

WebComposition/DGS: Dynamic Service Components for Web 2.0 Development

Martin Gaedke¹, Danilo Härtzer¹ and Andreas Heil^{1,2}

¹Chemnitz University of Technology
Faculty of Computer Science
09111 Chemnitz, Germany

{firstname.lastname}@cs.tu-chemnitz.de

²Microsoft Research
Computational Sciences Group
CB3 0FB Cambridge, United Kingdom
v-ahheil@microsoft.com

Abstract

Modern software solutions for distributed applications show a strong need for fast and flexible integration of heterogeneous service technologies, as well as for the orchestration of traditional service-based business scenarios (e.g. SOAP or RPC) and REST-driven approaches. The fulfilment of these requirements in existing enterprise platforms is typically accomplished with parallel working components based on different technologies and APIs, resulting in architectural overhead and increased complexity. The WebComposition/DGS (Data Grid Service) aims at solving this issue with the help of a central service layer which specifies a common plug-in framework used to adapt concrete service protocols and which offers an easy-to-use interface for deploying new logic and services. This approach reduces development effort and overall costs thanks to its coherent and integrated DGS-compatible service architecture with automated support for several protocols.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems – *Distributed applications*; H.3.4 [INFORMATION STORAGE AND RETRIEVAL]: Systems and Software – *Information networks*; H.3.5 [INFORMATION STORAGE AND RETRIEVAL] Online Information Services – *Web-based services*

General Terms

Management, Economics, Reliability, Standardization.

Keywords

WebComposition, Web 2.0, Web Engineering, DGS, REST, RESTful, Service-oriented Architecture.

1. INTRODUCTION

This work will introduce a new main component for the WebComposition/DGS platform [1], which is offering the ability to create, deploy, host and manage services in a generic

way. The WebComposition/DGS platform is the first element of the fourth generation of the WebComposition approach. It offers a XML-based access to distributed data and its semantic relationships based on RDF (Resource Description Framework) support. The DGS solution focuses on an easy manageability and a straightforward processing of XML-structures within the context of the complex distributed scenarios characteristic of modern Web applications, based on REpresentational State Transfer (REST) principles [2] and common SOA-based business scenarios. Inside the DGS' architecture, the central service component plays an essential role in taking over logic processing and orchestration. The DGS core component interacts with the integrated rights management API and supports user and role dependent policy rules for single-sign-on scenarios and environments with federative access rules.

This clear separation of concerns, based on a well-structured composition of data, logic and policy handling components provides a solid basis for the design of distributed applications. This paper will give the reader an insight into the processing logic of the central service component of the DGS as well as into its interfaces and specifications, which aims at supporting a rich, but easy-to-use infrastructure to deploy, manage and host dynamic service units.

2. STATE-OF-THE-ART

The DGS architecture approach introduced in this paper showcases a platform for the implementation of distributed applications. This section will provide a concise overview and comparison of alternative software solutions within the same problem spectrum.

2.1 Application Server

Application servers provide a runtime environment for business applications and interfaces within the context of enterprise solutions [3] (e.g. transaction management, O/R-Mapping approaches, authentication etc.).

An application server acts primarily as a technological platform on top of which user applications can be implemented and hosted. In order to write and deploy new services, the corresponding business logic and its components need to be implemented, typically within a development environment (IDE). Thus, supporting different service technologies requires tying the concrete service implementations to specific third-party libraries (e.g. implementing SOAP services by making use of the Apache Axis framework in a JEE scenario). As a matter of fact, there is currently no API or framework providing an automated mapping of service logic to a wide set of service

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ii WAS2008, November 24–26, 2008, Linz, Austria.
(c) 2008 ACM 978-1-60558-349-5/08/0011 \$5.00.

protocols and transports. Furthermore, the use of an application server to support a number of different service technologies (e.g. SOA-like or REST-driven approaches [4]) results in increased complexity for development and runtime environments. As a case in point, the implementation of a service call in a common SOA-scenario requires following technical artefacts: WDSL/XML technologies (SOAP, BPEL), service frameworks (e.g. Apache Axis), a specific application server API (.NET or JEE), an O/R-Mapping framework (e.g. Hibernate or Toplink) and a data layer, typically a relational or an object-relational hybrid database like Oracle, PostgreSQL or Microsoft SQL Server.

2.2 Orchestration of Web Services

Several platforms offer the possibility to orchestrate and manipulate web services, either based on recent specifications like BPEL or BPM, or with the help of non-standard Web 2.0 solutions (Yahoo! Pipes, RSS-Syndication). These tools allow for the implementation of logic by combining, mixing and linking existing web services.

Contrary to the DGS platform, this approach has evident drawbacks in the realization of applications based on a specific data model and contemplating a non-trivial business logic layer. For instance, the BPEL approach is indeed suitable for the orchestration of existing services, but does not currently support an integrated persistence layer, thus forcing the developer to implement by hand the systemic parts of the service components [5].

2.3 Summary

The WebComposition/DGS solution offers an approach combining the functional areas of application servers and web service orchestration. Distributed applications can be created, deployed and managed online without administrative effort thanks to the integrated components for service processing, rights management and XML handling.

The easiness with which new services for the WebComposition/DGS platform can be created and deployed represents an advantage over the traditional and bloated process based on the J2EE stack and on known application servers such as JBoss, IBM Websphere and BEA Weblogic. Furthermore, in comparison to common web service orchestration engines like BPEL engines (e.g. Apache ODE or jBPM), the WebComposition/DGS platform offers a more complete and mature support for data creation and manipulation.

3. INTEGRATED SERVICE COMPONENTS

The core functionality of the WebComposition/DGS platform lies in managing XML-based data for distributed applications.

The service layer acts as central hosting component for business logic, offering abilities to deploy dynamic service units as well as managing, maintain and introspect DGS core services (e.g. the rest driven CRUD services).

To realize described functionalities following requirements must be supported:

- Optimal processing of DGS data models
- Simple support of distributed applications and services
- Support of an extensible service architecture
- Service management capabilities
- Integration of the DGS rights management

3.1 Optimal Processing of DGS Data Model

The specification of deployable logic for the realization of services is compatible with the DGS data model and guarantees a powerful processing approach.

The data model of the DGS is based on the XML and XSD technologies which will be shortly discussed in the following paragraphs.

3.1.1 XPath

XPath is used to locate and extract specific sections of an XML document. As a matter of fact, XPath instructions are only able to select and extract one or more sections of an XML document, but cannot apply any modification to it. Due to this restriction, the XPath approach is not powerful enough in order to act as a processing language, but represents a viable choice for simple query and selection operations. Furthermore, XPath constitutes an integral subset of more advanced XML processing technologies like XSLT or XQuery [5].

3.1.2 XSL Transformation (XSLT)

XSLT is a programming language to transform XML documents. This approach offers a powerful and flexible solution for processing common logic in the context of a DGS data model. The disadvantage of this approach is given by the serious performance penalty associated with a wide use of XSLT instructions. The processing of an XSLT style sheet requires traversing the whole source document, even for the transformation of few XML elements, hence rendering this approach not suitable for the general manipulation of large document sets.

3.1.3 XQuery

XQuery is a query language for XML data standardized by the W3C consortium, which supports complex constructs used to manipulate, filter, link, transform and create XML documents. The usage of XQuery in the context of XML processing is expected to play in the coming years a similar role as the SQL query language in the traditional context of relational and hybrid relational-object databases [6]. Another advantage of this technology can be found in its independency from a specific database technology. Finally, it should be noted that XQuery can be successfully used to query and process XML stored in databases as well as in traditional text documents.

3.1.4 Service Abstraction Layer

To avoid a strict coupling in the specification of application logic, the WebComposition/DGS approach supports a service abstraction layer as a common way to process and manage deployable service units. The central service framework separates the processing logic from the protocol technology, and follows an extensible approach. A set of available operations, described within the DGS service specification, is defined for

each document type, i.e. an XML data structure which is deployed to the DGS. These XML data structures maintained within a DGS are called *lists* in the following.



Figure 1. Root element of the DGS service specification.

Each list dependent service, representing a document type, specifies an amount of operations which will be available and deployed within the DGS.

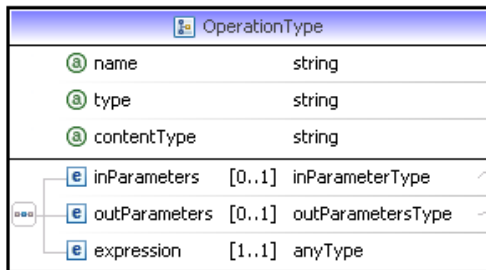


Figure 2. Specification for a service operation.

Each service operation consists of the following elements (cf. Figure 2):

- A unique name
- An optional content type for the result
- An expression type used to determine the processing engine
- A description for in and out parameters, if any
- An expression with type specific processing instructions

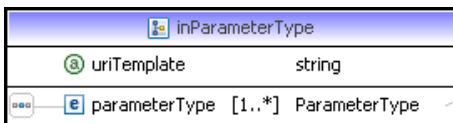


Figure 3. Specification for *inParameterType*.

The *inParameterType* owns a property, called *uriTemplate* (cf. Figure 3), which holds the information needed to deploy the operation in a REST-driven scenario.

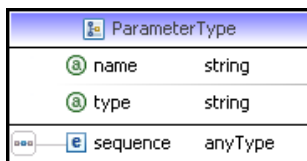


Figure 4. Specification for *parameterType*.

Figure 4 shows the specification of *ParameterType* used to describe in and out parameters for service calls.



Figure 5. Specification for *outParameterType*.

Finally, Figure 5 shows signature of the *outParameterType*, used to describe result schemes for service calls. With this specification, a protocol independent service can be described and deployed to the DGS.

Let us consider a sample operation:

```
<operation name="getOpenTasksOf"
  type="text/xpath"
  contentType="text/xml">
  <inParameters uriTemplate="{userid}">
    <parameter name="userid"
      type="xsd:string"/>
  </inParameters>
  <outParameters>
    <parameter name="result"
      type="xsd:complexType">
      <xsd:sequence>
        <xsd:element name="task"
          maxOccurs="unbounded"
          type="TaskType"/>
      </xsd:sequence>
    </parameter>
  </outParameters>
  <expression>
    task[users/user/@id={userid}][@state='open']
  </expression>
</operation>
```

The example above describes a service for an operation of a hypothetical task management service used to fetch all open tasks of a given user. This operation will be added to the task service after deploying it to the DGS.

3.2 Support Facilities to Access Services

Each dynamic service, once deployed, can be accessed over the HTTP protocol without the need of an external service plug-in:

```
GET http://{DGS-baseUri}/{listName}/
meta/service/{operationName}/{uriTemplate}]
```

The above GET method represents the dynamic endpoint for the REST-based approach in the DGS. Required arguments are specified as part of the *uriTemplate*.

```
POST http://{DGS-BaseUri}/{listName}/
meta/service/{operationName}
```

The POST method call above shows an alternate call syntax for a DGS service. In this case, the required parameters are contained in the body of the HTTP call.

3.3 Extensible Service Architecture

Based on the functionality briefly presented in the previous sections, the DGS offers the possibility to plug in additional adapters in order to support a number of different service technologies and protocols. Each adapter is responsible for the

mapping of service calls to specific endpoints and protocols. The main task of a service adapter consists in listening at its endpoint and serving each incoming request by delegating its logic processing to the central service layer.

Another important task lies in the dynamic expansion of WSDL service descriptions with accessory information on supported endpoints and protocols. As a matter of fact, for each document type handled by the DGS, a single centrally managed service description exists, which includes the definition of all available operations and of all supporting endpoints and protocols.

Dynamically created WSDL documents can be accessed at:

```
GET http://{DGS-BaseUri}/{listName}/meta/wsdl
```

Additionally, the standard distribution of DGS offers a SOAP support adapter reachable at the following endpoint:

```
POST http://{DGS-BaseUri}/{listName}/meta/soap
```

3.4 Management Capabilities for Services

The management interface is based on a REST-driven CRUD (create, read, update and delete) approach similar to the one already available in the DGS.

```
GET http://{DGS-BaseUri}/{listName}/meta/service
```

The above request returns the service description for the specified document type (given as *listName*) with all its available and deployed operations (the requested service description will be served like in section 3.1.4 described scheme). Add, delete and manipulate operations are supported by a CRUD behaviour similar to the one offered in DGS for list manipulation.

```
GET http://{DGS-BaseUri}/{listName}/  
meta/service/{operationName}
```

The above request returns the XML entry for the given operation.

```
POST http://{DGS-BaseUri}/{listName}/  
meta/service/{operationName}
```

The above POST method adds a new operation.

```
DELETE http://{DGS-BaseUri}/{listName}/  
meta/service/{operationName}
```

The DELETE method removes the specified operation.

3.5 Rights Management

Access rules for all deployed operations and services can be defined based on the central rights management component.

The DGS has an integrated WS-Federation Standard compatible authentication support, for instance providing identity through idFS (Identity Federation System), as a further component of the WebComposition approach [7].

Furthermore, the DGS offers user/role based rights management, so for each DGS managed operation, identified by its name and its list dependency) a set of in- and excluded roles can be defined.

4. Conclusion and Further Work

In this paper a simple but powerful approach for the implementation of distributed services on top of the WebComposition/DGS platform has been introduced. Under consideration of reusability and security requirements, the service specification described in the previous sections acts as a central instance for logic processing.

Furthermore, in this work a flexible design allowing the easy integration of different service technologies in a powerful architecture for SOA-based distributed solutions has been outlined and discussed. The introduced service component offers a simple, but powerful way to add, manage and remove new service units on-demand at runtime. In relation to these capabilities the DGS can be used as an application platform as part of distributed solutions. The service deployment and management functions presented above are structurally similar to the data manipulation behavior supported in the DGS platform and inspired to the guidelines of the W3C consortium.

The service layer aims at supporting specific core functionalities like REST-driven CRUD operations and metadata management. In order to fulfill this goal, the specifications must be extended and improved in order to cover all the remaining REST constraints. Furthermore, the DGS service specification needs to follow the WSDL 2.0 specification [8] and offer a complete support for the description of all available RPC- and REST-driven services.

5. REFERENCES

- [1] Heil, A. and Gaedke, M., "WebComposition/DGS: Supporting Web2.0 Developments With Data Grids." In IEEE International Conference on Web Services (ICWS 2008), Beijing, China, 2008.
- [2] Fielding, R., Architectural Styles and the Design of Network-based Software Architectures PhD thesis. Irvine: University of California, 2000.
- [3] Natis, Y. V., Pezzini, M., Iijima, K., and Favata, R., "Magic Quadrant for Enterprise Application Servers, 2Q08," Gartner, Inc. 24 April 2008.
- [4] Richardson, L. and Ruby, S., RESTful Web Services: O'Reilly, 2007.
- [5] Margolis, B. and Sharpe, J. L., SOA for the Business Developer: Concepts, BPEL, and SCA: Concepts, BPEL and SCA: MC Press, Lewisville, 2007.
- [6] Lehner, W. and Schöning, H., XQuery : Grundlagen und fortgeschrittene Methoden: dpunkt Verlag, Heidelberg, 2004.
- [7] Meinecke, J., Nussbaumer, M., and Gaedke, M., "Building Blocks for Identity Federations." In Fifth International Conference on Web Engineering (ICWE 2005), Sydney, Australia, 2005, pp. 203-208.
- [8] W3C, Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language: <http://www.w3.org/TR/wsdl20> (08-18-2008).