Components for Growing the RESTful Enterprise

Andreas Heil[1,2], Johannes Meinecke[1], Martin Gaedke[1]

[1]Chemnitz University of Technology
09111 Chemnitz, Germany
[2]Microsoft Research Cambridge
CB3 0FB Cambridge, United Kingdom
[1]{firstname.lastname}@cs.tu-chemnitz.de, [2]v-aheil@microsoft.com

Abstract: For a modern enterprise, it is vital to be on the Web. Beyond offering human-readable Web sites, organizations increasingly use the Web as a media for machine-readable data about itself. With the help of technologies like XML feeds, RESTful Web services and semantic markup, new forms of enterprises models emerge in a bottom-up way. These models are easily consumable and facilitate the interaction with departments, partners and customers. Engineering good publishing systems is however extremely challenging. On the one hand, knowledge of many technologies is required; on the other hand, it must be easy to extend systems and data models in accordance to agile businesses. In this paper, we propose a framework of components for publishing dynamically growing enterprise models on the Web, present an implemented system and discuss its use in a case study.

1 Introduction

An enterprise model is a computational representation of an organization, covering aspects like its structure, activities or processes [FG98]. In recent years, there has been a tendency for enterprise data to grow in a bottom-up way rather than in a planned top-down way [TW06]. Simple lists, which were originally created to fulfil some personal organization need, are made available to others, get extended and become valuable assets for the enterprise. Technologically, this bottom-up form of collaboration is enabled through Web-based tools like Wikis, blogs, feeds and mashups. These technologies are suitable for both, in usage the Internet with public accessibility as well as usage in corporate networks and intranets. The dynamic growth of enterprise data in both scenarios is favoured by the principles of the Web, after which relevant resources have a URI and can thus be linked to and combined. In accordance to the Web's underlying Representational State Transfer (REST) architecture [Fi00], companies with such Web-enabled models could be called *RESTful enterprises*.

This way of exposing information comes with a number of advantages for the company. The data can be easily combined in end user-written mashups that are characterized by decreased implementation costs over traditional software development [An06]. Typical scenarios involve the combination of external sources, like geographical maps, with

internal data, like employee workplaces. The relative simplicity of standards like HTML and XML makes it easy to reuse the information in many places, as e.g. to display information on multiple web sites. This in turn improves consistency and lowers the cost to maintain the information, changing the future business behaviour [Yo07]. In addition, existing Web tools can be leveraged, as for example to provide search across the company's XML data sources [MM04].

In this paper, we concentrate on the question of how to build the necessary Web-based systems to expose enterprise models efficiently, rather than on the enterprise models themselves. In section 2, we examine the engineering challenges that arise in typical scenarios with the help of an example. From this examination, we derive the aim of our work: to identify reusable components for exposing enterprise models in a Web-compliant way. As our main contribution, we then specify a framework of components that can be instantiated to support a large number of standards and tasks in section 3. In section 4, we discuss our experience with the implementation of a number of these components, which we used to gradually build up a RESTful enterprise model. Section 5 contrasts our work with related approaches and section 6 summarizes the conclusions.

2 Exposing Bottom-Up Enterprise Models on the Web

For illustration purposes, we explain the general problem domain with an example scenario. The example covers the typical evolutionary bottom-up growth of information assets within organizations that start within small teams and turn into large-scale undertakings over time [TW06]. Figure 1 shows entities inside an organization that gradually emerge as Web-based representations, called *lists* in the following.

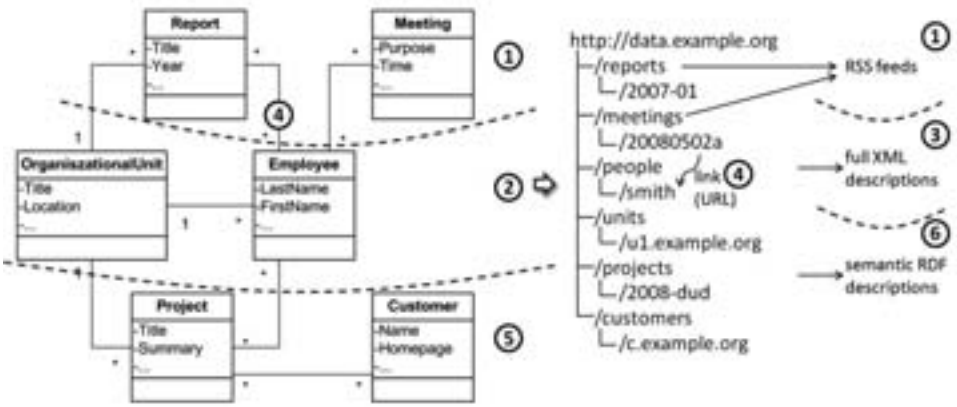

Figure 1: Example of a gradually growing enterprise model exposed on the Web.

In the beginning, a team sets up lists as ad-hoc solutions for managing reports and meetings. The content is published in the RSS format, which allows team members to subscribe to automatic notifications on any new list items and which enables other teams to integrate the lists in their homepages (1). Over time, this "model" is extended, as more

lists become available (2) and more data formats of the existing lists are offered (3). For example, in addition to the very generic RSS format, other XML formats might include geographical coordinates and domain-specific information that enable more powerful mashups. In order to improve the combined use of the data, multiple lists are then linked to each other (4). More precisely, URLs are included in the list items that point to other list items, as e.g. to point from reports the authoring employees. Data from existing project and customer management solutions is exposed in a similar way to be combined with the ad-hoc data sources (5). Future interoperability needs may demand the support for additional, semantically annotated formats (6). These can be searched, viewed and processed with a wide range of semantic tools, without the need for programming application-specific mappings.

The described way of exposing enterprise data can be seen as an application of the REST architecture style. REST comprises a set of design principles that are seen as the reason for the scalability and organic growth of the Web [Fi00]. These principles are applied here to achieve a similar advantage for enterprise models. A central requirement demands that every important entity must have an identifier so that it can be referred to. As shown in Figure 1, both lists and list items in the example are mapped to URIs. REST distinguishes between resources (the entities on the left side) and their various representations (the content types on the right side). The representation used depends on the demands of the requesting application. In the example, the same resource is reused in mashup engines, RSS readers, Web applications and Semantic Web browsers. All resources can be manipulated in a standardized way through the uniform interface of HTTP methods. In addition to read-access, this allows e.g. adding a new entry to the meeting list via an HTTP POST.

Whereas tools now become available for easily combining data sources on the Web [Ma08, Mi08, Ya07], exposing structured data beyond the simple publication of static XML files remains a largely unsupported task. [PZL08] list several design decisions to be made when programming RESTful services that include designing URIs, defining the interaction semantics of HTTP methods and choosing the data presentations. In this work we are interested in building systems that automate this process as much as possible to support the outlined target domain. We observe the following engineering challenges:

- **Gap between end users and technologies:** The initial data sources in the scenario are set up by users rather than a central IT-department. This corresponds to the findings of US Bureau of Labour Statistics [Us08], stating that 98% of users creating programs are end users without special IT-expertise who need to solve immediate non-technical problems. In contrast, the described final solution comprises a large number of technologies. While the benefits of e.g. Semantic Web formats may be highly desirable, the necessary knowledge to author them cannot be expected from the end user. Instead, technologies should be encapsulated in tools or parts of tools as much as possible.
- **Reusability of multiple resource representations:** As exemplified, the reusability of the exposed enterprise models requires the same resources to be accessible in many different representations. To free the user from the burden of managing differ-

ent representations, these should be generated automatically. For automatic mappings to work, structure must be imposed on the data. While unstructured content, as e.g. managed in Wikis, has the potential for organic growth, it cannot be reused easily and is therefore lost for applications beyond human browsing.

- **Dynamic growth of data structures**: The enterprise model in the example is gradually extended, as more data is added and linked to existing data. Corresponding systems must therefore support the effortless creation of new lists, including their structure and the mappings to their representations. Dynamic growth implies that this is possible at runtime and on the Web. Ideally, the process of extending the model can profit from the same simplicity and standard-conformance as used for manipulating individual resources.
- **Need for system extensibility**: In addition to the model, the system itself is subject to growth. In the example, this applies to the support for new formats (e.g. RDF), new data sources (e.g. customer management systems) and new client applications (e.g. search applications). Furthermore, the opening of the model to partners outside the enterprise may require adding security mechanisms that were unforeseen and unnecessary at the very beginning. The system architecture should reflect this need and allow for flexibility.

The observed challenges stress the need for solutions that comply with Web standards and that encapsulate these standards in reusable, extendable parts. Next, we therefore try to identify reusable components for exposing bottom-up enterprise models in a Web-compliant way.

3 The Data Grid Service Component Model

The WebComposition Data Grid Service (WebComposition/DGS) component model is a two-layer framework of reusable software artefacts for building customizable Web-based applications. A WebComposition/DGS component is a RESTful service for creating, managing and publishing lists of arbitrary data in the Web. It is based on a set of exchangeable sub-components to incorporate the latest technological developments. Exchanging these sub-components allows adopting the component itself to meet specific business needs. In the following we will first describe WebComposition/DGS component's building blocks (cf. Figure 2) before we will discuss the possible usage of the WebComposition/DGS component itself. Additional background information on the WebComposition/DGS component and its architectural description can be found in [HG08].

The Service Component (a) represents a central facet within the WebComposition/DGS component model. Incoming requests are accepted, processed and delegated irrespectively of their origination. Clients sending a request could be HTTP-based browsers, SOAP-based SOA components or legacy XML-RPC based clients that rely on so-called plain old XML (POX) invocations. Incoming data is processed by the Data Adapter (b). It is within the Data Adapter's responsibility to create and store data structures in form of lists using an external storage solution. Multiple Data Adapters could be specified within a WebComposition/DGS component, each responsible for a specific representa-

tion of the resources. Based on the requestor's needs, an appropriate Data Adapter is used to provide the corresponding representation of a resource. In case of a HTTP-based request this is achieved by HTTP content negotiation. Simultaneously to the creation of the data, metadata is saved along with the data. Unnoticeable to the user, a Meta Store component processes additional metadata (c). This includes e.g. creation and last update times, creator and other information that describes the actual data. The metadata already provided automatically by a Data Adapter can be completed by additional metadata added through the user. As the format of metadata varies from case to case, Input and Output Filter components (d) allow specifying the transformation of metadata formats into and from the internal representation within the Meta Store component. To be deployed in various business scenarios, an optional Access Control component incorporates the authentication and authorization functionality of external security components (e). Finally, optional sub-components can be added to a WebComposition/DGS component extending the Service Component's capabilities with additional unforeseen functionality (f).

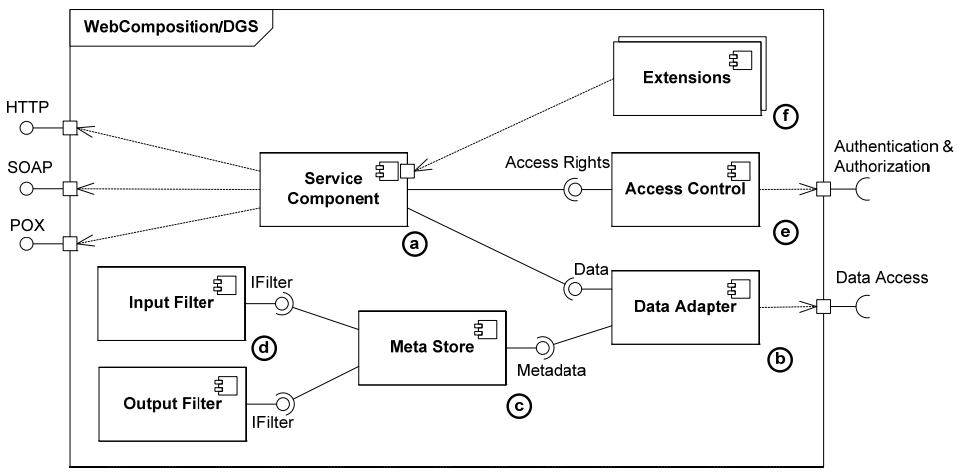

Figure 2: UML component diagram of WebComposition/DGS sub-components.

The components described thus far allow customizing a WebComposition/DGS component to a large extent. Once customized and configured, the WebComposition/DGS acts as a component by itself that can to be seen as a black-box, hiding its internal functionality (cf. Figure 3). A core concept of the WebComposition/DGS component (i) is its capability of being accessible in a uniform way through different types of clients. This includes standard Web-based applications such as common Web sites (ii), RSS feed readers as well as Semantic Web browsers. In addition, specialized clients can be used to access more specific functionality of a WebComposition/DGS component (iii). One example of this type of clients, a list managing application, will be discussed in the section 4. Based on a generic SOAP interface, derived from our previous work [MMG07], SOA-based clients can also access the WebComposition/DGS component (iv). The way enterprise models are stored depends heavily on the particular requirements and constraints of a business. Hence, this is foreseen as self-subsistent component within the system (v) allowing to apply database management systems or simple file based solution

– based on the particular requirements. This also allows reacting to the growing needs of projects that originally start small and increase in size over time. Another form of potential evolution is foreseen by the security component that provides external, centralized authentication and authorization functionality to complement the access control enforced within the WebComposition/DGS component (vi). Possible security components applied for this reason include, but are not limited to the Identity Federation System (idFS) [MNG05], Shibboleth implementations [In08] or the Active Directory Federation Services (ADFS) [Pi05].

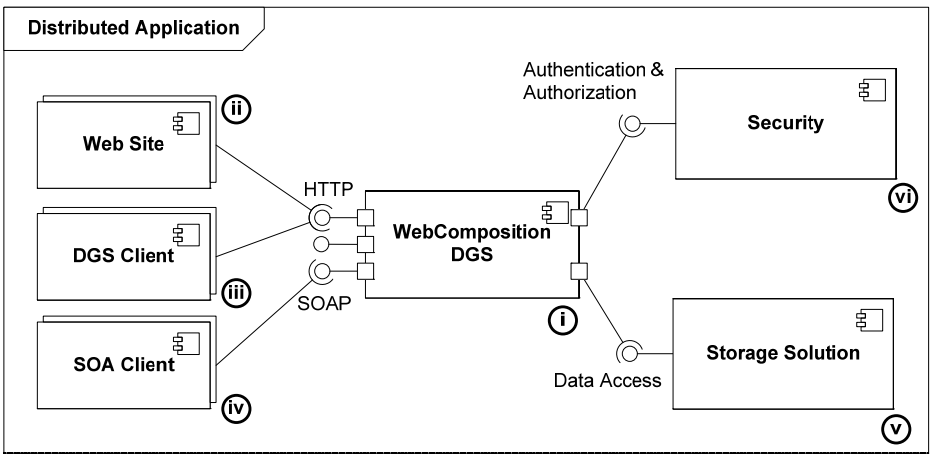

Figure 3: Distributed application based on a WebComposition/DGS component.

The proposed two-layer component architecture allows the disciplined and cost efficient engineering of Web-based application suitable to the current evolution stage. In the beginning, the out-of-the box functionality of a WebComposition/DGS component in conjunction with functionality of already available components could be sufficient to deal with scenarios where unstructured data is created ad-hoc. The bottom-up growth of this information as well as its increasing need for structures is tackled by exchangeable components addressing the particular needs. In the following section we will discuss this evolution of a system, based on a real data stock that grew over time.

4 Case Study

We now describe a project in which the presented component model was applied to bottom-up enterprise models. In the case at hand, the enterprise model represents the structure of a research group. Performed in the academic field, the nature of the exposed model is not restricted to the research domain, so that we consider the solutions applicable to enterprises in general. The study was conducted for 6 months in a production environment, using real, externally visible data. During this time, the model was gradually exposed and extended with new resources, new representations and new components according to the emerging needs of the group. A part of it was transformed from origi-

nally unstructured media, mainly managed with Wiki software. This old form of data management proved to be too hard to integrate and consume outside the Wiki itself. Therefore, an implementation of the specified WebComposition/DGS was used to successively expose data in accordance to Web standards and the REST principles. Over time, this included publications, courses, projects, student projects and people. The lists typically contain several hundred entries, describing both historical and currently active data. In addition, the model was extended several times to accommodate for new information needs (e.g. adding archive IDs to publications), to introduce links between different resources (e.g. relating publications and people) and to add new representations (cf. Figure 4).

(a) (b)

(c) (d)

Figure 4: Representations of resources from the WebComposition/DGS in an RSS reader (a), in a Semantic Web browser (b), in the DGS List Manager (c) and on a Web site (d).

The technical realization included the following components:

- As the foundation, we used our standard implementation [HG08] of the **WebComposition/DGS** that is based on the .NET Framework. In this implementation, subcomponents (cf. Figure 2) correspond to dynamic link libraries that are composed according to configuration files.
- The **XML Data Adapter** provided the necessary facilities for managing arbitrary XML lists. The adapter supports the creation of new lists at runtime as well as the enforcement of XML-Schemas to ensure that the data conforms to a given structure.
- In addition to the default file-based XML storage, a **Database Storage Solution** based on SQLite was developed to verify the exchangeability of storage solutions.
- Another component, the **Dynamic Transformation Adapter,** was used to generate new presentations from existing XML resources based on XSL stylesheets, as e.g. to generate RSS-feeds (cf. Figure 4a).
- Later, a **RDF Data Adapter** was developed to generate RDF representations of resources and metadata according to Linked Data guidelines [BCH07]. Once a list structure has been mapped to common ontologies, the model information can be reused in Semantic Web tools together with other data from the Web (cf. Figure 4b).
- For managing the model stored in the WebComposition/DGS, a **DGS List Manager** was developed. This generic client component retrieves the XML schemas from the list manager and automatically generates forms for editing resources (cf. Figure 4c).
- The integration into the research group's Web site was supported with a **PHP List Client Component**. The component encapsulates the technical details of rendering the data and generating the necessary links to other representations (cf. Figure 4d).

We summarize the lessons learned during the case study with respect to the engineering challenges outlined in section 2. The concept of encapsulating technologies in components proofed to be an important factor for end user support. With the developed system, new lists can be created ad-hoc and are automatically editable in Web forms, without any scripting or code deployment. They can also be integrated into Web pages without the need to know the involved internal components, transformations, protocols and formats. Whereas in our current system XML schemas and XSL stylesheets need to be specified when creating new lists, the architecture allows for adding more user-friendly components that automate this process further. The applied components also favored the reusability of the resources by automating the process of generating content representations. On the Web site alone, the data could be integrated at multiple locations for realizing different views on it, e.g. on personal homepages, on project pages and on central group pages. Furthermore, the study illustrated the system's potential for bottom-up data growth. Natural limits to flexible changes arose from the Web's need for stable URIs and from necessary updates of the corresponding mappings into formats and ontologies. The demand for system extensibility was met by the component-based architecture. As demonstrated, components were gradually added to the WebComposition/DGS, while the service was in productive use, i.e. integrated into the group's Web site.

The described systems are still in use and will be extended in future for further studies. A demo of the WebComposition/DGS as well as downloadable components can be found at http://www.webcomposition.net/dgs.

5 Related Work

The growing demand for creating, managing and publishing data within Web-based solutions is reflected in the high number of ongoing developments in this field. In this section we discuss approaches related to our work. We concentrate on solutions that follow the REST-style principles, allow building and hosting enterprise models and are applicable to our case study. The solutions are segmented into application-specific, database-driven and protocol-driven approaches.

Application-specific container solutions: This class of solutions includes Wikis, Weblogs and portals that maintain data in the form of lists. Wikis provide lists with extensive change-histories and references to list items that do not yet exist. Weblogs are also part of this class by providing chronologically ordered lists of entries and different views (e.g. by tags, month, week or day) on the data. While both follow REST-like architectures and gain increasing recognition within companies [Li07], they mainly focus unstructured content for human readers. An example of a commercial list-oriented application is Backpack [37Si08]. This easy-to-use Web-based service provides basic functionality to manage and share lists such as notes, images and files on the Web. The data model is however preset and not subject to growth. The more complex document management system Microsoft Office SharePoint [Mi07] provides the capability of maintaining lists of documents, tasks, contacts, appointments as well as self-defined lists. While this theoretically allows managing arbitrary models, the monolithic approach taken makes it hard to reuse the structure outside the portal itself in a standardized way. Generally, container solutions are suitable for dedicated tasks, offer a limited set of functionality and appear mostly as stand-alone solutions that are difficult to integrate into existing component-based environments.

Database-driven approaches: Currently, a number of services can be observed that provide traditional database functionality over Web standards. These services mostly target a commercial audience and are offered by particular providers. The Amazon SimpleDB [Am08] Web service includes a simple REST-style Web service interface that follows the traditional CRUD (Create, Read, Delete, Update) pattern [Ki90]. SimpleDB is designed to store relatively small amounts of data with the focus on fast data access. For growing needs, the commercial Amazon S3 service [Sh07] is offered. In contrast to WebComposition/DGS, these services form black-box components, lack the capability of customization and do not impose structure on the data. The Microsoft ADO.NET Data Services [Ca07] comprise an implemented set of patterns for data-centric services. The data is represented in various common formats such as XML, JSON or ATOM/RSS. All the characteristics of the REST-like architectural style appear as proposed in [Fi00]. Compared to the WebComposition/DGS component, this solution is designed for the Microsoft database management systems only and introduces dependencies to very specific platforms and technologies. None of the examined database-driven approaches supported creating and changing list structures via their Web interface. Their usage often requires extensive background knowledge of database management systems.

Protocol-driven approaches: The Atom standard [GD07, NS05] defines an XML-based format and an HTTP-based protocol for publishing and editing lists of related docu-

ments. For example, the POST method is used to create new resources within a collection, including an additional link entry for an extendable set of metadata about the particular resource. The most common usage of Atom is the syndication of news and Weblogs. While the format is extendable, there is no support for serving multiple representations of a resource. The Google Data APIs [Go08] provide simple protocols for reading and writing data on the Web based on RSS 2.0 and ATOM 1.0. The CRUD concept is fully supported through a REST-style interface using HTTP and the corresponding methods to access and query the XML-based data. Metadata is provided in the form of additional feeds containing referential information using e.g. the Google Base schema. However, this strategy is limited in terms of the fixed semantic information provided by the metadata feeds. According to their protocol-driven nature, the approaches do not cover component-based architectures for realizing extendable systems.

With respect to the aimed usage for bottom-up enterprise models, the related approaches were either too rigid for gradual growth, or too unstructured for providing the model data in multiple reusable representations.

6 Conclusion

In this paper we presented the WebComposition/DGS component model as a framework for the Web-compliant exposition of enterprise models. Our work focuses on the often observed bottom-up approach to creating valuable enterprise data sources that start out as simple lists on the Web. This process is supported by breaking down functionality and technology support into an extendable set of components. Within a case study, we showed the practical application of corresponding component implementations and demonstrated their ability to represent different representations of the model for maximum reusability. Upcoming work is targeted at finding ways to better support end user tasks, like e.g. the specification of new lists. Another interesting open problem is the transparent handling of URIs as references to other list entries or machine-readable information on the Web with corresponding user interfaces.

References

[37Si08] 37signals LLC: Backpack - Website, 2008, http://www.backpackit.com/ (02-19-2008).

[Am08] Amazon Web Services LLC: Amazon SimpleDB Developer Guide, 2008.

[An06] Anant, J.: Enterprise Information Mashups: Integrating Information, Simply, in 32nd International Conference on Very Large Data Bases, VLDB Endowment, Seoul, Korea; 2006, p. 3-4.

[BCH07] Bizer, C.; Cyganiak, R.; Heath, T.: How to Publish Linked Data on the Web, 2007.

[Ca07] Castro, P.: Project Astoria, The Architecture Journal, 2007(13): p. 12-17.

[Fi00] Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures, University of California, 2000.

[FG98] Fox, M.S.; Gruninger, M.: Enterprise Modeling, AI Magazine, 1998. 19(3): p. 109-121.

[Go08] Google Inc.: Google Data APIs - Website, 2008, http://code.google.com/apis/gdata/ (02-17-2008).

[GD07] Gregorio, J.; Hora, B.D.: The Atom Publishing Protocol - Request for Comments: 5023, 2007, http://www.ietf.org/rfc/rfc5023.txt (02-19-2008).
[HG08] Heil, A.; Gaedke, M.: WebComposition/DGS: Supporting Web2.0 Developments With Data Grids, in IEEE International Conference on Web Services (ICWS 2008), Beijing, China; 2008.
[In08] Internet2: Shibboleth - Website, 2008, http://shibboleth.internet2.edu/ (08-08-2008).
[Ki90] Kilov, H.: From semantic to Object-oriented Data Modeling, in First international Conference on Systems Integration, Morristown, NJ, USA; 1990, p. 385-393.
[Li07] Li, C.; Stromberg, C.: The ROI of Blogging, Forrester Research Inc., 2007.
[Ma08] Markl, V. et al.: Data Mashups for Situational Applications, in Model-Based Software and Data Integration: First International Workshop, Mbsdi 2008,, Berlin, Germany; 2008.
[MMG07] Meinecke, J.; Majer, F.; Gaedke, M.: Component-Based Content Linking Beyond the Application in Seventh International Conference on Web Engineering (ICWE), Springer, Como, Italy; 2007, p. 427-441.
[MNG05] Meinecke, J.; Nussbaumer, M.; Gaedke, M.: Building Blocks for Identity Federations, in Fifth International Conference for Web Engineering (ICWE2005), Springer, Sydney, Australia; 2005, p. 203-208.
[Mi07] Microsoft: Microsoft Office SharePoint Server 2007 Homepage - Website, 2007, http://office.microsoft.com/en-us/sharepointserver/ (02-01-2007).
[Mi08] Microsoft Corporation: Microsoft Popfly - Website, 2008, http://www.popfly.com/ (02-19-2008).
[MM04] Mukherjee, R.; Mao, J.: Enterprise Search: Tough Stuff, Queue, 2004. 2(2): p. 36-46.
[NS05] Nottingham, M.; Sayre, R.: The Atom Syndication Format - Request for Comments: 4287, 2005, http://www.ietf.org/rfc/rfc4287.txt (02-18-2008).
[PZL08] Pautasso, C.; Zimmermann, O.; Leymann, F.: RESTful Web Services vs. “Big” Web Services: Making the Right Architectural Decision, in 17th International World Wide Web Conference, ACM Press, Beijing, China; 2008, p. 805-814.
[Pi05] Pierson, N.: Overview of Active Directory Federation Services in Windows Server 2003 R2, 2005.
[Sh07] Shanahan, F.: Amazon.com MashupsEds., Birmingham, UK: Wrox Press Ltd., 2007.
[TW06] Tapscott, D.; Williams, A.D.: Wikinomics: How Mass Collaboration Changes Everything, New York, NY: B&T, 2006.
[Us08] U.S. Department of Labor: Bureau of Labor Statistics - Website, 2008, http://www.bls.gov/ (07-20-2008).
[Ya07] Yahoo! Inc.: Pipes: Rewire the web - Website, 2007, http://pipes.yahoo.com/pipes/ (02-19-2008).
[Yo07] Young, G.O.: IT Will Measure Web 2.0 Tools Like Any Other App, Forrester Research, Inc., Cambridge, MA, USA, 2007.