Development of Web 2.0 Applications using WebComposition/Data Grid Service

Olexiy Chudnovskyy, Martin Gaedke
Faculty of Computer Science
Chemnitz University of Technology
Chemnitz, Germany
olexiy.chudnovskyy@s2004.tu-chemnitz.de
martin.gaedke@informatik.tu-chemnitz.de

***Abstract*—Data integration and content publishing in terms of Linked Data is a complex and time-consuming task while developing Web 2.0 applications. Considering this problem separately from architecture design increases application maintenance effort and causes additional overhead to provide public access functions. In this paper, we present the WebComposition/Data Grid Service and its data management capabilities to meet demands of modern Web 2.0 applications. We show how to facilitate the application implementation and shorten development time by applying the Data Grid Service as Web Service-based storage solution.**

***Keywords*-REST; Linked Data; Web 2.0.**

I. Introduction

The classical approach while developing Web 2.0 applications foresees many steps beginning with problem analysis over data modeling, architecture design and ending with implementation and maintenance [1]. Consider the development process of a small Web 2.0 application. As an example, we create a small online-tool to support Scrum software development method [2]. In Scrum a product owner separates the project into stories, which are functionalities a client wishes from the application. Stories are implemented by the Scrum team during sprints - fixed periods of time, usually 2 or 4 weeks. The team divides the stories into small tasks and solves them by implementing the specified functionality. If problems occur, so called impediment requests are posted to the scrum master, who tries to solve them and cares about the smooth development process.

First we define entities and relationships using the UML class diagram from Fig. 1. Following the classical approach we use one of the Object-Relational-Mapping libraries (like Hibernate [3] or Microsoft Entity Framework [4]) to map the described classes and associations onto tables of a relational database. This way the application deal only with conceptual scheme and concentrates on business logic, abstracting from database read/write operations and communication details. With the help of a Model-View-Controller (MVC) Framework we develop the presentation level and implement navigation functions. Due to the simplicity of our example application it doesn't take much time to implement the business logic. The creation, edition and retrieval functions are usually automatically generated by modern MVC frameworks.

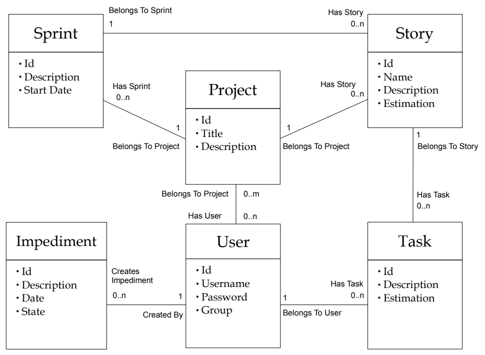


Figure 1. UML class diagram of scrum tool.

The created application works fine as a standalone tool, but still doesn't collaborate with other services or exchange any data. Assuming we would like to provide a Really Simple Syndication (RSS) feed [5] with newly added impediment requests, additional programming effort in implementation of a web-service is needed making this simple feature costs- and time-consuming. Publishing the information about ongoing projects in Resource Description Framework (RDF) format [6] requires new work again, transforming internal data into new representation and exposing it by implementing new web service methods. Further functionalities like public or new data representations become even more expensive due to the implementation and maintenance costs.

As we see, application development time and project costs could be decreased if collaboration with other services as well as publishing of content in terms of Linked Data would be considered in the planning phase. Addressing these problems after essential application functions are implemented, data is strictly modeled and manipulation methods are defined makes the further development inefficient and increases maintenance costs. Our solution acts as a web-based storage solution targeting common integration and data exchange needs of modern Web 2.0 applications and supporting the developer in implementation and maintenance of web-based applications. We show how schema-free data can be modeled using Data Grid Service (DGS) and manipulated in the RESTful way [7] (Section II). We illustrate the application of Data Grid Service as underlying storage engine (Section III) and discuss it respecting complexity and performance aspects (Section IV). We also present some related approaches in Section V.

II. WebComposition/Data Grid Service

In this section, we discuss the fundamentals of the WebComposition/Data Grid Service, present data modeling possibilities, access methods and internal architecture of the service.

A. Basic Principles

The Data Grid Service acts as a flexible and easy to integrate component providing wide information exchange and sharing possibilities. Focusing on the management of XML lists and corresponding metadata in a RESTful way, the service can be applied in a variety of scenarios with different requirements on discovery, presentation and integration of data. The concept of URI plays a decisive role in data access and manipulation methods, the variety of supported representation formats makes it easy to share the information and integrate it into existing applications. Though the service focuses on the maintenance of data in form of XML lists, further functionality such as content transformation using XSLT stylesheets, binary content or gateways to other data sources may be managed through extensions.

The logical view on the resources managed by Data Grid Service is described by a set of so called information stores (Figure 2). The information stores provide access to the resources inside and corresponding metadata. For example an information store may act as a single XML list, containing XML representation of people or publications and providing Create/Read/Update/Delete (CRUD) methods for item manipulation. The information stores, metadata and single items are references through URIs, service architecture allows items to contain further information stores or act as a gateway to other services or data sources. To create information stores within the Data Grid Service a corresponding HTTP request is made with descriptive information about the newly created information store. Configuring the stores using metadata allows not only the definition of functionality but also affects performance issues, e.g., XML lists may be internally stored either directly in separate files or in the database to maintain larger amounts of data. Moreover relationships between information stores can be configured to merge the contents and process the combined data.

B. Data Model

The WebComposition/Data Grid Service manages structured data in form of XML lists. Additionally service can handle lists of binary arrays, providing a fast and flexible storage solution for web application resources. Resource

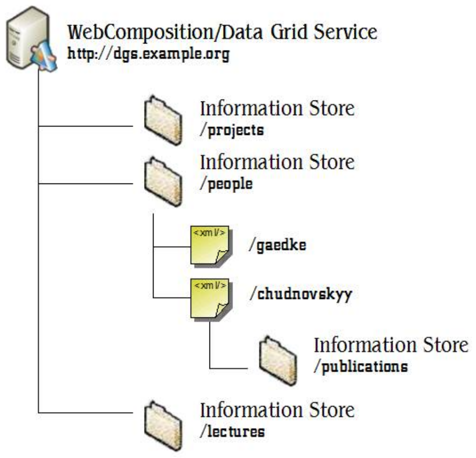


Figure 2. WebComposition/Data Grid Service. Logical view.

metadata in form of RDF statements can be created to annotate the stored information, connect it with related resources or configure access and manipulation methods. XSLT stylesheets are used to transform the content into another representation formats, such as RDF or JSON, or to organize the data as Atom or RSS Feeds.

Resources in DGS may be created via HTTP in the RESTful way and configured for the future use by provided descriptive metadata. Depending on the type of the created resource its behavior and access methods are defined. In case of XML lists, single XML blocks may be added, retrieved or deleted. To validate the incoming data a XSD schema can be defined, specifying either the overall list structure or making restrictions on the incoming elements. The approach simplifies the maintenance of objects used by web applications and allows making changes into data structure without reorganizing dependencies or affecting other contents. In order to support the development of applications with focus on relationships between objects foreign keys and connections may be specified between lists providing a flexible access to subordinate items. The relationships are defined using RDF statements so that external services may consume this information to optimize their own discovery and integration functions.

C. Data Manipulation

The WebComposition/Data Grid Service is a web component designed in a RESTful architectural style, providing a number of resource discovery and maintenance functions. With a HTTP GET request on service metadata (/meta) the RDF description of existing information stores and current service configuration is retrieved

New information stores are added with a POST request on Data Grid Service URL providing description of the resource to be created. The type of the resource (list of XML elements, XSLT transformation, gateway to other services etc.) defines the allowed operations both on the resource itself but also on the subordinate items. Single list items are created with a POST request on the corresponding parent list URL:

```
POST /authors HTTP/1.1
 Host: dgs.example.org
 Content-Length: 89
 Content-Type: text/xml

<author>
  <fname>Olexiy</fname>
  <sname>Chudnovskyy</sname>
  <city>Chemnitz</city>
</author>

HTTP/1.x 201 Created
 Location: http://dgs.example.org/authors/5
```

Unique id's are assigned to the newly created store items so these can be later retrieved, updated or deleted with corresponding GET, PUT or DELETE methods. To make the item URI even more descriptive a URI Template [8] may be defined to map the incoming request onto predefined XPath expression selecting the appropriate items from the list:

```
POST /authors/meta HTTP/1.1
 Host: dgs.example.org
 Content-Length: 89
 Content-Type: text/n3

 @prefix meta: <http://www.webcomposition.net
      /2008/02/dgs/meta/>.
<http://dgs.example.org/authors>
 meta:urlTemplate
 [
  meta:url "authors/{value}";
  meta:xPath "/authors/student[sname='{value}']"
 ].
```

The newly created item would be then alternatively available under the URI: *http://dgs.example.org/authors/chudnovskyy*.

The metadata of XML list items may provide further information in RDF format about the resource e.g., its creation date or list creator.

The described approach simplifies the fast development of many Web 2.0 applications, e.g., blogs, online presentations or information sharing portals by providing a flexible and intelligent storage solution. Satisfying the needs of developers to model structured data, Data Grid Service exposes its content in a RESTful way, so the content may be immediately consumed by other applications and services.

To support applications based on the domains with many connections between items, the pre-configured relationships are used by the Data Grid Service to aggregate subordinate items. The relationship is defined through 4 obligatory and 3 optional attributes:

- *Source*: A URI of the information store within the Data Grid Service to act as a primary list, e.g., *http://dgs.example.org/authors/*
- *Target*: A URI of the information store within the Data Grid Service to act as a subordinate list, e.g., *http://dgs.example.org/publications*
- *Predicate*: A URI of RDF predicate to act as a foreign key, defining a connection between primary and secondary list items, e.g., *http://www.webcomposition.net/2008/02/dgs/meta/has-Published*. Predicates are automatically stored in the metadata of the parent item.
- *URI*: The unique identifier for the relationship. In particular an URL within Data Grid Service domain is used to retrieve the relationship details, to modify or to delete it. The URL is provided by the service and is sent in the Location header to the client after creation.

A relationship defined through the obligatory attributes allows the service to process URIs after the following pattern:

```
http://{service_host}/{source_list_name}/
{source_item_id}/{target_list_name}
```

and as such, filtering only those items from target list that have a relationship to the parent list item *source_item_id* over the RDF property defined in *Predicate*-attribute. A POST request on the same URI is used to add new items to the subordinate list connecting it simultaneously with the given parent list item. An inverse operation to remove the relationship between items is performed using a DELETE request on the URI

```
http://{service_host}/{source_list_name}/
{source_item_id}/{target_list_name}/
{target_item_id}
```

An optional *Inverse-Predicate*-attribute can be specified to define a reverse relationship from the target list to the source list. A corresponding RDF statement is then automatically assigned to the child item metadata, acting as a foreign key to the parent list item. The approach improves both performance processing *n:m* relationships and lets the Data Grid Service process the URIs after the reverse pattern:

```
http://{service_host}/{target_list_name}/
{target_item_id}/{source_list_name}
```

In example above both publication(s) of some fixed author and also author(s) of some fixed publication can be retrieved with simple GET requests on the corresponding URIs. If many relationships between the same source and target list should be modeled, optional *Source* and *Target Aliases* are specified to resolve conflicts with already existing relationship definitions. The predicate of the relationship is then used to perform aggregation of target list items and response to the requests URIs like

```
http://{service_host}/{source_list_name}/
{source_item_id}/{target_list_alias}
```

or

```
http://{service_host}/{target_list_name}/
{target_item_id}/{source_list_alias}
```

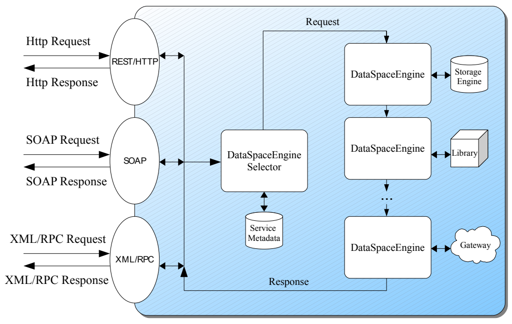


Figure 3. Data Grid Service architecture

The data modeling possibilities and processing functions let the developer concentrate on the contents and user interface of the web application supporting them by flexible and intelligent storage solution. The ease of service integration and data retrieval shortens both development of prototypes and real applications. As the content is immediately available in form of XML for consumption and integration into existing applications, implementation of additional web services or API to access the application data is unnecessary. The variety of representations formats and target clients is supported through configurable transformation of output content. E.g., author may provide an RSS Feed of his newly posted publications simply providing the Data Grid Service a XSLT stylesheet, that should be applied on the XML document returned to the request on *http://dgs.example.org/authors/chudnovskyy/publications*. The same way one creates RDF graphs or JSON representation from the data stored in Data Grid Service.

D. Data Grid Service Internals

The flexibility of the service is achieved by integrating new components, so called Data Space Engines, handling the incoming requests with predefined URI patterns (Figure 3). The request is first analyzed by the *DataSpaceEngine*-Selector component to determine the information store type of the requested URI.

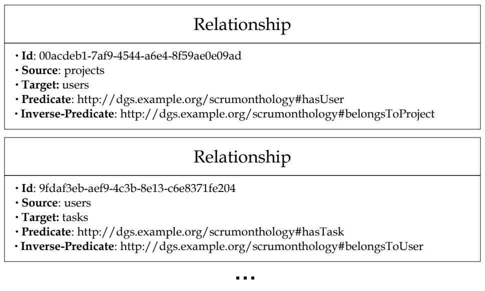
The processing of the request is done afterwards by a chain of Data Space Engines, providing the specific behavior of the information store. The Data Space Engines may complete different tasks, such as authorization, resource versioning, data manipulation or gateway functionality. 3rd party libraries, components, storage engines and services may be used to accomplish the task. Following Data Space Engines are currently implemented:

- *XmlDataSpaceEngine* - main component providing the basic functionality on XML lists. Both lists and XML items are created using corresponding HTTP requests. Metadata is maintained for stored resources, containing RDF statements describing the contents and relationships to other lists or items. URI Templates and relationship definitions are resolved and processed by the component.
- *XSLTDataSpaceEngine* - transforms the requested XML resource according to the defined XSLT transformation. XML lists or single items can be used to create an alternative representation of contents, e.g., Atom or RSS feeds, RDF graphs, JSON representation or HTML pages. The behavior is configured through the list metadata.
- *BinaryDataSpaceEngine* - manages lists of binary content, automatically extracting meta data from known formats and storing it using common RDF vocabularies.

The well defined interface allows developers to extend the functionality of the service implementing further functionality, e.g., synchronizing the stored contents with other data sources or restricting access to specific resources.

III. WebComposition/Data Grid Service in Use

In this chapter we apply the Data Grid Service as a storage solution to the example application discussed in Section I. To represent the entities, we start with defining 6 XML lists representing the entities (classes) of the UML model, i.e., their URIs */projects*, */stories*, */sprints*, */users*, */tasks* and */impediments* and corresponding schemas to perform data validation. We also describe the associations by submitting the corresponding information to the Data Grid Service (Figure 4).


Relationship

- **Id**: 00acdeb1-7af9-4544-a6e4-8f59ae0e09ad
- **Source**: projects
- **Target:** users
- **Predicate**: http://dgs.example.org/scrumonthology#hasUser
- **Inverse-Predicate**: http://dgs.example.org/scrumonthology#belongsToProject

Relationship

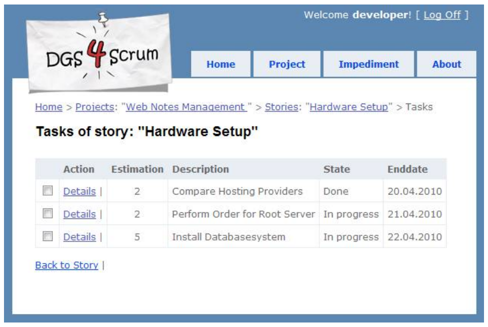
- **Id**: 9fdaf3eb-aef9-4c3b-8e13-c6e8371fe204
- **Source**: users
- **Target:** tasks
- **Predicate**: http://dgs.example.org/scrumonthology#hasTask
- **Inverse-Predicate**: http://dgs.example.org/scrumonthology#belongsToUser

...

Figure 4. Definitions of entity relationships.

Each relationship is specified in the service's metadata through a set of RDF statements, identifying source and target XML lists as well as connection predicates. The associations between entity objects (depending on the direction) are now represented by URIs as follows:

- */projects/*{*project id*}*/users* & */users/*{*user id*}*/projects*
- */projects/*{*project id*}*/stories* & */stories/*{*story id*}*/projects*
- */users/*{*user id*}*/tasks* & */tasks/*{*task id*}*/users*
- ...



Welcome **developer**! [Log Off]

Home > Projects: "Web Notes Management " > Stories: "Hardware Setup" > Tasks

Tasks of story: "Hardware Setup"

	Action	Estimation	Description	State	Enddate
☐	Details |	2	Compare Hosting Providers	Done	20.04.2010
☐	Details |	2	Perform Order for Root Server	In progress	21.04.2010
☐	Details |	5	Install Databasesystem	In progress	22.04.2010

Back to Story |

Figure 5. Example application based on Data Grid Service.

The definitions made above are enough to maintain the application data, to create new entities and to connect them. The content manipulation is performed using GET, POST, PUT and DELETE methods. As in the approach from Section 1 we implement the UI layer and authorization logic using a MVC Framework (ASP.NET MVC [9]). In contrast, implicit support for data exchange and integration is given through the RESTful architecture of Data Grid Service. Transforming XML lists using XSLT allows other applications to consume the data in a suitable for them format. For example we created a simple XSLT transformation to provide a RSS feed with information about newly added impediment messages. The same we exposed the RDF representation about current projects, stories and tasks.

IV. Evaluation

A. Performance

To test the performance of the Data Grid Service regarding relationship processing functions we evaluated the service response time while retrieving subordinate XML list items within a simple relationship. The evaluation was performed by measuring the service response time on a local machine after request for all users within a specific project (as in the example application from the section I) using the URI *http://dgs.example.org/projects/5/users*. The number of items within users list varied from 100 to 5000, which corresponds to the objects count in the middle-size Web 2.0 application. The measures were performed on a service, hosted in ASP.NET Development Server on a PC with Intel Core2 Duo 2.66 GHz CPU 3 GB RAM and WD Velocity Raptor HDD (10000 rot/min). The results in Figure 6 show the linearity of response time function due to the straightforward implementation of XML filtering procedures using XPath. The current implementation gives acceptable results for smaller amounts of data, but should be revised for larger XML lists and faster response times. Currently we consider usage of RDF Tripple Stores, caching techniques and XML databases to meet the requirements of larger Web 2.0 applications and to optimize the overall service performance.

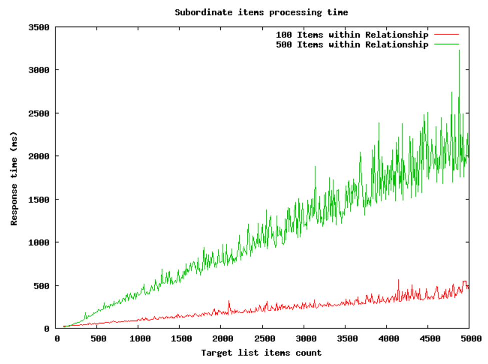


Figure 6. Measurements of service response time.

B. Development Process

We notice that the overall application complexity decreases due to the implicit resource oriented architecture and integration capabilities of Data Grid Service. The implementation of additional web services in order to expose application content or offer public data manipulation functions becomes redundant due to the RESTful architecture of Data Grid Service and its loose coupling with the application. The publishing of content in terms of Linked Data is implicitly supported through XSLT transformations and may be anytime reconfigured without changing the application code.

V. Related Work

During the last years a large number of distributed non-relational data storage solutions appeared. While meeting many web-specific requirements, the solutions concentrate on these main problems:

- High availability. An uninterrupted access to the stored data is especially important in Web 2.0 (business) applications to serve the customers around the world and to any time.
- Scalability. The time delay of read/write functions is stable even if maintenance routines are running or the number of clients emerges.
- Simple data modeling. The built-in support for key-value pairs or schema-free content simplifies the implementation of data driven Web 2.0 applications.

Following we discuss some interesting solutions providing the mentioned functionality and that are related to our approach.

- *CouchDB* - The Apache CouchDB Project [10] is a document-oriented storage solution accessible over HTTP in the RESTful style. The documents maintained by CouchDB are objects containing a variable number of named fields. The absence of document schemas makes the solution flexible for often data structure changes and new document types. Availability and robustness aspects are greatly solved, the manipulation of content requires solid JavaScript skills, the modeling of relationships between documents requires mixing objects and their metadata, the retrieval functions must be manually defined.

- *WCF Data Services* - The Microsoft WCF Data Services [11] is a part of .NET Framework, enabling developers to expose the data on the Web in a RESTful way. The contents are addressed through URIs and may be retrieved in different formats like JSON or XML. The data is described using Microsoft Entity Data Model based on the Entity-Relationship-Model. The service offers an easy traversal of collections, items and relationships, the underlying storage may be either a relational database, such as Microsoft SQL Server or any other data source accessed with custom implementations of data source provider component.
- *Amazon S3* - Amazon Simple Storage Service (S3) [12] provides essential functionality to maintain data over the Web being accessed both through SOAP and over HTTP in the RESTful style. While Amazon S3 is used to store unstructured data, it is often accompanied by Amazon Simple DB [13] offering a storage solution to access structured information and objects metadata. The data stored in Simple DB is schema-free and is automatically indexed to optimize query operations.
- *Google Data API* - The Google Data API provides access to the data stored in Google products such as Spreadsheets or Calendar using Google Data Protocol [14]. Besides client libraries are available for many programming languages, abstracting the conceptual schema from the data serialization formats used for data transport. The second version of protocol used is fully compliant with AtomPub RFC 5023 [15]. The data access and management functions are fast and flexible, additional methods to retrieve or update partial entities are implemented.

The presented approaches concentrate mainly on scalability and performance issues, but do not provide built-in functions to annotation the content with metadata as well as to transform it into alternative representation formats, e.g., RDF/XML. However these issues are essential for data exchange and integration in modern Web 2.0 applications. In contrast, Data Grid Service simplifies the development of Web 2.0 applications by providing an implicit support for the mentioned functionalities.

VI. Conclusion and Outlook

In this paper, we presented the WebComposition/Data Grid Service as a web-based storage solution to meet the needs of modern Web 2.0 applications and to support the developer in the implementation and maintenance process. Particularly Data Grid Service facilitates the Web 2.0 application development by

- Fast and flexible data management methods
- Content manipulation in RESTful way
- Data annotations in RDF format
- Rich data modeling capabilities and schema-free data structures
- Support for different data representation formats (XML, JSON, RDF, N3)
- URI templates and associations handling for easier data manipulation
- Flexible architecture to extend the functionality

Existing applications take advantage of the built-in support for resource annotation and sharing capabilities of the service. In combination with further WebComposition components like Data Grid Service List Manager [16] (DGSLM) user input elements, e.g., XHTML forms can be automatically generated to manipulate the contents directly from the external web application. To secure single lists or items within the Data Grid Service WebComposition/ Identity Federation System (idFS) [17] may be used as identity provider component. Both DGSLM and idFS have been successfully tested and are used as embedded modules on the web page of Distributed and Self-organizing Computer Systems research group to publish the information about publications, projects and lectures. In order to improve service performance and shorten response times we are currently working on indexing the contents and caching techniques to avoid redundant parse procedures and accelerate the content delivery process. To support the collaboration between single service instances publish/subscribe mechanism is being developed in current research projects. We are also working to provide iteration and pagination functions to simplify content discovery and navigation process.

References

[1] T. O'Reilly. (2005, September) What is web 2.0? design patterns and business models for the next generation of software. http://oreilly.com/web2/archive/what-is-web-20.html. Last Access: 04.07.2010.

[2] K. Schwaber and M. Beedle, *Agile Software Development with SCRUM*. Prentice Hall, February 2002.

[3] R. F. Beeger, A. Haase, S. Roock, and S. Sanitz, *Hibernate: Persistenz in Java-Systemen mit Hibernate und der Java Persistence API*, 2nd ed. Heidelberg: dpunkt, 2007.

[4] A. Adya, J. A. Blakeley, S. Melnik, and S. Muralidhar, "Anatomy of the ado.net entity framework," in *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2007, pp. 877–888.

[5] *RSS 2.0 Specification*, RSS Advisory Board, http://www.rssboard.org/rss-specification. Last Access: 04.07.2010.

[6] G. Klyne and J. J. Carroll, "Resource description framework (rdf): Concepts and abstract syntax," World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210, February 2004. [Online]. Available: http://www.w3.org/TR/rdf-concepts/

[7] L. Richardson and S. Ruby, *RESTful Web Services*. O'Reilly Media, Inc., May 2007.

[8] J. Gregorio and M. Handley, *URI Template*, http://tools.ietf.org/id/draft-gregorio-uritemplate-03.txt. Last Access: 04.07.2010.

[9] J. Galloway, S. Hanselman, P. Haack, S. Guthrie, and R. Conery, *Professional ASP.NET MVC 2*. Wiley Publishing, Inc, June 2010.

[10] J. C. Anderson, J. Lehnardt, and N. Slater, *CouchDB: The Definitive Guide Time to Relax*. O'Reilly Media, Inc., 2010.

[11] M. Corporation, *WCF Data Services*, http://msdn.microsoft.com/en-us/data/bb931106.aspx. Last Access: 04.07.2010.

[12] Amazon, "Amazon s3 developer guide," Amazon, Tech. Rep., 2010. [Online]. Available: http://aws.amazon.com/documentation/s3/

[13] D. Robinson, *Amazon Web Services Made Simple: Learn how Amazon EC2, S3, SimpleDB and SQS Web Services enables you to reach business goals faster*. London, UK, UK: Emereo Pty Ltd, 2008.

[14] Google, *Google Data API*, http://code.google.com/intl/de-DE/apis/gdata/. Last Access: 04.07.2010.

[15] J. Gregoric and B. de hOra, *RFC 5023 - The Atom Publishing Protocol*, http://tools.ietf.org/html/rfc5023. Last Access: 04.07.2010.

[16] R. Sommermeier, A. Heil, and M. Gaedke, "Lightweight data integration using the webcomposition data grid service," in *First International Workshop on Lightweight Integration on the Web (Composable Web'09) in conjunction with the 9th International Conference on Web Engineering (ICWE 2009)*, San Sebastian, Spain, 22.-26. Jun 2009, pp. 30–38.

[17] M. Gaedke, J. Meinecke, and M. Nussbaumer, "A modeling approach to federated identity and access management," in *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*. New York, NY, USA: ACM, 2005, pp. 1156–1157.