International Journal of Research, Science, Technology & Management
ISSN Online: 2455-2240, Volume 3 Issue II, May 2016

HIGHLIGHT AND THWARK THE ISSUES OF WEB SERVICES OVER TELCOMMUNICATION

Mahi Singh, Sami Khan

Abstract – **The current service-oriented solutions are often not as dynamic and adaptable as intended. The publish-find-bind-execute cycle of the Service-Oriented Architecture triangle is not entirely realized. This paper highlights some issue of current web service technologies, with a special emphasis on service metadata, Quality of Service, service querying, dynamic binding, and service mediation. The Vienna Runtime Environment for Service-Oriented Computing (VRESCo) gives the addresses these issues. The detailed descriptions of the different aspects are focussed on service querying and service mediation. Finally, the paper presents a performance evaluation of the different components, together with an end-to-end evaluation to show the applicability and usefulness of the system.**

Index Terms—Web services publishing and discovery, metadata of services interfaces, advanced services invocation framework.

I.INTRODUCTION

During the last few years, Service-oriented Computing has become an attracting research area to put forward a new paradigm for mastering the complexity of distributed appli- cations by using loose coupling, platform independent inter- face descriptions and well-established standards. Service- oriented architecture (SOA) is a means for capturing these principles by providing an architectural model for developing service-oriented applications. A reasonable technology for implementing SOAs are Web services. In the last few years, Web services evolved from an RPC-centric model to a messaging-based communication model built on SOAP (Simple Object Access Protocol). There is still an ongoing debate within different communities whether to see Web services from an RPC or message-centric point of view. We believe that Web services should implement a messaging-based model to achieve its well-known benefits like loose coupling and a certain degree of version tolerance. RPC-centric Web services do not bene_t from the SOA ad-vantages in terms of architectural exibility and result in Web services being just another distributed object technology.

Besides that, support for dynamic binding and invocation of services is often restricted to services having the same technical interface. In this regard, the lack of service metadata makes it difficult for service consumers to know if two services actually perform the same task. Furthermore, support for Quality of Service (QoS) is necessary to enable service selection based on non-functional QoS attributes such as response time (in addition to functional attributes). In this paper, we discuss the issues we see in current SOC research and practice by describing the problems that arise when building SOC applications with current tools and frameworks. The main contribution is the presentation of the VRESCo service runtime environment that aims at solving some of these issues. To be more specific, the present paper focuses on service metadata, QoS, and service querying, plus dynamic binding, invocation, and mediation of services. Additionally, we provide an extensive performance evaluation of the different components and an end to end evaluation of the overall runtime, which shows the applicability of our approach. The remainder of the paper is organized as follows: Section 2 presents an illustrative example and summarizes some issues of SOC research and practice. Section 3 describes the details of the VRESCo runtime environment, while Section 4 gives a thorough evaluation of our work. Section 5 introduces related approaches and Section 6 finally concludes the paper.

II.MOTIVATION AND PROBLEM STATEMENT

This section first introduces a motivating example, which is used throughout the paper. Then, we derive the problems which is developers face when engineering service-centric systems with current tools and frameworks.

1.Motivating Example

Cell phone operator CPO1 provides different kinds of services: Public Services (e.g., Rate Information Service) can be used by everyone. Customer Services (e.g., SMS Service) are used by customers of CPO1, whereas In-house Services (e.g., CRM Services) represent internal services that should only be accessed by the different departments of CPO1. Besides that, CPO1 consumes services from its partners (e.g., cell phone manufacturers and suppliers) and competitors (e.g., CPO2 and CPO3). As discussed later, this scenario bears several challenges that are typical in service centric software engineering.

2.SOC Challenges

Adaptive service-oriented systems bring along several distinct requirements, leading to a number of challenges that have to be addressed. In this section, we summarize the current challenges that we see most important. The contribution of VRESCo is to address these challenges in a comprehensive service runtime environment.

- *Service Metadata*. Service interface description languages such as WSDL focus on the interface needed to invoke a service. However, from this interface, it is often not clear what a service actually does, and if it performs the same task as another service. Service metadata can give additional information about the purpose of a service and its interface (e.g., pre and postconditions).

IJRSTM
ISSN ONLINE: 2455-2240

- *Service Querying*. Once services and associated metadata are defined, this information should be discovered and queried by the service consumers. This is the focus of service registry standards such as UDDI and ebXML. In practice, the service registry is often missing since there are no public registries and service providers often do not want to maintain their own registry.
- *QoS*. In enterprise scenarios, QoS plays a crucial role. This includes both network-level attributes (e.g., latency and availability) and application-level attributes (e.g., response time and throughput). The QoS model should be extensible to allow service providers to adapt it for their needs.
- *Dynamic Binding and Invocation*. One of the main advantages of service-centric systems has always been the claim that service consumers can dynamically bind and invoke services from a pool of candidate services. However, in practice, this requires identical service interfaces, which is often not the case.

III.SYSTEM DESCRIPTION

This section describes in detail the VRESCo runtime, which was first sketched. Besides an architectural overview, we discuss service metadata and querying, as well as dynamic binding together with our service mediation approach.

1.Overview

The VRESCo core services are provided as web services that can be accessed either directly using SOAP or by using the Client Library that provides a simple API. Furthermore, the DAIOS framework has been integrated into the Client Library, and provides stubless, protocol-independent, and message driven invocation of services. The Access Control Layer guarantees that only authorized clients can access the core services, which is handled using claim-based access control and certificates. Services and associated metadata are stored in the Registry Database, which is accessed using the Object-Relational Mapping (ORM) Layer. Finally, the QoS Monitor is responsible for regularly measuring the current QoS values. The overall system is implemented in C# using the Windows Communication Foundation. Due to the platform-independent architecture, the Client Library can be provided for different platforms (e.g., C# and Java). There are several core services. The Publishing/Metadata Service is used to publish services and metadata into the Registry Database. Furthermore, the Management Service is responsible for managing user information (e.g., name, password, etc.), whereas the Query Engine is used to query the Registry Database. The Notification Engine informs users when certain events of interest occur inside the runtime, while the Composition Engine provides mechanisms to compose services by specifying hard and soft constraints on QoS attributes. In this paper, we focus on the main requirements for our client-side mediation approach, which are the Metadata Service (including the models for metadata, services, and QoS), the Query Engine, and the dynamic binding, invocation, and mediation mechanisms. 3.2 Service Metadata The VRESCo runtime provides a service metadata model capable of storing information about services. This is needed to capture the purpose of services, which enables mediation between services that perform the same task. In this section, we describe service metadata and give examples from the CPO case study.

2. Metadata Model

The main building blocks of this model are concepts, which represent the definition of entities in the domain model. We distinguish among three different types of concepts:

- Features represent concrete actions in the domain that implement the same functionality (e.g., Check_Status and Port_Number). Features are associated with categories, which express the purpose of services (e.g., PhoneNumberPorting).
- Data concepts represent concrete entities in the domain (e.g., customer or phone_number), which are defined using other data concepts and atomic elements such as strings or numbers.
- Predicates represent domain-specific statements that are either true or false. Each predicate can have a number of arguments (e.g., for feature Port_Number, a predicate Portability_Status_Ok(Number) expresses the portability status of a given argument Number).

Furthermore, features can have pre and post conditions expressing logical statements that have to hold before and after the execution of the feature. Both types of conditions are composed of multiple predicates, each having a number of optional arguments. These arguments refer to a concept in the domain model. There are two different types of predicates:

- *Flow predicates* describe the data flow required or produced by a feature. For instance, the feature-Check_Status from our CPO case study could have the flow predicate *requires(Customer)* as precondition and *produces(PortabilityStatus)* as postcondition.
- *State predicates* express global states that are valid before or after invoking a feature. For instance, state predicate *notified(Customer)* can be added as post condition to feature Notify_Customer.

2. Service Model

The VRESCo service model constitutes the basic information of concrete services that are managed by VRESCo. A concrete service (e.g., *Number Porting Service of CPO1*) defines the basic information of a service (e.g., name, description, owner, etc.) and consists of a least one service revision. A service revision (e.g., the most recent version, or a stable one) contains all technical information that is necessary to invoke the service (e.g., a reference to the WSDL file) and represents a collection of operations (e.g., *Check_Status*). Every operation may have a number of input parameters (e.g., Customer), and may return one or more output parameters (e.g., *PortabilityStatus*). Revisions can have parent and child revisions that represent a complete versioning graph of a

IJRSTM
ISSN ONLINE: 2455-2240

concrete service. Both revisions and operations can have a number of QoS attributes (e.g., response time is 1,200 ms) representing all service-level attributes as described below. The distinction in revision and operation-specific QoS is necessary because attributes such as response time depend on the execution duration of an operation, whereas availability is typically given for the revision (if a service is not available, all operations are generally also unavailable). In Section 3.5, we show how concrete services are mapped to the metadata and service model in order to perform service mediation.

3.QOS Model

Besides functional attributes described in the metadata model, nonfunctional attributes are also important. For instance, in our case study, CPO1 may want to always bind to the Notification Service having the lowest response time. Therefore, QoS attributes can be associated with each service revision and operation in VRESCo. These QoS attributes can be either specified manually using the Management Service or measured automatically.

TABLE 1
QoS Attributes

Attribute	Formula	Unit
Price	n/a	per invocation
Reliable Messaging	n/a	{true, false}
Security	n/a	{None, X.509,...}
Latency	$q_{la}(n) = \frac{1}{n}\sum_{i=0}^{n} q_{la_i}$	ms
Response Time	$q_{rt}(n) = \frac{1}{n}\sum_{i=0}^{n} q_{rt_i}$	ms
Availability	$q_{av}(t_0, t_1, t_d) = 1 - \frac{t_d}{t_1 - t_0}$	percent
Accuracy	$q_{ac}(r_f, r_t) = 1 - \frac{r_f}{r_t}$	percent
Throughput	$q_{tp}(t_0, t_1, r) = \frac{r}{t_1 - t_0}$	invocations/s

IV.QUERYING APPROACH

The VRESCo Query Language (VQL) provides a means to query all information stored in the registry (i.e., services and service metadata including QoS). In this section, we discuss the architecture of VQL followed by query specification and query processing.

1.Query Specification

After describing requirements and architecture of the querying framework, we present how queries are specified. In general, VQL queries consist of six elements as follows:

- Return Type R defines the expected data type of the query results. The return type needs to be an element of the VRESCo metadata model (e.g., a list of Feature objects).

TABLE 2
VQL/SQL Translation

Type	VQL	SQL	Description
C_m	Add	WHERE	Mandatory criteria
C_o	Match	IN/JOIN	Optional criteria
E	And	AND	Conjunction of two expressions
	Or	OR	Disjunction of two expressions
	Not	NOT	Negation of an expression
	Eq	=	Equal operator
	Lt	<	Less operator
	Le	<=	Less or equal operator
	Gt	>	Greater operator
	Ge	>=	Greater or equal operator
	Like	LIKE	Similarity operator for strings
	IsNull	IS NULL	Property is null
	IsNotNull	NOT NULL	Property is not null
	In	IN	Property is in a given collection
	Between	BETWEEN	Property is between two values
O	Order	ORDER BY	Ordering of query results
	Asc	ASC	Ascending ordering
	Desc	DESC	Descending ordering

Mandatory Criteria Cm describe constraints that have to be fulfilled by the query (e.g., response time must be less than 500 ms).

- Optional Criteria Co add constraints, which should optimally be fulfilled but are not required (e.g., service provider should be company X).
- Ordering O can be used to specify the ordering of the query results (e.g., sort ascending by ID).
- Querying Strategy S finally defines how the query should be executed (e.g., exact or fuzzy matches).
- Result Limit L can be used to restrict the number of results (e.g., 10 or 0, which represents no limit).

The most important elements are criteria since they actually represent the constraints of the query. Moreover, criteria have different execution semantics depending on the querying strategy, which is discussed in Section 3.3.4. However, the main motivation is to allow the specification of mandatory and optional criteria.

In general, criteria consist of a set of expressions E that is used to define common constraints such as comparison (e.g., smaller, greater, equal, etc.) and logical operators (e.g., AND, OR, NOT, etc.). Table 2 shows criteria (C), expressions (E), and orderings (O), which are currently provided by VQL. Furthermore, the table indicates how each of these elements is translated into SQL, which is described in more detail later. It should be noted that VQL is extensible in that further expressions can be added easily.

As described above, queries are parameterized using the expected return type. In this case, the type ServiceRevision expresses that the result of the query is a list of service revisions. In our example, two Add criteria are used to state that services have to be active and that each service has to implement the Notify_Customer feature (by using the Eq expression). The first parameter of expressions is usually a string representing a path in the user or core model (e.g., Service.Owner.Company describes the company property of the service owner). These strings are central to VQL and are referred to as property paths.

The first criterion expresses that services provided by CompanyX are preferred, while the second criterion defines that revisions should have tags starting with “STABLE” (Like expression). The third criterion specifies an optional QoS constraint on response time, which should be less than 1,000

IJRSTM
ISSN ONLINE: 2455-2240

ms. All three Match criteria use priority values as third parameter to define the importance of a criterion.

Listing 1. VQL sample query

```
// create query object
var query = new VQuery(typeof(ServiceRevision));

// add query criteria
query.Add(Expression.Eq("IsActive", true));
query.Add(Expression.Eq("Service.Category.Features.Name",
                        "NotifyCustomer"));
query.Match(Expression.Eq("Service.Owner.Company",
                          "CompanyX"), 1);
query.Match(Expression.Like("Tags.Property.Name",
    "STABLE", LikeMatchMode.Start), 3);
query.Match(
    Expression.Eq("QoS.Property.Name", "ResponseTime") &
    Expression.Lt("QoS.DoubleValue", 1000.0), 5);

// execute query
var querier = VRESCoClientFactory.CreateQuerier(
    "username", "password");
var results = querier.FindByQuery(query, 10,
    QueryMode.Priority) as IList<ServiceRevision>;
```

2. Query Processing

Query processing is illustrated in Fig. 6. When the query is sent to the VQL Engine, the specified querying strategy is executed, which is implemented using the strategy design pattern [22]. The query is forwarded to the Preprocessor component (step 3), which is responsible for analyzing the VQL query and generating the corresponding SQL query. Next, an NHibernate session is created to execute the generated SQL query on the database (step 4). After execution, the ResultBuilder component takes the results from the NHibernate session context. Since these results represent core objects, they may have to be converted back into the corresponding user objects (i.e., if the return type refers to the user model). This is done dynamically by invoking the constructor of the corresponding object using reflection. For both models, however, the ResultBuilder guarantees type safety of the results, which are finally sent back to the client (step 5).

Algorithm 1 depicts the pseudocode of the Preprocessor. If the query refers to the user model, it is first transformed to the core model. The Preprocessor then iterates over all criteria and expressions. The ResolveAssoc function recursively analyzes the property paths of each expression to determine the necessary table joins. Similarly, the ResolveProp function extracts the property values of each expression. To give an example, reconsider Listing 1:

The property path Service.Owner.Company represents two associations Service and Owner that will be resolved using joins, and one property Company that will be compared with the expression's property value CompanyX. The concrete association/table and property/column names are retrieved using the ORM Layer. The collected information is finally used to build FROM, WHERE, and ORDER clauses of the SQL query, according to the VQL/ SQL translation shown in Table 2.

Algorithm 1. processQueryðR;C; S;OÞ
1: if (isUserObject(R)) then
2: R MapUserToCoreObject(R)
3: end if
4: assocInfo R
5: for all (crit 2 C) do
6: for all (expr 2 GetExpressions(crit)) do
7: assocInfo assocInfo [ResolveAssoc(expr)
8: propInfo params [ResolveProp(expr)
9: end for
10: end for
11: query BuildFrom(assocInfo, propInfo, S)
12: query BuildWhere(query, assocInfo, propInfo, S)
13: query BuildOrder(query, O)
14: return query

3.Querying Strategies

The querying strategy influences how queries are executed. More precisely, it defines the Preprocessor's behavior during SQL generation. The basic transformation process can be summarized as follows: Add criteria are transformed into predicates within the SQL WHERE clause, whereas Match criteria are handled as SQL subselects (IN or JOIN,).

The exact querying strategy forces all criteria to be fulfilled, irrespective of whether this is Add or Match. However, there are scenarios where Match has to be used instead of Add in order to get the desired results (i.e., by enforcing subselects using IN instead of WHERE predicates). In particular, when mapping N:1 and N:M associations (i.e., collection mappings in Hibernate terminology), a query cannot have the same collection more than once in the WHERE predicate. The use of subselects eliminates this effect in VQL; otherwise, such queries would result in null since the associated tables are joined more than once. As an example, reconsider the query in Listing 1 using the exact strategy. When having only one criterion with respect to QoS, Add can be used. However, if there would be a second QoS criterion, Match is required.

The *priority querying strategy* uses priority values for each criterion in order to accomplish a weighted matching of results. Therefore, each Match criterion allows to append a weight to specify its priority, which is internally added if the criterion is fulfilled. The query finally returns the results sorted by the sum of priority values. To give an example, the query in Listing 1 uses the priority values "1," "3," and "5." This means that the constraint on response time is more important than the constraint on revision tags. More precisely, queries that fulfill only the third Match criterion are preferred over queries that fulfill the first and second Match criteria (since 5 > 3 þ 1).

The relaxed querying strategy represents a special variant of priority querying, where each Match criterion has priority 1. Thus, this strategy simply distinguishes between optional and mandatory criteria. Results are then sorted based on the number of fulfilled Match criteria. This allows to define fuzzy queries by relaxing the criteria, which can be useful when no exact match can be found for a query. To achieve the necessary behavior, relaxed and priority querying both translate Match criteria into subselects using JOIN predicates.

TABLE 3
Rebinding Strategies

International Journal of Research, Science, Technology & Management
ISSN Online: 2455-2240, Volume 3 Issue II, May 2016

Strategy	Proxy reconsiders binding...
Fixed	never
Periodic	periodically
OnDemand	on client requests
OnInvocation	prior to service invocations
OnEvent	on event notifications

V.EVALUATION

In this section, we give an evaluation of the VRESCo runtime focusing on the topics covered in this paper. The purpose of this evaluation is twofold: First, we show the runtime performance regarding service querying, rebinding, and mediation by using synthetic data. The main goal of this evaluation is to analyze the performance impact of each aspect in isolation. Second, we combine these aspects into a coherent end-to-end evaluation using an order processing workflow. The main goal is to understand the influence of each aspect with regard to the overall process duration in a realistic setting. Additionally, we show how the individual results of the first part interrelate in an end to end setting. All experiments have been executed on an Intel Xeon Dual CPU X5450 with 3.0 GHz and 32 GB RAM running under Windows Server 2007 SP1. Moreover, we use .NET v3.5 and MySQL Server v5.1. For mediation, rebinding, and end-to-end evaluation, we have created different sets of test services and QoS configurations (with varying response times) using the web service generation tool GENESIS. These test beds are described in detail in the corresponding sections.

1.Querying Performance

First of all, we show the performance of the VQL Engine, which has been measured using the query shown in Listing 1.The test data are generated automatically: In every step, five categories are inserted, each having five alternative services with 10 revisions, while every revision has one tag and 11QoS attributes with random values. It should be noted that in every step, 20 percent of all services match the queried feature Notify_Customer and service owner CompanyX, while only 2 percent of all service revisions match all query criteria. To eliminate outliers, the results represent the median of 10 runs, while the database and Hibernate session cache are cleared after each run.

Table 5 depicts the duration of the individual steps during VQL query processing. Therefore, the previous query is executed on both core and user objects using the exact strategy. Generation (*G*) indicates how long the Preprocessor needs to analyze and generate the query. Execution (*E*) depicts the actual query execution time, while Conversion (*C*) represents the time needed by the *ResultBuilder* to convert the query results.

2. Mediation Performance

Besides rebinding, we have also evaluated the overhead introduced by the VRESCo mediation facilities. We have again used the GENESIS tool for these tests. We have evaluated five different scenarios:

1. no mediation,
2. mediation using only constant mapping functions
3. (replacing an input parameter with a constant string),
4. using mathematical functions (replacing a parameter
5. with a calculated value),
6. using string modification functions (adding a constant
7. string to a string parameter), and, finally,
8. using CS-Script (a simple script which exchanges the
9. order of two parameters).

3.End-to-End Evaluation and Discussion

The end-to-end scenario combines all aforementioned aspects (i.e., querying, rebinding, mediation, and invocation) into a larger order processing case study with the goal of ordering new cell phone contracts online (including mobile phone and SIM card). We implemented this workflow in C#. It consists of 19 overall activities split into four subprocesses. Basically, the process starts upon receiving an order via the company website. Afterward, the internal stock is checked for the availability of the phone and the SIM card. If one of those components is missing, it is ordered by using one of the internal or external suppliers, which is followed by a contracting subprocess. This subprocess creates a new contract, and if necessary, it adds a new customer to the CRM system. If the customer wants to transfer her old number, the number porting subprocess is executed. Finally, the payment and shipping subprocesses are enacted and the cell phone number is activated in the GSM network.

VI.RELATED WORK

In this section, we review related work concerning service repositories and service metadata, as well as service selection, invocation, and mediation.

Currently, several approaches and standards for service registries exist. We have compared some existing solutions with the VRESCo runtime, considering a carefully selected range of established standards, mature open-source frameworks, and commercial tools. We consider the standards UDDI [5] and ebXML [6] (with special emphasis on the registry), Mule ESB and Galaxy repository [26], WSO2 ESB and registry [27], and IBM WebSphere [28] (including ESB, service registry, and repository). Our comparison in Table 6 is structured according to the challenges introduced in Section 2.

Generally, all systems allow to store service metadata. Mostly, this is done in an unstructured way (e.g., using tModels in UDDI). There is only limited support for

TABLE 4
Related Enterprise Registry Approaches

Challenge		UDDI	ebXML	Mule	WSO2	WebSphere	VRESCo
Service Metadata	Unstructured	+	+	+	+	+	~
	Structured	~	~	~	~	+	+
Service Querying	Query Language/API	+	+	+	~	+	+
	Type-safe Query	–	–	–	–	~	+
Quality of Service	Explicit QoS Support	–	–	–	~	~	+
	QoS Monitoring	–	–	–	–	–	+
Dynamic Service Invocation	Binding & Invocation	–	–	+	–	~	+
	Service Mediation	–	–	+	+	+	+
Service Versioning	Metadata Versioning	–	+	+	~	~	–
	End-to-End Support	–	–	–	–	–	+
Event Processing	Basic Notifications	+	+	+	~	+	+
	Complex Event Processing	–	–	–	–	~	+

VII.CONCLUSION

One of the main promises of SOC is the provisioning of loosely coupled applications based on the publish-find-

IJRSTM
ISSN ONLINE: 2455-2240

bindexecute cycle. In practice, however, these promises can often not be kept due to the lack of expressive service metadata and type-safe querying facilities, explicit support for QoS, as well as support for dynamic binding and mediation. In this paper, we have proposed the QoS-aware VRESCo runtime environment, which has been designed with these requirements in mind. VRESCo offers an extensive structured metadata model and VQL as type-safe query language. Furthermore, we provide dynamic binding and mediation mechanisms that use predefined service mappings. We have evaluated our work regarding performance and discussed the results together with the experience gained in the CPO case study.

The results show that the VRESCo runtime is applicable to large-scale adaptive service-centric systems. As part of our ongoing and future work, we want to link the VRESCo eventing and composition mechanisms.

VIII.ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement 215483 (S-Cube). Additionally, the authors would like to thank Lukasz Juszczyk for providing the web service testbed GENESIS, and their master students Andreas Huber and Thomas Laner for their contribution to VRESCo.

REFERENCES

[1] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," Computer, vol. 40, no. 11, pp. 38-45, Nov. 2007.
[2] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D.F. Ferguson, Web Services Platform Architecture: SOAP, WSDL, WSPolicy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice Hall PTR, 2005.
[3] SOAP Version 1.2, World Wide Web Consortium (W3C), http:// www.w3.org/TR/soap, 2003.
[4] Web Services Description Language (WSDL) 1.1, World Wide Web Consortium (W3C), http://www.w3.org/TR/wsdl, 2001.
[5] Universal Description, Discovery and Integration (UDDI), Organization for the Advancement of Structured Information Standards (OASIS), http://oasis-open.org/committees/uddi-spec, 2005. 204 IEEE TRANSACTIONS ON SERVICES COMPUTING, VOL. 3, NO. 3, JULY-SEPTEMBER 2010
[6] ebXML Registry Services and Protocols, Organization for the Advancement of Structured Information Standards (OASIS), http://oasis-open.org/committees/regrep, 2005.
[7] A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, and S. Dustdar, "Towards Recovering the Broken SOA Triangle—A Software Engineering Perspective," Proc. Second Int'l Workshop Service Oriented Software Eng. (IW-SOSWE '07), 2007.
[8] D. Bodoff, M. Ben-Menachem, and P.C. Hung, "Web Metadata Standards: Observations and Prescriptions," IEEE Software, vol. 22, no. 1, pp. 78-85, Jan./Feb. 2005.
[9] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," ACM Trans. Web, vol. 1, no. 6, p. 6, 2007.
[10] L. Zeng, B. Benatallah, A.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," IEEE Trans. Software Eng., vol. 30, no. 5, pp. 311-327, May 2004.
[11] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, "End-to-End Versioning Support for Web Services," Proc. Int'l Conf. Services Computing (SCC '08), 2008.
[12] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Advanced Event Processing and Notifications in Service Runtime Environments," Proc. Second Int'l Conf. Distributed Event-Based Systems (DEBS '08), 2008.
[13] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Service Provenance in QoS-Aware Web Service Runtimes," Proc. Seventh Int'l Conf. Web Services (ICWS '09), 2009.
[14] P. Leitner, F. Rosenberg, and S. Dustdar, "Daios—Efficient Dynamic Web Service Invocation," IEEE Internet Computing,vol. 13, no. 3, pp. 72-80, May/June 2009.
[15] J. Lo¨wy, Programming WCF Services. O'Reilly, 2007.
[16] F. Rosenberg, P. Celikovic, A. Michlmayr, P. Leitner, and S. Dustdar, "An End-to-End Approach for QoS-Aware Service Composition," Proc. 13th Int'l Enterprise Computing Conf. EDOC '09), 2009.
[17] F. Rosenberg, P. Leitner, A. Michlmayr, and S. Dustdar, "Integrated Metadata Support for Web Service Runtimes," Proc. Middleware for Web Services Workshop (MWS '08), 2008.
[18] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping Performance and Dependability Attributes of Web Services," Proc. IEEE Int'l Conf. Web Services (ICWS '06), 2006.
[19] M. Fowler, Patterns of Enterprise Application Architecture. ddison- Wesley, 2002.
[20] Hibernate Reference Documentation v3.3.1, Red Hat, Inc., http:// www.hibernate.org, 2008.