WEB 2.0 MASHUPS FOR CONTEXTUALIZATION, FLEXIBILITY, PRAGMATISM AND ROBUSTNESS

Thorsten Hampel
Department of Knowledge and Business Engineering, University of Vienna, Rathausstr. 19/9, A-1010 Vienna, Austria

Tomáš Pitner
Masaryk University, Faculty of Informatics, Botanická 68a, 602 00 Brno, Czech Republic

Jonas Schulte
Heinz Nixdorf Institute, University of Paderborn, Fuerstenallee 11, 33102 Paderborn, Germany

Keywords: Mashup, Mistel, Web 2.0, Web Service, Flexible Integration.

Abstract: In current Web 2.0 developments the word *"mashups"* stands for a totally new way of flexible application bringing together several Web 2.0 services. Mashups form a serious new trend in web development by contextualizing existing services and bringing customizable applications to the user. Out of this, a new style of web-based applications arises. To get a better understanding of the phenomena of mashups, this paper observes and interprets the way the development of web based applications and appropriate background systems changed. It presents several state of the art Web 2.0 technologies and points out why the evaluation of Web 2.0 applications currently moves from web-based APIs to web services. We will present a short taxonomy of mashups as flexible, pragmatic, and robust web applications. Further, their new form of flexibility and customization will be illustrated with the help of our mistel Framework.

1 INTRODUCTION

A first look at the so-called Web 2.0 movements unveils a large number of different understandings and definitions. People often think of social software and community developments such as *"power to the people"*. Others mainly confer informal semantics such as tags or folksonomies. An extensive overview of folksonomies, their characteristics, and semantics in communities is given in (Michlmayr, 2005).

"Mashups" a new buzzword within the Web 2.0 context originally is used in the scope of music and means remix; in the context of Web 2.0 mashup means reusing and conflating existing technologies to develop a new style of applications. Anymore mashup also stands for the requirement changes of the Web 2.0, which needs faster reusability and integrability than ever before.

As far as a new understanding for the new web and therefore for the next generation of the Internet can be, so important (we think) it is to be aware of the new web not to be a collection of single services, but a network (mashup) of highly interoperable services. No longer users are served only by a single service, they can profit from a network of services. Shaped transparently for the user, the application consists of different network services, which are contextualized and integrated dynamically to the users needs.

Therefore our understanding for the next generation web (we still call that for a better understanding Web 2.0) is about:

- contextualization of services along the users needs,
- flexible interoperability of services, and
- robust and secure service integration.

Most of the above-mentioned issues are attributed to *Service Oriented Architectures (SOA)* (Bih, 2006). There is a plethora of definitions of the *service* concept in IT, ranging from very abstract ones, such as "a functional unit" (IBM), to more concrete ones "self-contained stateless business function that ac-

Hampel T., Pitner T. and Schulte J. (2008).
WEB 2.0 MASHUPS FOR CONTEXTUALIZATION, FLEXIBILITY, PRAGMATISM AND ROBUSTNESS.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - SAIC*, pages 107-112
DOI: 10.5220/0001693201070112
Copyright © SciTePress

cepts one or more requests and returns one or more responses through a well defined standard interface" (Wikipedia).

The discussion of web services (Gurguis and Zeid, 2005; Schranz, 1998) stands now for more than five years for the same idea: offering flexible and self-describable web based services. Since then is the concept of services applied in many areas including basic system infrastructure, such as storage service, network monitoring, user identity management and authentication, and geoinformation services.

Popular Web 2.0 services grow very quickly, therefore a robust, scalable architecture, and, at the same time, a simple, easy-to-use user interface is needed. Thus, architectural styles and patterns supporting non-functional requirements, such as the *Representational State Transfer (REST)* are highly welcome.

In this paper we explore the leading Web 2.0 technologies to point out why the evolution moves from web-based APIs to web services and why mashups offer a new technology for developing flexible, pragmatic, and robust web applications. We compare different models of integration to emphasize functionalities and technologies that refine the concept of mashups. We will concentrate primarily on the technological aspects while business and legal issues are discussed in (Drášil et al., 2008).

The paper is structured as follows: Primarily, we identify why are the Web 2.0 services different from the traditional view of Web services in the Section 2. The technological foundation of service and system integration is outlined in the Section 3 providing a comparison between static web-based APIs, SOAP- and REST-based web services by in addition pointing out the benefits of using web services. The next Section 4 focuses on development of services and integrated systems including mashups. After presenting the state-of-the-art in open system integration (4.1), the shortcomings of current solutions and techniques are highlighted in 4.2. Since Web 2.0 mashups are an enhancement of traditional web services, 4.3 presents the requirements and visions for creating mashups. As the ultimate goal, self-descriptive services are described and accompanied by a proposed implementation in 4.4. The paper is concluded in Section 5.

2 WHAT IS NEW TO WEB 2.0 SERVICES?

Once might ask what is really new to the Web 2.0 movements regarding the well-known web service idea? New is not the basic idea nor the used technology (e.g. SOAP and XML); new are its attributes of:

- a broad and successful usage of different web services as part of any class of Web 2.0 applications (e.g. Google maps is widely used and therefore generates complete new classes of applications),
- a pragmatic integration of web services on different levels of semantics and complexity (e.g. web services can be on a high semantic level, such as semantic web services (Narayanan and Mcilraith, 2002; Patil et al., 2004) or on a very low semantic level, such as screen scraping), and
- a new quality in the integration of web services. Services are integrated in a way that they offer a new quality of interoperability and interactivity to the users.

Results of these new attributes is real system convergence. Different parts of applications serve as a flexible, highly customizable, and contextualizable applications to the users. As a result the Web 2.0 movements significantly increase the stability and usability for many well-known technologies. This leads to a larger number of successful new applications and robust technologies. The following contribution will discuss the above-mentioned criteria for web services in the focus of contextualization and system convergence. Different stages of service integration will be presented and analyzed, from the high-level web service integration up to simple forms of application integration. As part of different levels of service complexity, also different ways of using web services will be discussed in detail. As a special focus of the paper various security aspects of web services will be presented. Our goal is to shape a new understanding of mashups as the accelerators for the Web 2.0 movement.

3 FROM APIs TO WEB SERVICES - THE TECHNOLOGY BEHIND

Web-based systems are in general characterized by *orientation to network*. However, the architectural and interaction models, protocols, and data exchange formats vary significantly among different types of web-based systems.

3.1 Web-based API

Networks of web-based systems can be build upon a tight coupling via a *Web-based API*, enabling to integrate the functionality of a remote service or system providing an interface specific for the given programming language and the service integrated. For a typical contemporary programming language, such an API has a form of object-oriented library encapsulation the remote service functionality. An integration is thus technically easy but inflexible, being closely tight to the respective programming environment; actually not gaining any advantage of the service-orientation. Despite of these disadvantages, many successful Web 2.0 services (like Delicious, Flickr, and others) offer such an API.

3.2 SOAP-based Web Services

To provide a more flexible, platform and language neutral interfacing, current Web Services provide a description of the service using the *Web Services Description Language* (WSDL) that can be used to automatically generate clients interacting with the web service. This allows an easy modification of existing applications if the interface of a web service changes.

In Web services, SOAP (Consortium, 2008) serves as a protocol for messaging between a Web service and its clients. Being based on XML, it represents a platform neutral foundation for wide system interoperability with looser coupling than a web-based API. Nowadays, SOAP-based Web services are, despite of their complexity, the industry standard for implementing a service-oriented architecture in an enterprise environment. They also quickly replaced older interoperability efforts like CORBA, based on distributed objects rather than on services.

However, for integration services on Web 2.0, they might not always be the right option, as they do not warranty scalability in a *whole web dimension*. An alternative usable for web scale services requires even simpler use, less overhead, and better scalability than SOAP-based Web Services. *REST architectural style* may be the right concept for such Web Services.

3.3 REST Web Services

REST (REpresentational State Transfer) was coined by R. T. Fielding (Fielding, 2000). Fielding's goal was to specify a set of *architectural constraints* (also called *architectural style*) ensuring better performance and scalability within distributed hypermedia systems fully based on successful HTTP protocol.

1. It uses the concept of *resource* instead of *service*. A user, group, calendar item, map, or a document are examples of a resource in the REST sense.
2. Resource are *identified* by unique URIs (typically URLs), and may have one or more *Representations* that can be textual (like plaintext), (semi)structured data (XML), or multimedia (like images), among others.
3. HTTP protocol methods (like POST, GET, PUT, and DELETE) correspond to basic operations on resources.

Notably, the state of the processing flow ("session") between the service provider and consumer is not kept at the server – the communication is *stateless*. Therefore, it supports robustness and scalability because the communication between server and client consists only of exchanging representations of resources and thus may be proxied and/or cached.

Vinoski (Vinoski, 2007) notes namely that REST architectural style with its uniform HTTP interface as an important constraint significantly simplifies development of large service distributed applications and leads to more scalable and robust ones. While SOAP service definition language (WSDL) inherently binds data formats together with interface constraints, in REST the data formats are necessarily *orthogonal to interfaces*. REST thus introduces self-describing messages based on standard formats (typically but not limited to open web formats) negotiated between client and server.

The REST architectural style simplicity and ease-of-use led to large popularity of REST among Web 2.0 service developers. Many of the present Web 2.0 services (ProgrammableWeb.com, 2008) prefer a REST-based API instead of a SOAP-based one. Some services employing both SOAP- and REST-based interface (like Amazon(Goth, 2004)) show that simpler REST-based service gaining about 80 % of the total traffic clearly outperform their SOAP counterparts.

4 DEVELOPING WEB SERVICES AND MASHUPS

This section focusses on generating networks of web services, so called mashup applications. After explaining the shortcomings of today's web services in Section 4.2, Section 4.3 presents the definition of a mashup and the key advantages. In order to automatically build networks of services, the self-descriptiveness of web services is required as described in Section 4.4.

4.1 Web Service Integration

In the past four years several frameworks and concepts for an "Open Service Integration" for *Computer Supported Cooperative Work* (CSCW) applications appeared. The range of these concepts and frameworks spreads from simple ones, requiring manual user interaction to select services from an UDDI inventory to those, that try to find and dock with newly available services fully automatically.

The authors of (Lima-Gomes et al., 2005a; Lima-Gomes et al., 2005b) offer a *Loosely-coupled Environment for Integrating Collaborative Applications* called LEICA. This framework consists of three different components: session management, event-notification service, and adapter for each used collaborative application. The session management exchanges information with the adapters to configure each session. After exchanging all necessary data, the systems communicate using events, which are distributed through the event notification service. As a result, in LEICA each application is still independent and no data is exchanged directly between systems.

In (Anzures-García et al., 2006) the authors propose a SOA-based architecture for collaborative applications. Their architecture includes a component for managing sessions and for group awareness. Due to its service-oriented structure, it is possible to include new services as web services. The disadvantage of this architecture is that existing applications cannot be used offhand as part of this service oriented architecture. Instead, Computer Supported Cooperative Work (CSCW)-Services need to be explicitly developed for this architecture in order to use the provided services.

When applications and services rely on services that are subscribed dynamically and on-demand many other aspects, e.g. monitoring of Service Level Agreements (SLA) for web services have to be considered. In (Keller and Ludwig, 2003) the authors present the Web Service Level Agreement (WSLA) framework, which is targeted at defining and monitoring SLAs for web services. The WSLA framework is part of IBM's web service toolkit. Obviously such developments are so successful, since the number of available web services is increasing rapidly and thus automatic finding, monitoring and coupling of web services becomes essential.

4.2 Current Shortcomings and Limitations

The searchability of web services is a significant benefit in comparison to web-based APIs since potential consumers can find unknown services and providers. However, the concept of the UDDI (Clement et al., 2002) directory is not completely practicable, because the supported search bases on WSDL files which includes plain text comparison and lacks of semantics. A search returns all web services whose description contains the searched keywords. Obviously, a description can be on the one hand extensive and not only point out the key functionality of the associated web service and on the other hand same terms can be used in different contexts. As an example, searching for the keyword "weather" finds 3% of all registered web services at a popular web service repository like *Salcentral.com* (Patil et al., 2004). Since the search has to build on WSDL files and thereby to base on text-search, the used terms have to be set in relation to the semantic context.

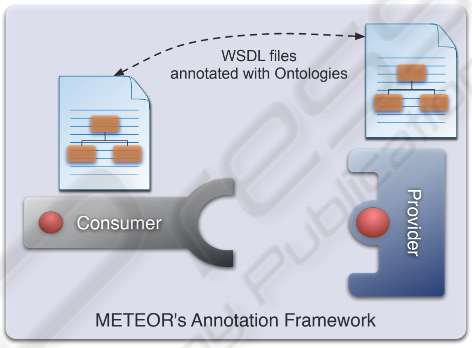

Figure 1: WSDL Files Annotated with Ontologies.

Patil et al. have extended WSDL with semantic annotations (Patil et al., 2004) in the METEOR-S Framework to handle the missed out descriptiveness of web services (see Figure 1). Setting the description in a semantic context by using ontologies improves the search quality and reduces the number of not suitable search hits. The description of a web service applies to its input / output (*data semantics*) as well as its functionality or business workflow processed by the web service (*functional semantics*). A search which can differ between data semantics and functional semantics is desirable to improve the search quality by a more precise selection of eligible services. Furthermore, in the context of the METEOR-S Framework the *execution semantics* as well as the *QoS semantics* have been identified which describe the correctness and verification of the execution as well as the performance and cost parameters associated with the service. Since ontologies will increase over time, it was necessary to develop a semi-automated process of annotating web services with ontologies. The METEOR-S Framework addresses the requested semi-automatism and scalability by implementing adequate algorithms for semantic annotation and categorization of web services.

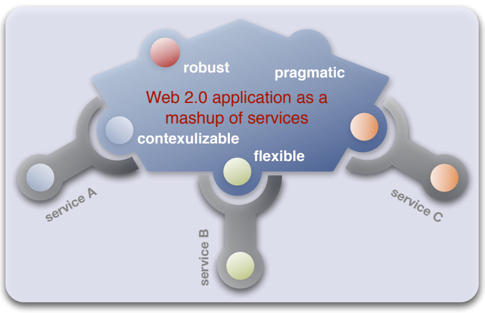

Figure 2: Web 2.0 Application as a Mashup of Services.

4.3 Integration in Web 2.0

A Web 2.0 application being a mashup unions different web services in order to provide more functionality for a client interacting with this Web 2.0 application. As an example, the mashup application capsulates the interaction with a service A for searching in a digital library with a service B for storing or linking documents in a CSCW system. Hence, the mashup application offers to clients the whole functionality to search for digital literature as well as storing documents in a CSCW system by interacting only with one application. The interaction with one single application has a big benefit for the service consumer, since searching for literature and making these documents available for e.g. other students should be one step without a media breach.
Figure 2 depicts our definition of mashups. The key idea of developing Web 2.0 applications as mashups of services is that they are pragmatic and contextulizable by combining different web services in order to higher the value of using the services. In particular, often the value of using the combined services is significantly higher than using services separately. Since the Web 2.0 application can add, remove, or exchange arbitrary services of their mashup, they provide highest flexibility for services. Web 2.0 applications can hold themselves a set of useable web services providing the same functionality. If integrated services are not running or not available, the Web 2.0 application can use other services proceeding the same business workflow. This interchangeability of web services increases the robustness of services being mashups.

Note that the semantic annotation of web services presented in Section 4.2 enables the contextualization and generation of meaningful and pragmatic composition of several web services. However, to completely implement the composition, more information (for example security aspects) than provided by the WSDL file is required. Section 4.4 addresses this issue.

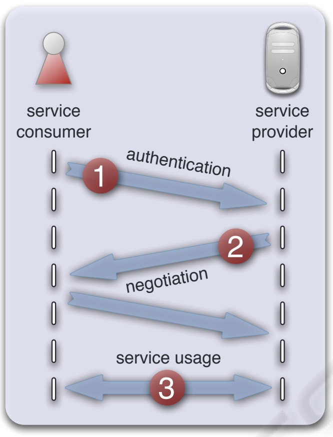

Figure 3: Negotiation Process Between Service Provider and Service Consumer.

4.4 The Ultimate Ambition: Self-descriptive Web Services

As mentioned above, our understanding of mashups does not only means to reuse existing web services but also remixing services of different applications. The following steps describe a communication cycle between service provider and service consumer in mistel:

1. The first step contains security aspects like authentication and authorization, since not every service consumer is allowed to use all available services of any service provider.
2. In the second step both actors, the service provider and the service consumer, negotiate about the service usage. This means, that the service provider describes the web service functionalities.
3. After successful service negotiation the consumer uses the service by invoking its methods.

Figure 3 depicts the communication process in three phases. The difference between a normal web service invocation and the mistel web services is the negotiation process in step two with its enhanced self-explanation capabilities. In fact a service provider may also act as a service consumer and vice versa.

We have chosen a similar approach with defining a common API for collaborative systems called the CowAPI. The goal is a to allow a loose coupling of collaborative systems into an application. Because of the varying functionality of different groupware systems we defined some functional areas:

- Users: Create, Query, and Manipulate Users
- Groups: Create, Query, and Manipulate Groups, Group Membership Files: Create Files and Containers, get inventory of containers

- Calendars: Create and manage appointments

A system implementing the CowAPI could implement all functional areas or a subset. The explain functionality allows to return the supported functionality. Because of the standardized API it is now possible to exchange systems without needing to rewrite any code. The CowAPI could be used as a mediator service to connect different backend groupware servers or could be made directly available within any groupware system.

5 CONCLUSIONS

The usage of web services providing access to already developed applications had increased the number of application fields in comparison to static web-based APIs. However, a single web service often does not offer the complete functionality requested by consumers. As a consequence, networks of services offer a higher value for consumers since they gain more functionality or can use more complex business workflows by interacting with one single Web 2.0 application being a mashup of various services. Mashups feature pragmatism, robustness, and contextualization since they can add, remove, or exchange specific web services of their network.

To enable an automatic generation of mashups, on the one hand semantic annotations are desirable in order to identify those services whose composition offers a higher value for consumers. On the other hand web services must be able to describe themselves in detail. Hence, a negotiation phase will be performed before a service can be added to the mashup.

ACKNOWLEDGEMENTS

The research has been supported by the German Research Foundation (DFG), project *MISTEL*. For further information please refer to `www.dfg.de` and `www.systemkonvergenz.de`. Moreover the research has been partially supported by the Czech National Program Information Society, project *E-learning in the Semantic Web Context*, No. 1ET208050401.

REFERENCES

Anzures-García, M., Paderewski-Rodríguezguez, P., and Hornos, M. J. (2006). Soa-based generic architecture for cscw systems. In *Proceedings of First International Conference on Ubiquitous Computing (ICUC), Alcalá de Henares, Spain.*

Bih, J. (2006). Service oriented architecture (soa) a new paradigm to implement dynamic e-business solutions. *Ubiquity*, 7(30):1–1.

Clement, L., Hately, A., von Riegen, C., and Rogers, T. (2002). Uddi version 3.0 published specification.

Consortium, W. (2008). Soap version 1.2 – w3c recommendation 27 april 2007.

Drášil, P., Hampel, T., Pitner, T., and Steinbring, M. (2008). Get ready for mashability! concepts for web 2.0 service integration. In *ICEIS 2008 Proceedings, Barcelona, Spain*. INSTICC.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine. Chair-Richard N. Taylor.

Goth, G. (2004). Critics say web services need a rest. *IEEE Distributed Systems Online*, 5(12).

Gurguis, S. A. and Zeid, A. (2005). Towards autonomic web services: achieving self-healing using web services. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5.

Keller, A. and Ludwig, H. (2003). The wsla framework: Specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manage.*, 11(1):57–81.

Lima-Gomes, R., de Jesús Hoyos-Rivera, G., and Courtiat, J.-P. (2005a). Leica: Loosely-coupled environment for integrating collaborative applications. In *DEXA Workshops*, pages 635–639. IEEE Computer Society.

Lima-Gomes, R., de Jesús Hoyos-Rivera, G., and Courtiat, J.-P. (2005b). Loosely-coupled integration of cscw systems. In Kutvonen, L. and Alonistioti, N., editors, *DAIS*, volume 3543 of *Lecture Notes in Computer Science*, pages 38–49. Springer.

Michlmayr, E. (2005). A case study on emergent semantics in communities. In *Proceedings of the Workshop on Social Network Analysis, International Semantic Web Conference (ISWC)*.

Narayanan, S. and Mcilraith, S. (2002). Simulation, verification and automated composition of web services. In *Proceedings of the 11th International Conference on World Wide Web*, pages 77–88, New York, NY, USA. ACM Press.

Patil, A., Oundhakar, S., Sheth, A., and Verma, K. (2004). Meteor-s web service annotation framework. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW2004)*, New York, USA. LSDIS Lab, University of Georgia.

ProgrammableWeb.com (2008). Programmableweb – mashups, apis, and the web as platform.

Schranz, M. W. (1998). Engineering flexible world wide web services. In *SAC '98: Proceedings of the 1998 ACM symposium on Applied Computing*, pages 712–718, New York, NY, USA. ACM.

Vinoski, S. (2007). Rest eye for the soa guy. *IEEE Internet Computing*, 11(1):82–84.