M. Gaedke, A. Heil GET /dgs HTTP/1.1 Host: www.WebComposition.net Hawaii International Conference on System Sciences (HICSS-42), Waikoloa, Big Island, Hawaii, USA, 5.-8. January 2009, ISBN 978-0-7695-3450-3; ISSN 1530-1605.

GET /dgs HTTP/1.1
Host: www.WebComposition.net

Martin Gaedke[1]
[1]Chemnitz University of Technology, Germany
Faculty of Computer Science
{firstname.lastname}@cs.tu-chemnitz.de

Andreas Heil[1,2]
[2]Microsoft Research, UK
Computational Science Group
v-aheil@microsoft.com

Abstract

During the last years, the asset costs for storage have been decreasing continuously. Coincidentally, the demand to create and publish data in the Web has grown in an unprecedented manner. Within the Web 2.0, the user became an active part by creating, publishing, changing and annotating data and its related metadata in a wide variety of new kinds of applications. Many data management but also architectural decisions for such applications are driven by distribution and semantic aspects. In this paper, we present how the WebComposition Data Grid Service (WebComposition/DGS) emerges new kinds of data-centric applications as a REST-architectural style component within the context of Web 2.0 but also satisfies requirements within traditional SOA-based business scenarios.

1. Introduction

Within the idea of Web 2.0, data is a substantial factor [1]. Architectural decisions are based on distributed and semantic aspects about data, its related metadata and its overall availability but also accessibility. Competitive advantages arise through data, the way it is generated and the way it is made available. A typical characteristic of the data is that a large amount of it is created by the users themselves. Some of the best known and commonly used representatives are eBay auctions, Amazon reviews, Flickr photostreams, Last.fm scrobblings or Twitter feeds. This data is not created by an editorial team but rather by the users directly. Consequently, this has led to a vital change of the digital identity of users in the Web. Originally, the user's homepage was the one and only place where all information was stored and published, since it was the only place where the user typically gained write access in the Web. Nowadays, the digital identity of a user is scattered all over the Web. Photos are hosted with Flickr, news articles are posted on a weblog such as Blogger, microblogging and instant messaging is achieved through Twitter feeds, links are archived within del.icio.us and contact details are stored in social network portals such as LinkedIn or Plaxo (see Figure 1). Various portals such as Facebook or MySpace provide functionality to mesh up this scattered data into one single Web page or a Web site as substitution for the personal homepage. While the data of a user formerly resided on this dedicated, central place where he was allowed to create data, it is now conflated from multiple data providers, possibly stored in data formats the user has no direct influence on.

Figure 1. Digital identity through multiple data providers

The usage of this distributed, heterogeneous data sources is rather limited. If ever, each data provider offers a different way of interacting with the stored data in other applications. This circumstance makes the efficient development of applications based on those data sources often very unpredictable, if not impractical at all. The question now is how to interact with the data, how to link the data from the various data sources in a meaningful way to each other and how to reuse this data in other applications.

In this paper we discuss the WebComposition Data Grid Service (WebComposition/DGS) component. Our main contribution here lies in its capability to implicitly manage metadata based on Semantic Web

standards as first-class constructs. We will especially focus on the capability to use this metadata and to make use of the concept of linked data within different architectural styles. Subsequently, we discuss the extensible core elements of the WebComposition/DGS that allow a high degree of reuse to reduce development costs of systems dealing with metadata, irrespectively of the underlying architecture style.

2. Motivation and Background

2.1. Example Scenario

During a code review, Clark, a software engineer in an international software company came along with an idea for a tool to increase the productivity within his team. To make this tool available to all team members, he decided to make this tool available within the company's intranet. He set up a Web server and wrote this little, Web-based application in his spare time. At this time, the Web-server's address was known only by his team members and Clark did not care a lot about security. Also, the tool stored all the data in plain text files, since Clark simply did not want to spend too much time on this.

Several weeks later, during an internal architecture review of the company's latest product, he accessed the tool via his Web browser to show the information that was requested from him. Several colleagues became very curious about that tool, they had never seen before. They realized the added value by this tool and asked Clark, if their teams might use it as well. Clark was flattered and agreed. He also realized that the tool was not capable of dealing with multiple data sets from different teams. So he spent a remarkable portion of his spare time to change the tool's way of dealing with data. At this point he thought a database might be a good solution to dealing with the increasing amount of data.

Once he was contacted by the manager of the internal tools division, who asked Clark to make this tool available to all employees within the company. This increasing degree of awareness also required some additional functionality to deal with security aspects since not all users should have been able to see all the projects managed by this tool. Clark had to implement additional functionality. Luckily, a developer from the internal tools division supported Clark to modify the tool to use the company's internal user directory.

During the annual review, the company realized that the tool caused an incredible increase in performance within the various product groups. The management thus realized the potential of this tool and that it might be an appropriate supplement to the company's portfolio of Web-based services. Hence, they decided to offer this as a new service. Based on the company's user directory, a fixed integration with the storage solution, a data structure that was fitted with the company's internal operations and the integration into several in-house products, the tool had to be redesigned and newly developed, which caused not to be underestimated costs for the company. The results were the great idea behind the tool and a well designed system that however, did not succeed on the market for a long time.

The difference was indeed, that the tool, as long as developed and used in-house, underwent a permanent evolution. Since it was Web-based, it is not possible to roll-out new versions of the tool on a regular basis. Changes have been introduced gradually. Hence, it adapted to new requirements and technologies. While Clark was aware of other in-house products and their related data formats and metadata, he developed plug-ins that enabled the tool to connect to various other tools used by the product groups. The service offered by the company was however not capable to integrate with many other tools. In fact, the easy extensibility and in particular, the integration with other tools, was the secret of the success of Clark's tool.

2.2. Data & Semantic

Based on [2], we define a catalogue of essential aspects and relations among these structures to be considered when designing a data-centric Web-based application.

Data: The creation, maintenance and handling of data must be easy and reliable. Storing the data in a corresponding data storage and simple querying the data are key features, which the solution must provide. The underlying technology must not restrict the user in terms of content to be stored and should provide the possibility of different views on the data. Changes or further developments in the underlying data processing and storage solutions should not affect already established applications based on the service.

Systematic creation and structuring of data: While in the context of Web 2.0 flexibility is eligible, the systematic creation of structured data is required to address business-scale scenarios, where the integrity of data is indispensable. In certain cases, the user might be forced to adopt a specific structure for the data, e.g. for constraining structures and content or for validating the data. The systematic creation and structuring of data, thus allows defining how symbols can be combined.

Metadata: Metadata is required to describe the data itself. It must be provided in a machine-readable

format. The metadata must provide information for understanding what one needs to know about data received from other sources, in order to proceed intelligently with the data. Also metadata is required to provide meanings of syntactically valid collections of data.

Linked Data: Data needs to be identified, published and linked. Based on its metadata, useful information has to be provided for connecting data. Similarly appears the idea of links in HTML (especially automatically generated trackbacks and pingbacks in the weblog domain): data and data clouds must be connected for lookups and traversing the data. Metadata is thus required to step through the data.

Security: Security aspects are important for applications within a personal scope up to business-scale scenarios. Applications dealing with personal or business data must guarantee both the integrity of the data itself and the protection of privacy. The absence of a security mechanism, e.g. in a business environment, might result in financial risks for the particular company and is thus a key requirement for the particular solution.

Reuse and Integration: The reuse of the components includes easy deployment, extensibility of the system, and the avoidance of one-off efforts. How easy can the solution to be used in already existing solutions and how much effort is required to achieve this, are key factors, especially in the Web 2.0 context, which must be considered carefully. It is important that a solution supports the reuse of existing data and provides capabilities needed to integrate components within the context of another application and within different architectural styles.

3. The WebComposition Approach

The WebComposition system was first introduced during the WWW6 conference in 1997 [3] as an object-oriented approach, especially for the discipline of Web Engineering [4] and in contrast to the discipline of traditional Software Engineering. The WebComposition approach was continuously developed up to today, where it abstracts the development and evolution of Web-based applications that are composed out of Web components. The WebComposition approach describes only the development and evolution process, the concept of composing and reusing Web components, but does not address the concrete technology applied to create the solution.

The Web components used to create a Web-based solution address different perspectives of an application: the content-perspective includes aspects related to data and semantics, the UIX perspective contains aspects related to the user interface experience and the DSA perspective contains especially aspects related to distributed system and architecture behavior.

The two core concepts of the WebComposition approach focus especially on two different perspectives of reuse. The development of Web components for reuse focuses on the creation of units that implement a certain perspective or corresponding aspects and, on the other hand, the development of solutions by reusing existing Web components to create complete Web applications/systems by composing existing Web components. To develop the WebComposition/DGS, as a central component of the 4th Generation of the WebComposition approach, we also applied the lessons learned from our previous work, the WebComposition Service Linking System and the therefore developed CRUDS-Service [5, 6].

3.1. WebComposition/DGS

The WebComposition/DGS is especially designed to meet today's requirements for developing distributed application in the Web, especially following the concept of Representational State Transfer (REST) architecture style. REST refers a set of constraints that define the distributed architectural style [7]. Resources are addressed using unique identifiers to which access is given through a uniform interface. In contrast to a service-oriented architecture, manipulation of resources is performed through stateless interaction with their representations. The most common REST-based architecture is the Web itself, using the HTTP protocol to provide a uniform access to resources [8].

The WebComposition/DGS allows to put new ideas into practice very easily by using these REST principles but also common SOA-based approaches. We find multiple interfaces for different scenarios: while facing the particularities of Web 2.0 and taking into account the standards of the Semantic Web, the WebComposition/DGS is furthermore designed to be also applied in traditional business scenarios.

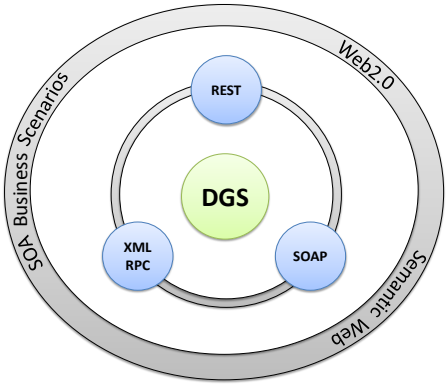

Figure 2. WebComposition/DGS corner pillars

The core interfaces (cf. Figure 2) provided are (i) a HTTP-based interface to be used within REST-like architecture styles, (ii) a simple XML/RPC interface to be used in simple ad-hoc implementation and (iii) a SOAP interface, supporting document/literal SOA-based architectures suitable for most business scenarios.

Designed to be suitable for a variety of application scenarios, the WebComposition/DGS comes as a modularized component [9] that can be easily extended and adapted for specific needs. Before showing example scenarios where the WebComposition/DGS is already successfully applied in productive system, we discuss the key functionality of the WebComposition/DGS regarding the various aspects already identified in Section 2.2.

4. WebComposition/DGS Internals

In this section we will discuss the core features of the WebComposition/DGS. We will especially focus on the usage within REST-like architecture styles; however, the described functionality is accessible also through the provided SOAP and XML-RPC interfaces.

4.1. In Direct Touch with the Data

Regardless of whether we are facing a service-oriented architecture [10], a REST- like architecture style [7] or merely an unconstrained architecture, we always deal with the concept of resources when looking at a Web-based application/system. Therefore, we enforce a strict usage of Uniform Resource Identifiers (URI) [11] within the WebComposition/DGS. Each resource created, regardless whether data or metadata, can be accessed via its strictly composed URI such as depicted in Figure 3 below.

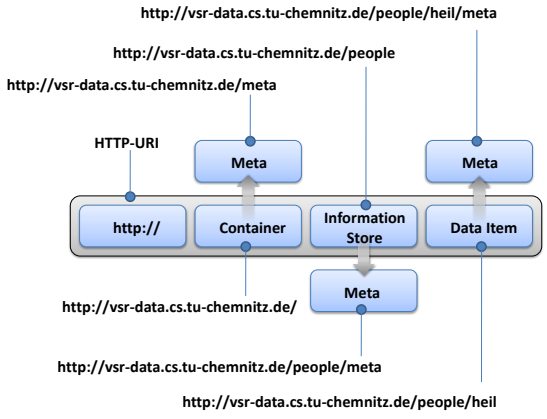

Figure 3. WebComposition/DGS URI concept

Each WebComposition/DGS service represents a *container* for one or many *information stores*. An information store is the access point, at which users can add and query data. One could understand an information store as a list or set, where related data is logically grouped together. Within an information store, each *data item* stored in it can be accessed via its own URI. For each of the concepts (container, information store and data item) the key path segment *meta* can be concatenated to access the related metadata.

4.2. Create and Structure Data

In some ad-hoc scenarios, the user might create data in a quite easy fashion as Clark did in our example during his initial approach in Section 2.1. On the other hand, sophisticated business scenarios might require the creation and validation of structured data.

The WebComposition/DGS component supports both scenarios: Data is processed by a so-called *data adapter* (cf. Figure 4). This exchangeable component of the WebComposition/DGS is the core element for creating and manipulating data.

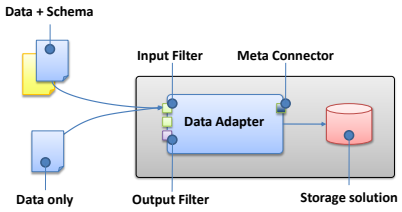

Figure 4. WebComposition/DGS data adapter concept

The concept of the data adapter is based on two major features: (i) Extensibility allows to create own data adapters for new kinds of data. For the current version of the WebComposition/DGS, data adapters for XML and HTML are already provided. Additional data adapters however can be developed and specified via a configuration file (cf. Figure 5). (ii) Customization of data adapters via *input* and *output filters*. Filters specify the format of data as well as the kind of accepted data. Additional filters can be developed and specified via the configuration file. Hence, the system can be simply reconfigured if new requirements arise. This even allows specifying an output format different from the original input format and thus provides a high flexibility regarding the data representation. Transforming the data to or from the internal representation of the data adapter is incumbent upon the corresponding filter.

The provided XML data adapter supports creation of both, structured and unstructured data. As such, ad-

hoc scenarios can be realized very easily. Regarding the interface for REST- like architecture style, the HTTP verbs are used as follows:

GET queries a resource at the given URI. The representation of the resource depends on the output filter specified in the configuration file of the WebComposition/DGS instance. By specifying a *accept* HTTP-header in the corresponding HTTP-request, we indirectly specify which data adapter to use. However, the concept of the data adapter is transparent to the client that simply requests any resource from the service. If no accept header is specified, the WebComposition/DGS applies the default data adapter.

POST allows for the creation of new information stores on non-existing URIs. The XML data adapter, which is the default adapter provided in the current WebComposition/DGS implementation, accepts a XSD schema send along with this request. The schema is then used to validate the data added to this newly created information store. Additional settings regarding the validation of the data can then be applied by adding further metadata as we will see in Section 4.3 below.

New data items are added using *PUT* on the information store's URI. The *content-type* specified in the HTTP-header specifies which data adapter is used to create the data item. Data items, but also complete information stores, can be deleted by sending *DELETE* to the corresponding URI.

The SOAP interface can be used for document/literal SOA to achieve the same functionality. However, the functionality here is contained in the SOAP message, rather than in the protocol semantic. To allow easy integration with already existing SOA systems, a *SOAP adapter* is specified equivalent to the procedure of the data adapter connecting to data adapter (cf. Figure 5). Hence, the WebComposition/DGS can be easily customized to fit into already existing SOA-based systems, while reusing existing components within the service.

```
<webComposition>
  <dataGridService>
    <dataAdapters>
      <dataAdapter
        type="WebComposition.Dgs.Content.Data.XmlDataAdapter,
              WebComposition.Dgs.Content,
              Version=1.0.0.0,
              Culture=neutral,
              PublicKeyToken=null"
        inputNotation="TEXT/XML"
        outputNotation="TEXT/XML"
        contentType="text/xml">
      <metaData inputNotation="RDF/N3"
                outputNotation="RDF/XML"/>
      </dataAdapter>
      ...
    </dataAdapters>
    ...
  </dataGridService>
</webComposition>
```

Figure 5. Example data adapter configuration

4.3. Metadata is Data

Following [12], we understand metadata as first class resource within the WebComposition/DGS. Therefore, metadata can be accessed via its dedicated URI. Data and metadata are treated as open data and open metadata (i.e. you have access to all data and metadata you create).

By default, all metadata is stored in the format of the Resource Description Framework (RDF). This comes in handy for two reasons: at first, due to the strict URI concept within the WebComposition/DGS, each resource (including metadata) is identified by its URI. Therefore, each resource can appear in a RDF triple as subject and/or object. This especially allows us to use metadata to describe metadata again. Secondly, RDF data is machine readable and accessible by a wide audience. Also the complexity of the metadata is not restricted, e.g. compared to the Exchangeable Image File (EXIF) metadata [13]. It is useful, that additional information can be simply added by inserting additional RDF triples. By default, each created resource is supplemented with the Dublin Core [14] metadata, stored as RDF triples as well. These 15 standardized attributes are used to describe and classify Web resources using common characteristics of resources. This information base is supplemented by the *WebComposition/DGS metadata vocabulary*. This data differs for each resource type and can be extended through data adapters. The container instance provides information about the contained information stores, an information stores again provides additional information about the contained data items such as content-types and item count. Additional metadata might be generated through the data adapter processing the data. For instance, it might be useful to extract EXIF metadata contained in images and store it directly as metadata with the corresponding resource when uploading digital photos to a WebComposition/DGS service.

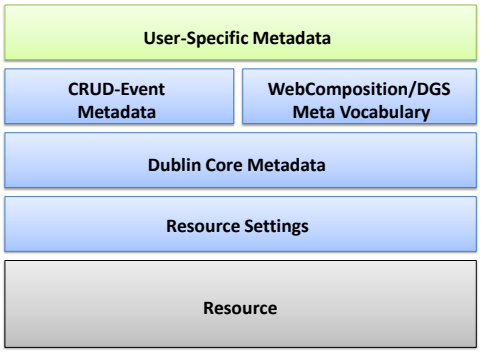

Figure 6. WebComposition/DGS information stack

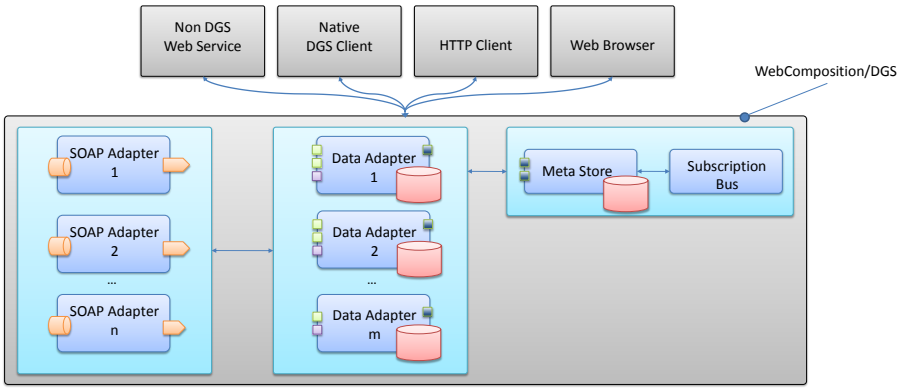

Figure 7. WebComposition/DGS component interactions

A further aspect within the WebComposition information stack (cf. Figure 6) is the *CRUD-Event metadata*. CRUD stands for create, read update and delete and describes the core operations performed on data [15]. Each operation on a resource is thus tracked and stored as CRUD metadata. This is especially helpfully in addressing the question of provenance, since the representation in RDF is already suitable to process the information [16]. If we consider this, the CRUD-Event metadata also addresses the issues of provenance in SOA-based systems [17] in general. Easy access to the events is granted through CRUD-Event RSS feed as depicted in Figure 8. By publishing the events to the feed, arbitrary clients can subscribe to the feed and monitor activities on the data. As metadata is understood as a first class resource, additional, user-specific metadata can be added in the same way as data itself. Also for metadata, we can specify different input and output filters (cf. Figure 5). By default, the WebComposition/DGS supports RDF/XML and Notation3 (N3) as metadata formats. Additional filters can be created and specified.

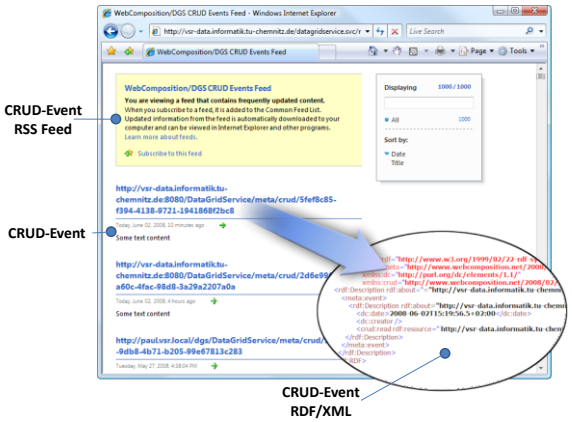

Figure 8. CRUD-Event RSS feed

Finally, the WebComposition/DGS information stack provides the possibility to store settings for resources, such as the service itself. Settings usually describe the behavior of a system or a resource. Since everything within the WebComposition/DGS is understood as a resource, settings are stored as metadata describing this resource. We will subsequently discuss the usage of metadata for settings based on two common settings of the WebComposition/DGS.

As the WebComposition/DGS supports the creation of structured data, the XML data adapter supports the validation against XSD schemas (cf. Section 4.2). It is possible to change the scope of the validation by using the settings to no validation at all, validating the submitted data (a-priori validation of the data) or validation of the complete XML structure (a-posteriori of the data after inserting into the XML structure). The default validation, applicable for any newly created information store, is set in the configuration file as seen in Figure 9 below.

```
<webComposition>
   ...
  <dataGridService>
    <defaultSchemaScope scope="None"/>
   ...
   </dataGridService>
   ...
</webComposition>
```

Figure 9. Default schema validation

Using the N3 input filter, the default settings can be changed by putting the metadata through a simple HTTP-request to the information store's URI containing the N3 statement in Figure 10 below. By adding this metadata, we can simply set the validation for a particular information store to the a-priori validation described above.

```
@prefix dm: http://www.webcomposition.net/dgs/meta/.
<http://vsr-data.cs.tu-chemnitz.de/people>
dm:schemaScope
„Element".
```

Figure 10. Schema validation metadata in N3 notation

As we can see in Figure 3, each data item within an information store provides its own URI. To address data items within a information store we apply the concept of *URI templates* [18]. To illustrate this concept we assume that the XML data for the information store *http://vsr-data.cs.tu-chemnitz.de/people* is structured as seen in Figure 11 below.

```
<?xml version="1.0" encoding="utf-8"?>
<people>
    <person>
        <name>Heil</name>
        <firstName>Andreas</firstName>
        <title>Dipl.-Inform.</title>
        <office>1/B204</office>
        ...
    </person>
    <person>
        <name>Gaedke</name>
        <firstName>Martin</firstName>
        <title>Prof. Dr.Ing.</title>
        <office>1/B319</office>
        ...
    </person>
    ...
</people>
```

Figure 11. Example information store data

In our example we want to address the data items (i.e. the person elements) via URIs such as *http://vsr-data.cs.tu-chemnitz.de/people/Heil* and *http://vsr-data.cs.tu-chemnitz.de/people/Gaedke*. To do so, we use an URI template that specifies the URI we require as well as a XPATH expression mapping the requested data to the specified URI. The corresponding URI template, as it is submitted, is shown in Figure 12.

```
@prefix dm: http://www.webcomposition.net/dgs/meta/.

<http://vsr-data.cs.tu-chemnitz.de/people>
dm:urlTemplate [meta:url "people/{value}";
dm:xPath "/people/person[name='{value}']" ].
```

Figure 12. URI template in N3 notation

It is important to know that all URIs specified for resources serve also as unique identifiers to address the resource in a SOA-based scenario using e.g. document/literal SOAP.

4.4. Linking Data and Metadata

By combining data and metadata we address the challenges identified in Section 1 how to link data to each other in a meaningful way. The concepts introduced so far help us in achieving this goal. If we apply the WebComposition/DGS as component in a merely SOA-based system, each resource is still identified by its own URI. Hence, all resources can be referenced transcending the boundaries of information stores when creating linked data [19].

Data as well as metadata are combined by specifying a XSL transformation. A depiction of the process how the data is combined is given in Figure 13. Hence, not only the metadata but also the data itself are provided in a machine-readable format. Data from different information stores as well as the corresponding metadata is combined by specifying a XSL transformation to combine and format the data.

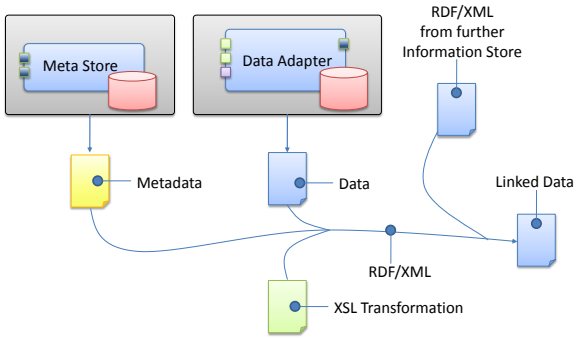

Figure 13. WebComposition/DGS linked data

For instance, we can use the contact details out of our information store for people seen before, add metadata such as geospatial information and combine this information with project and publication data stored, to create machine-readable Friend-Of-A-Fried (FOAF) files using the RDF technology [20]. For both projects and publications we already use location information (e.g. about the project or conference location) that again can be used for creating the linked data. As there is no common security concept for REST-like architecture styles, various approaches are applied for different scenarios. In the majority of cases we will see the usage of HTTPS using SSL encryption for point-to-point encryption and authentication. The Backpack API [21] extends this by sending a 40-byte SHA1 hash token along with the request content to authenticate the user. The Amazon S3 [22, 23] also uses a standard challenge-response approach by sending a token, the so-called AWS Access Key ID. along with the HTTP-request as a supplementary HTTP-header. In addition, a signature in the form of a HMAC-SHA1 [24] token is created and included to ensure authentication of the request. Both, the AWS Access Key ID as well as the signature are also used for SOAP-based invocation of the service.

Figure 14. Example for creating linked data

4.5. Reuse with Security for REST and SOA

As you learned, reuse is the central idea within the WebComposition approach. Therefore, we enable another WebComposition component, the Identity Federation System (idFS), as an optional security component along with the WebComposition/DGS. The idFS provides a well proved security infrastructure based on the active and passive requestor profiles of the WS-Federation specification [25]. The idFS provides mechanisms to access Web services and resources through security token service access control and identity providers, single sign on and federating components, i.e. accessing resources across organizational boundaries [26, 27, 28].

The idFS system is based on a role based system where permissions and restrictions are based on a fine grained security system. While idFS follows the WS-Federation specification and thus can be applied immediately to SOA-based scenarios, we use SSL encryption for REST-like architecture styles using the HTTP protocol. The idFS allows us to define groups and to issue tokens for special purposes. This includes for instance granting rights to certain resources, rights for a certain time or rights for a certain number of actions. Issued tokens describe the roles a user is member of. For the WebComposition/DGS we specify the rights following a bottom-up approach. Rights are defined for any anonymous request first, which means when a request does not contain any tokens at all. In the following, we define additional users and roles to grant specific rights on resources. Hence, you can, for example, hand out trial access (such as access limited in time or number of read/write requests). The important point here is to realize the idFS as an optional component. If not required, the WebComposition/DGS can be used without any security concepts at all. If required, the idFS can be deployed and configured afterwards. This can also happen during the evolution of the system as it was required within our initial example in Section 2.1 where Clark's had to implement user access.

The idFS is currently deployed and actively used on various sites such as the IT-Management and Web Engineering Research Group's site [29], the Web Engineering community portal [30] as well as the portal of the International Society for Web Engineering [31]. Resources offered by WebComposition/DGS instances deployed within further organizations thus be made accessible to the sites already supporting idFS in a very convenient way, without the need of creating and managing additional user profiles or logins.

This introduced concept allows us to use one WebComposition/DGS instance in both SOA-based scenarios as well as in Web 2.0 scenarios based on a REST-like architecture styles. Access to the data is thus not restricted due to any architectural decisions.

5. Related Work

Several commercial approaches currently become noticeable to deal with the new requirement of creating, storing and publishing large amounts of data. The Amazon S3 [22, 23] and the Amazon SimpleDB [32] provide functionality to store, process and query data over the Web. The services provide the basic functionality of databases. Both services provide SOAP and HTTP interfaces for service-oriented and REST-like architecture styles. While designed for relatively small amounts of data, the SimpleDB is also publicized as storage for metadata for a corresponding Amazon S3. While the WebComposition/DGS is designed as a Web component, the Amazon S3 as well as the SimpleDB are hosted exclusively by the corresponding provider while the service itself is sold.

In contrast, Microsoft's ADO.NET Data Services [33] is an extension to Microsoft's .NET Framework and can be understood as additive component to the .NET Framework. The ADO.NET Data Services include a set of patterns to interact with data by using HTTP, addressing resources and linking data among services. This approach lines up well with the REST-like architectural style. In addition, the ADO.NET Data Services also provides a RPC-like interface. Being one of the approaches, which meets the requirement of a REST-like architecture style as defined in [7] the most, the ADO.NET Data Services are on the other hand especially designed to connect to Microsoft's SQL Server and are thus bound to a single data provider.

Backpack [21] also provides basic functionality to store and maintain data in the form of lists such as notes, images and files on the Web with a strong focus on visual representation. A HTTP-based interface is complemented by a set of wrappers for multiple

programming languages to access the service. Similar to Amazon's approach, Backpack is merely offered as a commercial service. It is also not designed to support SOA-based scenarios as the WebComposition/DGS does.

The WebComposition Service Linking System CRUDS service [5, 6, 34] is a generic, SOAP-based service to store, manipulate and publish data. As a predecessor of the WebComposition/DGS, many ideas influenced the design of this component. However, the CRUDS service does not provide the flexibility of the WebComposition/DGS and is a purely SOAP-based Web service.

Menagerie [35] is a relatively young software framework targeting similar issues as the WebComposition/DGS, such as combining data scattered across multiple data providers in the Web and manipulating this data with standard applications. In contrast, this approach focuses on personal data specifying the Menagerie Service Interface and the Menagerie File System. The WebComposition/DGS however also targets business-scale scenarios and provides a higher flexibility. As such, the support of idFS as security infrastructure component for the WebComposition/DGS is optional. The WebComposition approach does not restrict the system to any of its components. Google's OpenSocial [36] focuses especially on social network data and defines an API that allows any social network platform to host third party social applications. Also, the Goolge Data API [37] defines a more general approach by providing a set of simple APIs for reading and manipulating data on the Web.

While some of the features of the WebComposition/DGS are closely related to the previously mentioned systems, it is unique with its combination of implicit usage of metadata, extensibility through its component based approach, and transparent usage within different architectural styles suitable for both, REST-like architecture style based systems in the context of Web 2.0 and SOA-based business scenarios.

9. Summary and Outlook

In this paper, we presented the WebComposition Data Grid Service (WebCompostion/DGS) as the first component of the 4th generation of the WebComposition approach. The data dispersion within the Web, e.g. seen when personal data is used to create ones digital identity, introduces new issues in maintaining and meaningfully linking data on the Web. The WebComposition/DGS addresses these issues with the consequent application of Web technologies, WS-* specifications, standards from the semantic Web and a design that provides a high flexibility in terms of reuse, extensibility and customization. The WebComposition/DGS appears to be a component suitable for REST-like architecture and SOA style-based systems and even allows integrating systems of different architecture styles. We showed the Web component-based approach based on the Identity Federation System (idFS) that provides a well-established security infrastructure component for the WebComposition approach.

Beside using the WebComposition/DGS as central component of the Distributed and Self-organizing Systems Group at Chemnitz University of Technology's Web application, the component is recently applied in multiple research project. Among others we are currently working on additional data adapters for common data types, an information flow system, based on the WebComposition/DGS and the integration of additional security infrastructure components.

The WebComposition components, demos and additional documentation are accessible through http://www.WebComposition.net/DGS and http://www.WebComposition.net/idFS.

10. References

[1] T. O'Reilly, "Web 2.0 Compact Definition: Trying Again", http://radar.oreilly.com/2006/12/web-20-compact-definition-tryi.html (05-29-2006).

[2] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice. Boston, San Francisco, New York: Addison-Wesley, 2003.

[3] H.-W. Gellersen, R. Wicke, and M. Gaedke, "WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle", 6th International World Wide Web Conference, Santa Clara, CA, USA, 1997, pp. 1429-1437.

[4] Y. Deshpande, S. Murugesan, A. Ginige, S. Hansen, S. Schwabe, M. Gaedke, and B. White, "Web Engineering", Journal of Web Engineering, vol. 1, pp. 3-17, 2002.

[5] M. Gaedke, M. Nussbaumer, and E. Tonkin, "WebComposition Service Linking System: Supporting development, federation and evolution of service-oriented Web applications", 3rd Int. Workshop on Web-oriented Software Technology (IWWOST 2003), 2003.

[6] IT-Management and Web Engineering Research Group (MWRG), "WebComposition Service Linking System", http://mwrg.tm.uni-karlsruhe.de/wsls (02-10-2008).

[7] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", University of California, Irvine, 2000.

[8] L. Richardson and S. Ruby, RESTful Web Services: O'Reilly, 2007.

[9] A. Heil and M. Gaedke, "WebComposition/DGS: Supporting Web2.0 Developments With Data Grids", IEEE International Conference on Web Services (ICWS 2008), Beijing, China, 2008.

[10] C. M. MacKenzie, K. Laskey, F. McCabe, and R. Metz, "Reference Model for Service Oriented Architecture 1.0", http://www.oasis-open.org/committees/soa-rm/ (

[11] T. Berners-Lee, "Universal Resource Identifiers in WWW", http://www.ietf.org/rfc/rfc1630.txt (11-24-2007-2007).

[12] T. Berners-Lee, "Metadata Architecture", http://www.w3.org/DesignIssues/Metadata.html (05-31-2008).

[13] Japan Electronics and Information Technology Industries Association, "Exchangeable image file format for digital still cameras: Exif Version 2.2", 2002.

[14] L. Andresen, "Dublin Core Metadata Element Set, Version 1.1: Reference Description", http://dublincore.org/documents/dces/ (02-18-2008).

[15] H. Kilov, "From semantic to Object-oriented Data Modeling", First international Conference on Systems Integration, Morristown, NJ, USA, 1990, pp. 385-393.

[16] J. Futrelle, "Harvesting RDF Triples", International Provenance and Annotation Workshop (IPAW'06), Chicago, Il, USA, 2006, pp. 64-72.

[17] V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau, "Security Issues in a SOA-based Provenance System", International Provenance and Annotation Workshop (IPAW'06), Chicago, Il, USA, 2006, pp. 203-21.

[18] J. Gregorio, M. Hadley, M. Nottingham, and D. Orchard, "URI Template", http://tools.ietf.org/id/draft-gregorio-uritemplate-03.txt (02-06-2008).

[19] T. Berners-Lee, "Linked Data", http://www.w3.org/DesignIssues/LinkedData.html (02-20-2008).

[20] D. Brickley and L. Miller, "FOAF Vocabulary Specification 0.9", http://xmlns.com/foaf/spec/20070524.html (06-03-2008).

[21] 37signals LLC, "Backpack", http://www.backpackit.com/ (02-19-2008).

[22] Amazon Web Services LLC, "Amazon Simple Storage Service Developer Guide", http://docs.amazonwebservices.com/AmazonS3/2006-03-01/ (06-03-2008).

[23] F. Shanahan, Amazon.com Mashups. Birmingham, UK: Wrox Press Ltd., 2007.

[24] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", http://www.ietf.org/rfc/rfc2104.txt (06-03-2008).

[25] H. Lockhart, S. Andersen, S. J. Bohren, Y. Sverdlov, M. Hondo, H. Maruyama, A. Nadalin, N. Nagaratnam, T. Boubez, K. S. Morrison, C. Kaler, A. Nanda, D. Schmidt, D. Walters, H. Wilson, L. Burch, D. Earl, S. Baja, and H. Prafullchandra, "Web Services Federation Language (WS-Federation)", 2006.

[26] M. Gaedke, J. Meinecke, and M. Nussbaumer, "A Modeling Approach to Federated Identity and Access Management", 14th International World Wide Web Conference (WWW'05), Chiba, Japan, 2005, pp. 1156-1157.

[27] J. Meinecke and M. Gaedke, "Modeling Federations of Web Applications with WAM", Third Latin American Web Congress (LA-WEB 2005), Buenos Aires, Argentina, 2005, pp. 23-31.

[28] J. Meinecke, M. Nussbaumer, and M. Gaedke, "Building Blocks for Identity Federations", Fifth International Conference on Web Engineering (ICWE 2005), Sydney, Australia, 2005, pp. 203-208.

[29] IT-Management and Web Engineering Research Group (MWRG), "Home of the IT-Management and Web Engineering Research Group", http://mwrg.tm.uni-karlsruhe.de/ (03-06-2008).

[30] webengineering.org, "The Web Engineering Community Site - WebEngineering.org", http://www.webengineering.org/ (06-03-2008).

[31] International Society for Web Engineering e.V. (ISWE), "International Societe for Web Engineering e.V." http://www.iswe-ev.de/ (06-03-2008).

[32] Amazon Web Services LLC, "Amazon SimpleDB Developer Guide", 2008.

[33] P. Castro, "Project Astoria", The Architecture Journal, pp. 12-17, 2007.

[34] M. Nussbaumer, "Entwicklung und Evolution diensteorientierter Anwendungen im Web Engineering", Universität Karlsruhe (TH), Karlsruhe, 2007.

[35] R. Geambasu, C. Cheung, A. Moshchuk, S. D. Gribble, and H. M. Levy, "Organizing and Sharing Distributed Personal Web-Service Data", 15th International World Wide Web Conference (WWW 2008), Bejing, China, 2008, pp. 755-754.

[36] Google Inc., "OpenSocial", http://code.google.com/apis/opensocial/ (06-03-2008).

[37] Google Inc., "Google Data APIs", http://code.google.com/apis/gdata/ (02-17-2008).