International Conference on (SCC'04), IEEE Computer Society (2004) 23–30
12. LSDIS Lab, University of Georgia: (Meteor-s: Semantic web services and processes)
13. Oldham, N., Thomas, C., Sheth, A.P., Verma, K.: METEOR-S web service annotation framework with machine learning classification. In: SWSWPC. Volume 3387 of LNCS. (2004) 137–146
14. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services. Information Technology and Management **6**(1) (2005) 17–39
15. Rajasekaran, P., Miller, J., Verma, K., Sheth, A.: Enhancing web services description and discovery to facilitate orchestration. In: First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC). (2004)
16. Srinivasan, N., Paolucci, M., Sycara, K.P.: An efficient algorithm for OWL-S based semantic search in UDDI. In: First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC). Volume 3387 of LNCS. (2004) 96–110
17. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic web services. IEEE Intelligent Systems **16**(2) (2001) 46–53
18. Mandell, D.J., McIlraith, S.A.: A bottom-up approach to automating web service discovery, customization and semantic translation. In: International Semantic Web Conference. Volume 2870 of LNCS. (2003) 227–241

19. Agarwal, V., Chafle, G., Dasgupta, K., Karnik, N., Kumar, A., Mittal, S., Srivastava, B.: Synthy: A system for end to end composition of web services. Journal of Web Semantics **3**(4) (2005)
20. Duftler, M.J., Mukhi, N.K., Slominski, A., Weerawarana, S.: Web services invocation framework (WSIF). In: Workshop on Object-Oriented Web Services (OOPSLA). (2001)
21. Mukhi, N., Khalaf, R., Fremantle, P.: Multi-protocol web services for enterprises and the grid. In Hopgood, P.B., Matthews, D.B., Wilson, P.M., eds.: EuroWeb. Workshops in Computing, BCS (2002)
22. Sacha, J., Dowling, J.: A self-organising topology for master-slave replication in peer-to-peer environments. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P) (2005)
23. Baset, S., Schulzrinne, H.: An analysis of the skype peer-to-peer internet telephony protocol. Technical report, CUCS-039-04, Department of Computer Science, Columbia University (2005)
24. Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with cfs. In: SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles, New York, NY, USA, ACM Press (2001) 202–215
25. W3C: Web Services Addressing (WS-Addressing). http://www.w3.org/Submission/ws-addressing/ (2004)