Proceedings of the

First International Workshop on Lightweight Integration on the Web (ComposableWeb'09)

Florian Daniel[1], Sven Casteleyn[2], Geert-Jan Houben[3]

[1]University of Trento, Italy
daniel@disi.unitn.it

[2] Vrije Universiteit Brussel, Belgium
sven.casteleyn@vub.ac.be

[3] TU Delft, Netherlands
g.j.p.m.houben@tudelft.nl

Preface

While the word mashup is widely used today, to some of us it is still not really clear what a mashup is and what it is not. Some mashups focus on integrating RSS feeds, others on integrating RESTful services, SOAP services, Atom feeds, or user interfaces. Yet, everybody recognizes that mashups represent a new way of expressing innovation, sometimes even *user innovation*, i.e., innovation in the form of simple web applications "implemented" or "mashed up" by web users.

Typically, implementing a mashup means *integrating* resources available on the Web into a new, value-adding application. The integration may occur at the user interface level (most mashups do integrate presentation content, not just data), at the application logic level (web service are one of the cornerstones of mashups), or at the data level (RSS/Atoms feeds or XML files are common practices today), or at a combination of them. Therefore, we say a *mashup* is a web application that is developed by composing data, application logic, and/or user interfaces originating from disparate sources available on the Web.

In general, developing mashups is a hard and tricky task. For instance, a mashup composer can of course use any conventional *programming language* to integrate, i.e., mash up, the resources and components of his choice. For instance, among the most used languages today we find PHP on the server side and JavaScript on the client side; many other languages are used as well. Given the heterogeneity of technologies, programming languages, interaction protocols, the complexity of the necessary integration logic, and similar, *manual development* of mashups is only an option for highly skilled programmers. And even they could experience a hard time in mastering all the development challenges. Service composition approaches "a la BPEL" are not able to cope with the above mentioned heterogeneity of technologies.

With the advent of *mashup tools*, the mashup phenomenon has however become more popular even among web users, as they offer users the unique chance of getting involved into the development process of web applications in an intuitive and *assisted* fashion. Usually, mashup tools aim at simplicity more than at completeness of features, and they support fairly sophisticated development tasks in the browser.

Despite the current emphasis on mashups and lightweight integration on the Web, we still register a lack of agreed-upon reference models (e.g., for component and composition languages), development processes and methodologies, architectures, execution platforms, analysis techniques, and so on, that effectively aid mashup development. In particular, the involvement of web users into the development of composite online applications demands intelligent concepts and a high degree of assistance – especially to the most inexperienced users. It even results in new (social) development practices, which in turn may require new software support.

In this context, several challenging research issues are emerging, among which the following seem of particular interest:

1. *Reusable components*: Expressive component models for data, application logic, and user interface components, as well as suitable description languages, and discovery and selection facilities (e.g., registries and protocols) are needed.

2. Simple, *lightweight composition languages*: Easy-to-learn yet expressive execution languages are required, which enable the plug-in style development of composite applications.
3. *Graphical composition tools*: Composition languages should be equipped with suitable graphical modeling formalisms that are able to hide the actual composition complexity and allow for computer-aided development environments.
4. *Suitable execution platforms*: Ready compositions require proper execution support (e.g., an interpreter or parser). We expect such support to be provided through online hosting and execution platforms.
5. Design aimed at *interoperability*: Components and mashups should be interoperable, meaning that they have cross-platform reusability. Mashup-specific standards might be necessary.

In light of these considerations, the goal of ComposableWeb is to challenge the Web Engineering community with these new research issues and to stimulate the discussion of key issues, approaches, open problems, innovative applications, and trends in these and related research areas, so as to identify technologies, solutions, instruments and methodologies that effectively support the lightweight composition of web applications.

This volume collects the proceedings of the first edition of ComposableWeb, held in conjunction with the International Conference on Web Engineering (ICWE) on June 23, 2009, in San Sebastian, Spain. Out of all submissions, eight papers (six full papers, two position papers) attained sufficient quality and were selected for presentation during the workshop; unfortunately, other submissions could not be accepted.

We would like to thank all authors, of both accepted and rejected papers, for their contributions, the presenters and the people participating in the workshop for their engagement and contributions to the discussions during the workshop, and the chairs of ICWE'09 for their support in the organization of the event.

Trento, Brussels, Delft
June 2009

Florian Daniel
Sven Casteleyn
Geert-Jan Houben

Organization

Organizing Committee

- Florian Daniel, University of Trento, Italy
- Sven Casteleyn, Vrije Universiteit Brussel, Belgium
- Geert-Jan Houben, TU Delft, the Netherlands

Steering Committee

- Sven Casteleyn, Vrije Universiteit Brussel, Belgium
- Florian Daniel, University of Trento, Italy
- Maristella Matera, Politecnico di Milano, Italy
- Geert-Jan Houben, TU Delft, the Netherlands
- Olga De Troyer, Vrije Universiteit Brussel, Belgium

Programme Committee

- Sören Auer, University of Leipzig, Germany
- Boualem Benatallah, University of New South Wales, Australia
- Fabio Casati, University of Trento, Italy
- Peter Dolog, Aalborg University, Denmark
- Marlon Dumas, Tartu Univesity, Estonia
- Schahram Dustdar, Technical University of Vienna, Austria
- Rama Gurram, SAP Labs Palo Alto, USA
- Frank Leymann, University Stuttgart, Germany
- Michael Mrissa, University of Lyon, France
- Moira C. Norrie, ETH Zurich, Switzerland
- Cesare Pautasso, University of Lugano, Switzerland
- Gustavo Rossi, Universidad Nacional de La Plata, Argentina
- Takehiro Tokuda, Tokyo Institute of Technology, Japan

Table of Contents

Data Extraction from Semantic Annotated Deep Web Sites

Eduardo Martín Rojo and Vicente Luque Centeno

Universidad Carlos III de Madrid
Av. Universidad 30, 28911
Leganés (Madrid), España
{emartin,vlc}@it.uc3m.es

Abstract. Automatic navigating and gathering information from Deep Web sites requires the use of Web Wrappers in order to simulate human interaction with Web sites. Web Wrappers have some drawbacks: their implementations are specific to the accessed site and also their source code needs a constant maintenance in order to support new changes on Web site.
In this work we propose an annotation model for Deep Web sites that could be used for data extraction from the point of view of a Web client. Using these annotations will enable Web Wrappers to be more adaptable to Web site changes.

1 Introduction

Nowadays most popular Web sites provide to their users with development tools that make possible to create applications that access their services, like *eBay Developers Program*[1]. Also, sometimes they provide Web services that could be accessed by *Mashup* applications like *Google Mashups*[2], *Yahoo Pipes*[3] or *Microsoft Popfly*[4]. However, the great majority of Web sites do not provide development tools or Web services because they are only focused on being operated by human users. Automatic accessing Web sites that do not provide these facilities is achieved by using *Web Wrappers*[7].

Web Wrappers have some drawbacks: first, they require developers with strong knowledge on the accessed Web site because Web Wrappers are site specific solutions that have dependencies with Web structure; and second, Web Wrappers require constant maintenance in order to support new changes on the Web sites they are accessing. Web Wrappers development tools evolved trying to solve these drawbacks and also to make possible an easier integration of Web data from heterogeneous sources. Some examples of common Web Wrappers development tools are site specific solutions like *GreaseMonkey*[5] and *Ubiquity*[6],

[1] http://developer.ebay.com/
[2] http://www.googlemashups.com/
[3] http://pipes.yahoo.com
[4] http://www.popfly.com
[5] http://www.greasespot.net/
[6] http://ubiquity.mozilla.com/

user-friendly tools that allow natural language references like *Chickenfoot*[7], and also graphical IDE solutions like *OpenKapow RoboMaker*[8].

The main common characteristic among all current Wrapper development tools is that, although their operation is simple, they still have strong dependencies with Web site structure, originating a continuous effort of maintenance with the created Wrappers. In our work we will define a formalization of Web structure that can be used for semantic annotation of Deep Web sites in order to represent their navigation operation. These annotations combined with current Wrapper development tools will allow implementation of Wrappers focused on the Web site model, avoiding overlapping with site structure. The maintenance required by new Web site changes will be isolated inside this model layer. Any future modifications on the structure of the Web site will need only to change the semantic annotation for the Web site, and all the Wrappers that make use of these annotations will not need to be changed.

One of the main problems raised with the navigation model generation is the election of a point of view. In [11], it is indicated that a Web navigation model can be generated from three points of view: from server point of view, analyzing physical structure of Web site; from client point of view, analyzing client interaction with Web site; and from a hybrid point of view as a combination of client and server point of view. In our work we have chosen to follow the client point of view because of the following reasons:

- The navigation model must support the changeableness and diversity of Web content. Client side technologies like AJAX or Flash are widespread. The model must support also this kind of content.
- It is non intrusive; it does not require to change the Web site being annotated as other semantic annotation methods like RDFa[12] and Microformats[10].
- Client-only point of view allow third party and end users to participate and collaborate in the creation of annotations about any Web site because they will not have access to the server.

In this document, section 2 will introduce previous works related with Deep Web navigation modeling and data extraction. The annotation model for expressing navigation graphs that we present is described in section 3 by defining its main classes and properties. Also we present an example of annotation for a shop Web site that uses our model. Section 4, describes how can be used the annotations for the extraction of information by using an example that performs a query over different sources; and finally, future works and conclussions are described in section 5.

2 Related Works

The task of generating a navigation model from a Web Site has been previously faced, but previous works do not provide a Web model annotations formalization.

[7] http://groups.csail.mit.edu/uid/chickenfoot/
[8] http://openkapow.com/

There were attempts to automatically extract the model using real navigation examples like [1], where the authors make use of navigation examples that were extracted by recording and replaying the actions performed by a user and [11], focused on generation of models for visualization of Web sites hierarchy.

Deep Web navigation model generation is treated by [13], where the authors generate a navigation model where they use keyword-matching for identifying different Deep Web pages. Other work more focused on interacting with Deep Web sites is *Transcendence*[2], a system that allows users to generalize their queries for expanding their search scope. It allows also to define data extraction patterns for obtaining output data that may be combined with other data sources.

One of the main problems of Deep Web remains in that a Deep Web page cannot be always represented uniquely with an URL[9]. The problem of referencing Deep Web pages is addressed in [8], where the authors present a way of creating bookmarks of a Deep Web page using the sequence of steps specified with *Chickenfoot* scripts that simulate the user interaction in order to reach the bookmarked page. This is combined with images that show the visual representation of the bookmarked page.

Extracting semantic data from Web sites is a problem that have been faced in *Marmite*[16], a system that provides an end-user interface that allows him to construct the workflow needed for extracting the data, or *PiggyBank*[9], a system where the user can make use of scripts called *Screen Scrapers* for converting HTML to semantic data in RDF. Related with this last work is Sifter, a tool that

In our demo system we use *Chickenfoot* as Wrapper development tool. The main characteristics of this tool and its natural language references to HTML elements are presented on [3] and [4]. *Chickenfoot* enable the specification of client-side Web interactions with a simple Javascript based language.

3 Deep Web Annotation Model

The objective of a Web client is to reach a specific state through interacting with the Web. Our model represent the states and transitions that composes the navigation as a graph that describes all possible situations that could occur in a Web site from client's point of view. In the graph, vertexes represents all possible logical states that require an action produced by the user, while edges represent the actions produced that allow the transitions between logical states.

Every state could be divided in elements called fragments, and also these fragments could be divided into new fragments. Every fragment represents a type of semantic content that can be extracted by selecting part of the element that it belongs. Figure 1 shows an example of state division in different fragments. Fragments of logical states will allow us to extract semantic information from Deep Web pages if we know the location of the state inside the navigation graph of the Web site.

[9] Uniform Resource Locator, defined on http://tools.ietf.org/html/rfc1738

Fig. 1. States and Fragments

A transition represents the actions available for the user. These actions will allow changing among different states in the Web site. Transitions are composed of, first, an ordered sequence of interactions that originates the change of state, and second, a list of all possibe destinations that could be feasible by using the transition. There could be different possible destinations because the Web site could act in different ways as a consequence of dynamic factors like the state of the service framework, the interactions historical, date and time, etc.

Our model of states and transitions for the navigation graph is represented by the following types of elements:

1. PageState: a uniquely identifying state that represents a Deep Web page. A state contains fragments.
2. Fragment: represents an element inside a PageState or another Fragment. It is domain of the following properties:
   (a) fragmentOf: Defines this fragment as part of another fragment or part of a PageState as a hierarchy.
   (b) locatedBy: a XPath expression that identifies the position of the fragment inside the XHTML representation of the PageState.
   (c) numResults: an integer that defines how many times this fragment appears inside his father Fragment or PageState in hierarchy.

   (d) optional: a boolean that indicates if the fragment appears optionally inside his father Fragment or PageState. It may happen when a page has variable elements.
   (e) semantic: a set of RDF triples that represents the knowledge refered by this fragment. This property is also defined in Input and Action.
3. transitions: represents all possible transitions to other states that could be followed from this state. It has the following properties:
   (a) actions: any transition is originated by an ordered sequence of actions (instances of Action class)
   (b) sources and destinations: all possible states that could be source or destination of this transition respectively.
   (c) precondition and postcondition: preconditions and postconditions required in order to use this transition for travelling from sources to destinations. Postconditions can be used for distinguishing between different possible destinations.
4. Action: an interaction that could be performed over a fragment. It contains the following properties:
   (a) hasType: type of interaction (clicking element, selecting element, entering text...). It is represented by a *Chickenfoot* command in our demo annotations.
   (b) inputs: all the form values needed for performing the interaction are refered in this property as a list that contains elements of type Input. For each input it must be provided a name and a description.

We have represented this formalization in an OWL[10] ontology that can be accessed in [15].

In figure 2 we have represented an example of the annotated navigation model for searching products inside a typical Shop of Books site. From initial page at this site represented by node state0, there are two fragments: element0 (a searching textbox) and element1 (a button that initiates the searching task). From state0 it is possible to go to state1 through transition0. This transition requires to perform two actions (represented by the orderedActions list of actions): the first action is performed by inserting a search string inside element0 textbox and the second action is performed by clicking element1 button. At state1, there are three fragments that can be accessed: element2, element3 and element4. Because each of them is a part of other fragment, the XPath expression referred by the locatedBy property is relative to its father's locatedBy property. For example, element4 is constructed with element3 and element2 locations, and so its location should be //div[@id='atfResults'][$INDEX]//div[@class='productPrice'].

In this figure we have used an hypothetical ontology for defining concepts of Shops, but the annotation model presented can be combined with any ontology for defining semantic knowledge inside the Fragments, Actions or Inputs of the model. This knowledge can be provided as RDF data inside the property semantic of these elements.

[10] http://www.w3.org/TR/owl-features/

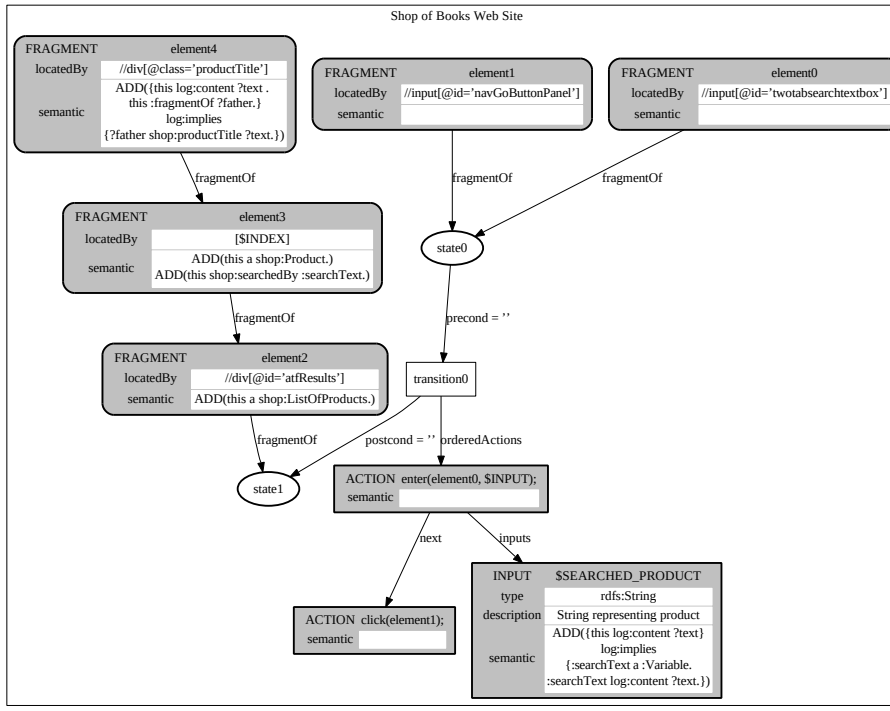


Fig. 2. Annotated navigation model for searching at Shop of Books site

Representing the semantic of a specific element inside a state for a Deep Web site requires not only the knowledge inferred from the specific state, but also the knowledge of all the transitions that have been followed for reaching the state. This knowledge may be transformed during the transitions, and new knowledge may be added or removed from the client's working memory of assertions, depending on the client interactions with the Deep Web site, as in a Production System or Rule-Based System [5].

As can be seen in figure 2, Fragment, Input and Action have defined the semantic property that indicates which knowledge (represented as RDF triples or inference rules) is added (ADD) or deleted (DEL) from the working memory. Adding a RDF triple simply adds itself to the working memory, while adding a RDF rule executes it inside working memory and every time new RDF triples are added, the rule must be reapplied. Deleting a triple or a rule eliminates its effect from the working memory. The effect of adding or removing RDF data of client's working memory follows these rules:

1. If the client is located at a State that contains fragments, the semantic property of the fragments is processed inside working memory from outer fragments to inner fragments following the fragmentOf property. Fragments at the same level of indexation can be processed in any order.
2. If the client has travelled through a Transition that contains a ordered list of actions, the semantic property of the actions is processed following the same order of the ordered actions list.
3. If the client uses an input defined in a fragment of a State or in an action of a Transition, the semantic property of the input is processed after all the other semantic properties of the State or Transition.

Semantic information is associated with every instance of Fragment, Action or Input by using the semantic property. With this property, the user that is annotating a Web site indicates which kind of information does the fragment, action or input represents. Its content is expressed in RDF.

For defining semantic information, in figure 2 we make use of the following properties and concepts (based on the built-in CWM reasoner functions[11]):

- log:implies → Property that relates antecedents and consequents of a RDF expressed rule. Antecedents and consequents are defined between brackets { and }.
- log:content → This property relates the logical representation of an element with its string representation. It is used for defining the text content of an input, or for defining the specific values of an information.
- this → When it is used inside the semantic property of an element, it represents the element itself. It is used for adding knowledge to an element.

In order to facilitate sharing and reusing annotations, it is needed the use of ontologies for modeling this semantic content.

[11] http://www.w3.org/2000/10/swap/doc/CwmBuiltins.html

4 Data Extraction using Annotations

The content of the semantic property allows to locate a particular information inside the navigation map by performing a query that specifies the conditions in which the information can be found in the working memory. As our model deals with knowledge expressed as RDF triples, we have selected SPARQL[12] as query language for this purpose. SPARQL allows to indicate the Named Graph[6] that a set of conditions are referring. We make use of this functionality for relating annotations of different Deep Web sites in order to perform a distributed query among their graphs. Figure 3 shows an example of query that uses two different maps. The SPARQL query requires the price of a book that fulfills the following conditions expressed in the WHERE part of the query:

1. Select from RSS Web Site any element identified as Product that has a defined title
2. Select from Shop of Books Web Site any element identified as Product that has been searched in the site using the string that represents the title of the product from RSS Web Site previously obtained

A set of conditions expressed for a specific navigation model in the query can be achieved by a list of transitions. The actions of these transitions could need to provide client inputs. Also these inputs could be needed for accessing specific fragments in a state. In the example at figure 3, RSS map requires an input called INDEX for accessing a fragment inside rss_state0, and Shop of Books map requires the input SEARCHED_PRODUCT for performing the actions of transition0. Inputs are the points where the SPARQL query can relate data among different maps. Because inputs require that an information must be supplied, the SPARQL query must face with the possible situations with these rules:

- If the conditions states clearly the specific value of the input, use this value.
- If the conditions indicate that the information required by the input must be obtained from another graph, then the query must first perform its actions in that other graph. This happens in the example with Shop of Books map at condition ?product2 shop:searchedBy ?title because title is refered by the RSS map.
- If the information required can not be infered, then the SPARQL query must use all possible informations that could be used for the input. In the example, this happens with the input INDEX, that is not provided, so the query must iterate among all the elements.

We have developed a demo system that may be accessed through [14]. Our demo generates Web wrappers scripts composed of *Chickenfoot*[13] commands that could be executed in *Firefox* with *Chickenfoot* plugin installed. *Chickenfoot* is a programming environment that provides a set of special functions for

[12] http://www.w3.org/TR/rdf-sparql-query/
[13] http://groups.csail.mit.edu/uid/chickenfoot/

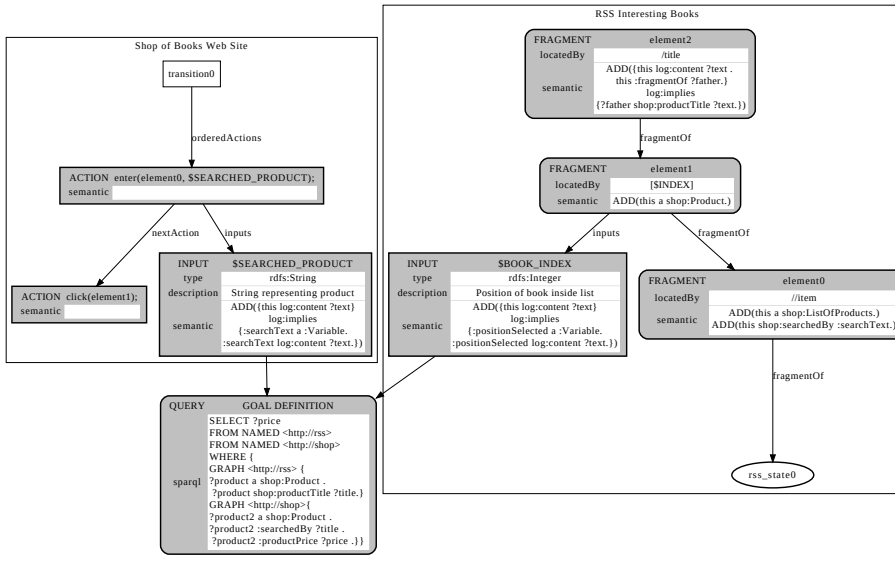


Fig. 3. Example querying over different maps

performing Web tasks like clicking a link, entering data in a HTML form, etc. *Chickenfoot* scripts are written in a superset of Javascript. One of its main characteristic is its support of natural language naming for HTML elements which eases development of Web automatic tasks to end users [3].

The scripts are generated from SPARQL queries that can use annotations expressed in the formalization that we have presented in this article, also accesible as OWL at URL [15]. Although with SPARQL we can not define very complex tasks, we think it is a good starting point for defining a more powerful language that may be used in a semantic-oriented Web Wrappers development. We think this new approach may be useful for solving the drawbacks of previous Web Wrappers development on Deep Web sites, like dependencies with Web structure, constant maintenance of wrappers scripts, the need of strong knowledge of the physical structure of accessed Web sites, etc.

5 Future works

Accessing and integrating data from non-structured heterogeneous sources is a problem that currently is solved with Web Wrappers despite its drawbacks. Web Wrappers are also tools that may be used to give structure to that non-structured information and made it available to the Web of Data. We think that wrapper development must be adapted to the new environment of linked data using semantic-oriented wrapper definitions, and not site-specific implementations. In order to achieve this, we are interested in researching more powerful

languages and development tools for this new semantic approach of semantic wrapper development.

We think our model will improve development of Wrappers, because all the Web site structure changes will require only to modify the annotation model, and all Wrappers that make use of the annotations will be corrected. However, it would be adequate an exhaustive evaluation of user effort with our annotation model, that will be accomplished in future work.

We are interested in continuing working on ontologies integration with semantic wrapper implementations in order to exploit the common concepts among different Web sites. The same semantic wrapper implementation may be used in any Web site that is annotated using the same ontology.

6 Acknowledgements

This work has been partially funded by the spanish Ministry of Education and Science, project ITACA No. TSI2007-65393-C02-01

References

1. Robert Baumgartner, Michal Ceresna, and Gerald Ledermuller. Deep web navigation in web data extraction. In *CIMCA '05: Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-2 (CIMCA-IAWTIC'06)*, volume 2, pages 698–703, Washington, DC, USA, 2005. IEEE Computer Society.
2. Jeffrey P. Bigham, Anna C. Cavender, Ryan S. Kaminsky, Craig M. Prince, and Tyler S. Robison. Transcendence: enabling a personal view of the deep web. In *IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 169–178, New York, NY, USA, 2008. ACM.
3. Michael Bolin and Robert C. Miller. Naming page elements in end-user web automation. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, 2005.
4. Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 163–172, New York, NY, USA, 2005. ACM.
5. Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, May 2004.
6. Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM.
7. Sudarshan Chawathe, Hector Garcia-molina, Joachim Hammer, Kelly Irel, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The tsimmis project: Integration of heterogeneous information sources. In *In Proceedings of IPSJ Conference*, pages 7–18, 1994.

8. Darris Hupp and Robert C. Miller. Smart bookmarks: automatic retroactive macro recording on the web. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 81–90, New York, NY, USA, 2007. ACM.
9. David Huynh, Stefano Mazzocchi, and David Karger. *Piggy Bank: Experience the Semantic Web inside your web browser*, volume 5, pages 16–27. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 2007.
10. Rohit Khare and Tantek Celik. Microformats: a pragmatic path to the semantic web. pages 865–866, 2006.
11. Dirk Kukulenz. Adaptive site map visualization based on landmarks. In *IV '05: Proceedings of the Ninth International Conference on Information Visualisation*, pages 473–479, Washington, DC, USA, 2005. IEEE Computer Society.
12. W3C. Rdfa primer. W3C Working Group Note 14 October 2008, October 2008.
13. Yang Wang and Thomas Hornung. Deep web navigation by example. In Tomasz Kaczmarek Marek Kowalkiewicz Tadhg Nagle Jonny Parkes Dominik Flejter, Slawomir Grzonkowski, editor, *BIS 2008 Workshop Proceedings, Inssbruck, Austria, 6-7 May 2008*, pages 131–140. Department of Information Systems, Pozna, University of Economics, 2008.
14. WebTLab. Site annotation demo. http://corelli.gast.it.uc3m.es/siteannotation.
15. WebTLab. Site annotation ontology. http://corelli.gast.it.uc3m.es/siteannotation/ontology.owl.
16. Jeffrey Wong and Jason I. Hong. Making mashups with marmite: towards end-user programming for the web. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1435–1444, New York, NY, USA, 2007. ACM.

Reputation-Based Self-Service Environments

Donato Barbagallo, Cinzia Cappiello, Chiara Francalanci, Maristella Matera

Dipartimento di Elettronica e Informazione, Politecnico di Milano,
Via Ponzio 34/5, 20133 Milano, Italy
{barbagallo, cappiello, francala, matera}@elet.polimi.it

Abstract. The availability of a huge amount of information on the Web raises a set of issues concerning the research, the selection, and the representation of trustworthy sources and services. This paper describes a new research, which we have recently started and that aims at developing a platform to support users in the creation of mashup-based personalized self-service environments, where the users can access dependable services, selected on the basis of their reputation. This paper illustrates the motivations behind our research, and introduces some preliminary ideas about the platform design.

Keywords: Self-Service Environments, Mashups, Reputation, Quality.

1 Introduction

The Web is a huge and heterogeneous source of information. Web 2.0 technologies have enabled an active role of the users, who can create and make available their contents very easily [7]. This allows people to express their opinions, and to distribute them through several means, such as forums, blog posts and comments, and social networks. It therefore becomes possible to access other people opinions, direct witnesses and spread ideas bypassing traditional and official sources of information such as corporate websites. Of course, the availability of such an amount of information raises a set of issues concerning the research, the selection, and the representation of trustworthy sources and services. The information retrieved on the Web is indeed often characterized by inconsistent, incomplete, and erroneous data, and users are not able to distinguish the right or the most suitable data along their needs. Furthermore, all the accessible sources could be better exploited if they are combined in a mashup wise, so that to obtain a tangible added value. To respond to the previous needs, this paper discusses some issues behind the provision of personalized self-service environments where users can build their view over the Web information space, by integrating trustworthy services for information access.

The remainder of this paper is organized as follows. Section 2 illustrates a scenario that clarifies the novel requirements addressed by our research. Section 3 describes the architecture of the platform that we want to build, while Sections 4 and 5 discuss the two salient themes addressed by our project, namely reputation assessment and the construction of self-service environments based on mashup technologies. Finally Section 6 outlines future work.

2 A Reference Scenario

To understand the idea at the basis of our research, let us consider a usage scenario in the context of *patient empowerment*. We suppose that an individual with a health

problem is searching for a dependable source of information, such as a forum where to get advice on his/her specific problem. This raises a number of issues: (i) the selection of the most dependable forum, (ii) understanding who other participants are trustworthy, and eventually (iii) integrating different answers from multiple forums and dynamically monitor the most reliable sources of information on specific subjects. In addition to discovering the best sources of information, the individual might also add new value to the identified dependable sources by combining different trusted services for creating his/her personalized information access, e.g. combining the identified forum with a map service showing the location of the hospitals mentioned in the forum, as well as with services for accessing rankings, news, and images. This implies the availability of methods for easy service aggregation, which can be based on *mashup* technologies.

To the state of the art, the current technology does not exhaustively cover the previous needs. Our aim is therefore to provide a platform satisfying them.

3 Platform Architecture

The previous scenario highlights two fundamental needs: (i) to select dependable services that can fulfil specific information needs and quality requirements and (ii) to provide users with tools to compose on-demand personalized applications by means of the selected dependable services.

To respond to these needs, we aim at designing a platform providing self-service mashup functionalities, to help people create a personalized Web access environment. Based on a user profile, the platform searches for the "best" information sources, selects relevant and authoritative information, and then wraps it as data services. Data services are then mixed with other services, typically mashup components [8], to create a personalized environment based on mashup technologies.

Figure 1 illustrates the main components of the proposed platform. The *Service Registry* stores a catalogue of services that can be used for the creation of the personalized user environment. They can be *data services*, providing a binding with a dependable data source through a description of the data structure, generic Web services enabling the retrieval of some relevant information, or also UI components [8]. The registry is populated by a domain expert, who is in charge of scouting relevant data sources. A *Reputation Monitor* evaluates the services from the catalogue, and assigns measures based on objective reputation criteria (e.g., institutional reputation). Such assessment is continuously updated: periodically, the *Reputation Monitor* verifies and updates the quality level of the data sources. The output of the assessment activity is a *Reputation Descriptor*, which stores the result of the computation of the reputation measures.

The *Broker* selects the services that best match the specified user's settings (e.g., information needs expressed in the queries that the user submits to the platform) and the user's profile. The services selected by the broker are then used by the self-service environment that helps users create their personal information access environment.

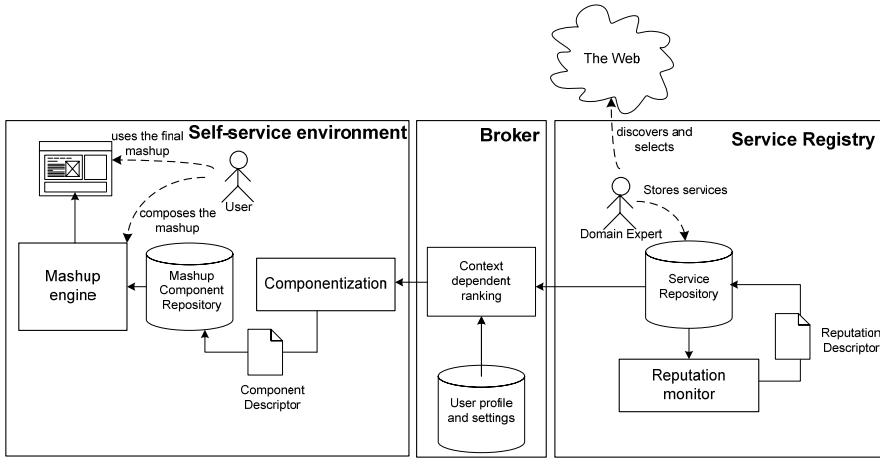


Figure 1- Platform architecture

The *Self-Service Environment* is centred on the availability of a *mashup engine*, through which the user can select some relevant components from the *Mashup Component Repository* and combine them to generate new value. The components available in the repository are those previously selected by the broker. In order to be combined into a mashup, these services need to be *componentized*, i.e. each service must be associated with a descriptor highlighting the properties useful for combination and choreography purposes.

When the services selected by the broker are not provided with a proper user interface, as it happens for pure data services, the componentization process also requires the generation of a presentation layer. This process implies the selection of some visualization widgets (from a widget repository) that best match the operations and the exchanged data as indicated by the service descriptors (WSDL descriptor, API, or also additional service profiles). The previous functionalities raise the research challenges described in the following sections.

4 Reputation Assessment

The selection of the "best" sources, the assessment of their trustworthiness, and the integration of the relevant contents are all based on the ability to assess the reputation of the information sources. The concept of reputation is the result of the assessment of several quality properties of information sources, including correctness, completeness, timeliness, dependability, and consistency [3]. The literature provides consolidated data quality techniques in the case of structured data. For example, a classical approach to data brokering in the context of syndicated data is represented in [1], where data are structured and query answers are fully integrated and returned to users as a table (or a set of tables), as with traditional databases. This work has been subsequently extended with the concept of reputation [2]. In a multi-source context, it proposes to assess the reputation of each information source by means of i) an a-priori assessment of the reputation of the information source, based on the source's authority in a given field and ii) an assessment of the source's ability to offer relevant

answers to user queries based on historical data on the source collected by the broker as part of its service. This approach is original since it defines reputation as a context- and time-dependent characteristic of information sources, and leverages the ability of the broker to keep a track record of each source's reputation over time, but it should be extended with dimensions related to the quality of mashup components to be integrated into the final self-service environment. Quality criteria for the different aspects of traditional software applications have been proposed and thoroughly analyzed in the literature, but the adaptability, dynamicity, and heterogeneity that characterize the mashup ecosystem require a separate and focused analysis.

Our research goal is therefore to define an assessment of the reputation of an information source, e.g., a Web page, based on the assessment of a number of properties of the sources along the traditional data and software quality dimensions, and also on the indirect assessment of the information sources and contributors that the original source includes. This involves the selection of relevant quality dimensions, their operating definition, their metrics, and their composition into an overall assessment of reputation that leverages the historical and contextual knowledge base of the broker.To understand the methodological approach that we will implement in our platform, let us assume that the Web page belongs to a forum. First of all, the Reputation Monitor in Figure 1 must provide an evaluation of the forum's reputation as information source. A source's *completeness* represents a fundamental quality dimension. The completeness of the information provided by a forum could be defined as the breadth of the forum in terms of number of users and issues raised, which, in turn, would lead to the operating definition of completeness as the total number of subscribers/contributors and their participation rate in terms of number of posts per day. Then, we could evaluate the forum's reputation along the *dependability* dimension in terms of the probability with which a post receives an answer, which leads to define dependability as the number of answers per post. This metric could be corrected with a semantic analysis of the relevance of responses. Subjective measures should also be considered, such as comments on the quality of a contribution that the site records explicitly, as different types of ratings, which can be internal to the forum, or external, i.e., stored by sites that aggregate and rate other sites' news. Numerous time-dependent measures of reputation could also be defined, for example along the *timeliness* dimension, by considering metrics such as the frequency of update of a site, its age, etc.

Behind assessing the reputation of the information sources, the broker must be able to evaluate the quality of the mashup components that must provide access to the information source in the self-service environment, also trying to assess the overall quality of possible integrations of multiple components. For this purpose, quality criteria for the different aspects of traditional software applications already proposed in the literature can be exploited, but the adaptability and dynamicity that characterize the mashup ecosystem require a separate and focused analysis. In fact, mashups integrate heterogeneous components available on the Web, such as RSS/Atom feeds, Web services, wrapped content or programmable APIs (e.g., Google Maps). It is self-evident that the quality of the final combination is strongly influenced by the quality of each single component. If we look at components as standalone modules, then we can say that their quality is determined by the attributes that traditionally characterize software quality. However, it is necessary to consider that the publication of mashup

components through APIs hides their internal complexity and, therefore, also their internal details. Given that from the outside component-internal properties cannot be assessed, a novel quality model is required in order to characterize an "external" quality model. Selected quality attributes should be able capture the specific requirements deriving from the components' intended use, i.e., their combination within mashups. In fact, after a component has been deployed, external quality factors are the only criteria able to drive the evaluation of its suitability into mashup compositions. We have already defined a preliminary model of external quality to fill the literature gaps discussed above [4]. Further work is however needed to extend the broker for the coverage of such new quality requirements.

5 Self-Service Environments

Mashups are innovative applications that create new value out of the services they integrate, in that they combine them in a novel, value-adding manner and thereby provide a functionality that was not there before. A variety of mashup tools (e.g., Yahoo Pipes, Google Mashup Editor, Intel Mash Maker, Microsoft Popfly, and IBM QEDWiki - now part of IBM Mashup Center) have recently emerged. Their principal goal is to facilitate the combination of components via simple, graphical user interfaces, sets of predefined components, and abstractions from technicalities. However, they all assume the existence of ready-to-use components published on the Web, while they neglect the ensemble of issues related to the creation and selection of trusted components.

Our research project will exploit the availability of a consolidate mashup environment, *Mixup* [8], which supports the fast development of Web applications based on the mashup of UI components. The distinguishing characteristic of Mixup is that it focuses on the integration of components at the presentation layer, leaving application and data management logic inside components. It also makes use of component and composition models that are inspired by the research on Web services and the service-oriented architecture (SOA). The component model specifies the *events* that a component can generate and that communicate to the outside world changes in the internal state. It also specifies the *operations* which enable the outside world to modify the internal state of a component. Given such abstract component description, the composition logic is then described through an event-driven composition model, where events from one component may be mapped to operations of one or more other components; mappings are expressed by means of so-called listeners. In addition to the direct mapping of events to operations, listeners also support data transformations in form of XSLT transformations, and the specification of more complex mapping logics via inline JavaScript. The definition of listeners represents the composition logic, while the layout of a composite application is specified by means of a suitable HTML template that contains placeholders, which can be used at runtime to embed and execute components, thereby re-using their UIs.

Within this project, we will exploit Mixup as the basic engine for allowing users to mashup their self-service environment. The new issues to be investigated are related to the componentization of trusted services. We want to define some techniques to turn services providing dependable information (Web services, data services, but also Web applications publishing relevant and trusted information) into components for

mashup combination. In [5], we already investigated a technique for turning Web applications into mashup components. The idea is that the HTML of generic (dynamic) Web pages can be augmented with annotations for event and operation tagging and combined with a descriptor specifying such events and operations. A wrapper then provides programmatic access to the application, i.e., an API. Under particular conditions, both the descriptor and the wrapper can be automatically generated from the HTML tags.

We expect to further improve the previous results by introducing techniques for the componentization of generic Web and data services. Such may require mechanisms for the automatic selection of visualization widgets suitable for rendering the service data. Based on the semantic description of services, suitable UI widgets are used to display the data (e.g., a regional map is used to display postal addresses, 2D graphs are used to display historical financial data, graphs with nodes and arcs are used for depicting social networks). For this purpose, a registry of UI widgets, together with their semantic descriptions, will be designed and implemented. This feature, which will go beyond the provision of a mere Web programming environment exploiting ready-to-use components, is to our knowledge still unexplored.

6 Conclusion and Future Work

This paper has presented some ideas about a new research project that deals with the construction of self-service environments for the access to dependable information services. We are now working at the platform development, acting on two different fronts that reflect the two fundamental themes also discussed in this paper. On the one hand we are extending the quality broker with techniques for reputation assessment and for the assessment of the quality of mashup components [4]. On the other hand, we are developing an environment for the easy creation of mashup components, based on an automated technique for user interface construction. Our research is still in its infancy. However, given the recent trends in Web information access, we believe it is very promising and we hope to get soon sound results.

References

1. Ardagna, D., Cappiello, C., Comuzzi, M., Francalanci, C., Pernici, B.: A broker for selecting and provisioning high quality syndicated data. In: 10th International Conference on Information Quality (ICIQ 2005), pp. 262-279, MIT Press, Boston, 2005.
2. Ardagna, D., Cappiello, C., Francalanci, C.: Reputation-based brokering of multisource data. Technical Report n. 12/2009, DEI - Politecnico di Milano, Jan. 2009.
3. Batini, C., Cappiello, C., Francalanci, C., Maurino, A.: Methodologies for data quality assessment and improvement. ACM Comp. Surveys, 41(3), Sept. 2009.
4. Cappiello, C., Daniel, F., Matera, M.: A quality model for mashup components. Proc. of ICWE 2009.
5. Daniel, F., Matera, M.: Turning Web applications into mashup components: issues, models and solutions. Proc. of ICWE'2009.
6. Ennals, R. Garofalakis, M.N.: MashMaker: Mashups for the Masses. Proc. of SIGMOD 2007, pp. 1116-1118.
7. Murugesan, S.: Understanding Web 2.0. IT Professional, 9(4), pp. 34-41, 2007.
8. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M.: A framework for rapid integration of presentation components. Proc. of WWW 2007, pp. 923-932, 2007.

User Profile Elicitation and Conversion in a Mashup Environment

Erwin Leonardi[1], Geert-Jan Houben[1,2], Kees van der Sluijs[2], Jan Hidders[1], Eelco Herder[3], Fabian Abel[3], Daniel Krause[3], Dominikus Heckmann[4]

[1] Delft University of Technology, PO Box 5031, 2600 GA Delft, the Netherlands
{e.leonardi, g.j.p.m.houben, a.j.h.hidders}@tudelft.nl
[2] Technische Universiteit Eindhoven, PO Box 513, 5600 MB, Eindhoven, the Netherlands
k.a.m.sluijs@tue.nl
[3] L3S Research Center, Appelstr. 9a, 30167 Hannover, Germany
{herder, abel, krause}@l3s.de
[4] German Research Center for Artificial Intelligence, Saarbrucken, Germany
heckmann@dfki.de

Abstract. Many Web applications have offered personalization and adaptation as their features in order to provide personalized services to their users. The user profiles are gathered independently by these applications often through an explicit dialogue with the user. As a result, the users have to go through a similar elicitation process multiple times, that is, providing similar information that is used to build the user profiles to different applications. In the springtime of mashup applications, we observe the importance of considering user information in order to make the presented content more relevant to the user. For this purpose, it is necessary to have a platform/framework that enables components in the mashup to reuse and exchange user profiles. In this paper, we present the Morpho framework that elicits, enhances, and transforms a user profile from one application to another application in a mashup environment. It deals with *semantic* and *syntactic heterogeneity* of data and schema of the user profile. We present the architecture of Morpho and a case study to exemplify the approach followed in this current work.

Keywords: user profile, interoperability, mashup

1 Introduction

With the evolution of the Web, Web applications have become more complex and offer their users many interesting and advanced features. Adaptation and personalization become important features of today's Web applications. In order to be able to adapt and offer personalized contents for a specific user, a Web application must have enough information about this user. This information can be gathered explicitly and implicitly. The explicit approach is by asking directly to the user, for example, by using a survey form or by asking the user to give ratings to certain products, thus building up a user profile. In the latter approach, the Web application monitors the behaviors of the users while the users use the application in order to construct a user model fitting with the goal of the application.

Parallel to the growing of the Web, the number of Web applications offering various services has also increased significantly. Consequently, Web users may be using more than one application that offer different products/services, and keep adding new applications. Each of these applications independently asks for the user profiles of its users that are used to provide personalized contents and services. They do not share or learn from one another. This leaves no choice for the users but to go through the same process again and again. That is, the users have to, for example, fill in different questionnaires for different applications or rate products until the applications have enough information to describe their interests. This can be a cumbersome thing to do for many users and something for which support to help them in constructing these profiles is welcome.

Nowadays a new breed of Web applications called *mashup* has been deployed in several areas and increasingly becomes more popular. A mashup combines data from two or more sources into a single integrated tool. The data that originally belongs to other Web applications is blended in order to provide enriched user-oriented contents. Note that the Web applications from which a mashup tool fetches data may have their own specific ways to describe information about their users. We suggest that by considering information about a user we can have a better mashup - a mashup that provides *more relevant* contents for the user. In addition, the sharing of user profiles facilitates a better integration and cooperation between underlying applications in the mashup. For example, consider a mashup Web application named BookTour[1] that combines Google Earth[2] and Amazon[3] to show book tours happening around the world. It would be more relevant and interesting for a user if BookTour shows only the events that are related to, for example, favorite authors of this user based on the books in Amazon that he/she buys and rates. So, BookTour uses what it knows about the user to ask more specific queries to Amazon resulting in more relevant information from Amazon. Observe that a user may only use several applications provided in a mashup, but never uses the rest of the available application. This causes the absence of a user profile in some of these applications. Consequently, these applications may not be able to provide customized and personalized contents for this user. The challenge is to develop a mechanism to reuse the information about a user, which is originally from one application, for another application [1,2]. We refer to this as *user profile interoperability*.

One technique for user profile interoperability is by using a unified (central) model that serves as a predefined structure and is easily exchangeable and interpretable [1,11]. However, it is impractical to force Web applications to use a unified model because the Web is an open and dynamic environment [3]. A more flexible alternative approach is to provide a framework that elicits the data in the user profile of one application and *transforms* it into a user profile of another application. Thus the mashup can use the profiles to retrieve more relevant content from the underlying applications. If we do this integration by exploiting Semantic Web techniques, it can be flexible because the applications do not need to follow a fixed model for their user profile. This technique raises some main challenges, that is, to deal with *semantic* and

[1] http://www.booktour.com/
[2] http://earth.google.com/
[3] http://www.amazon.com/

syntactic heterogeneity of data and schema of the user profile. The flexibility offered by this approach has a trade-off. The main advantage is that the "glue" that the mashup represents can be applied more precisely and that the connection between mashup and base applications works at an increased relevance level. However, in many situations (especially when starting the use of an application at the-so-called cold start [20]) an increase in relevance is welcome. Of course, a transformation of profiles will not be perfect as it may lead to the possibility of losing data during transformation process. It is also possible that the transformation cannot be made because the model is simply incompatible.

In this paper, we propose a framework (called Morpho) that aims at eliciting the user profile of an application and transforming it to the one of another application. It is a part of our *User Pipes* project that aims to allow user profile reasoning by mashing up different user profile data streams. It can be used as a component in a mashup application to ensure user profile interoperability and sharing between other mashup components. It deals with semantic and syntactic heterogeneity of user profiles and ensures the interoperability of user profiles of different applications. In addition, it is configurable and extensible as the mashup application administrators (the *administrators*) are able to specify a configuration by which elicitation and transformation processes are guided. Note that this framework works with the assumption that user profile data can only be accessed and exchanged with explicit consent of the owner. This assumption is important because the privacy issue is critical for user profiles and modeling [4]. As Morpho can be used as a component in a mashup application, the explicit consent given by the mashup application users (the *end-users*) to the mashup application is also given to Morpho.

This paper reports on our ongoing work related to Morpho and User Pipes and it is structured as follows. In Section 2, we present related work. Section 3 presents a motivating case study that is used as running example in the discussion. Section 4 discusses the architecture of the Morpho framework and elaborates on user profile elicitation and transformation. Finally, Section 5 concludes the paper by highlighting some future directions of this research.

2 Related Work

To address the user profile interoperability there are basically two approaches: the *shared format* approach and the *conversion* approach. In the shared format approach, a common language for a unified user profile (*a lingua franca*) is needed. Applications have to follow the unified format [13]. Examples of this approach are the General User Model Ontology (GUMO) [1] and Composite Capability/ Preference Profiles (CC/PP)[4]. This approach is easily exchangeable and interpretable as there is no syntactic and semantic heterogeneity issue to be addressed [1]. However, this approach is not suitable for open and dynamic environments, such as the Web, as it is impractical and in many cases impossible to enforce Web applications to follow the *lingua franca* [3]. The conversion approach is more flexible and suitable for open and

[4] http://www.w3.org/Mobile/CCPP/

dynamic environments [14]. In this approach, a technique has to be developed for converting a user profile of one application to another application. The developed technique should deal with the problem of syntactic and semantic heterogeneity. Observe that potential drawbacks of this approach are that it is possible that some information is lost during the conversion process, and that it is possible that models are simply incompatible. This means that there is no suitable mapping for these models. It is also possible that the mappings are incomplete because required information in one model is not available in the other model.

Mashing up data and tools into one integrated tool has become increasingly popular recently [15, 16, 17]. One work in the mashup related environment that is closely related to our research is presented in [18, 19]. In [18,19], Gosh et al. present a framework called SUPER (*Semantic User Profile Management Framework*) for capturing and maintaining user profiles using semantic web technology in the retail domain. SUPER aggregates user profile information that spreads over several services/data sources.

There is a host of related work about user profiling and about mashup that could be discussed; however, we do not discuss this in this workshop paper.

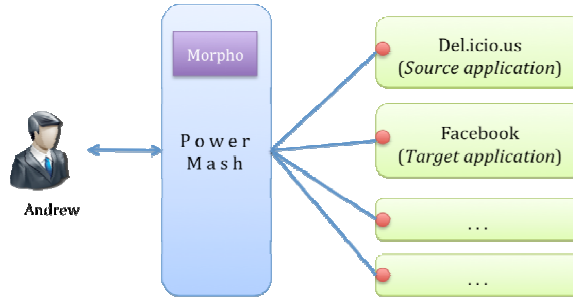


Figure 1: The PowerMash Application

3 Motivating Case Study

In this section, we present a case study to exemplify our approach discussed in the subsequent sections. Consider an end-user named Andrew that uses a mashup application called PowerMash as shown in Figure 1. Two of the underlying applications are Del.ious.us[5] and Facebook[6]. Suppose Andrew has used Del.icio.us to bookmark web pages that are interesting for him. He has a Facebook account; however, he has not filled up many parts of his Facebook profile. PowerMash uses the Morpho framework as its component for user profile elicitation and transformation. In this scenario, the challenge is to see to what extent Morpho is able to use Andrew's

[5] http://delicious.com/
[6] http://www.facebook.com/

bookmark entries in Del.icio.us to help him enhance his Facebook profile as much as possible.

Figure 2 depicts three bookmark items of Andrew. Each bookmark item in Del.icio.us contains fields of information about creation date, URL, title, note/description, and tags for the bookmark item. The URL field identifies the resource location (or web address) of a bookmarked web page. The title field stores the title of a bookmarked web page. A user may write additional description or information about the web page. This information is maintained in the note field. Tags are typically one-word descriptors that a user can assign to his/her bookmarks to help him/her organize and remember them. Each bookmark item can have many tags and they do not form a hierarchy. Note that the note and tags are optional. Consider the third bookmark item shown in Figure 2. This bookmark item was created on 8 April 2009 and has URL `www.youtube.com/watch?v=73-V2A3NuWo` and title "*Kelly Clarkson – Because of You*". Andrew has put a note "*High quality video from official Kelly Clarkson channel at YouTube*" and given the tags "*pop"* and *"song*".

For the purpose of enhancing and improving Andrew's Facebook profile (as much as possible), PowerMash employs Morpho to elicit Andrew's user profile in the form of bookmark entries in Del.icio.us and transform them into elements of his Facebook profile. In this situation, PowerMash retrieves Andrew's bookmark entries from Del.icio.us. Note that the communications between a mashup application and its underlying basae application can be performed by using API, RSS feeds, RESTful service, or SOAP service. Since Morpho is for user profile elicitation and transformation, the way a mashup application retrieves and sends data to its underlying applications or web services is beyond the scope of our work in this paper. Having retrieved Andrew's bookmark entries, PowerMash sends a request to Morpho to transform his bookmark entries into data for his Facebook profile.

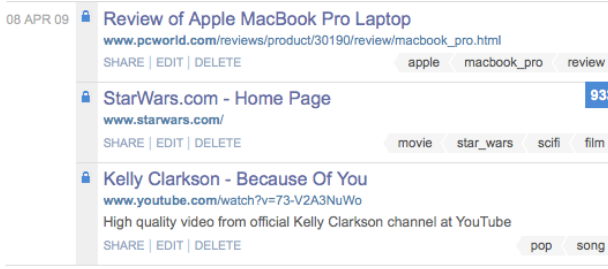


Figure 2: Three Bookmark Items of Andrew from Del.icio.us

4 Morpho Framework

Figure 3 depicts the architecture of the Morpho framework. Let us first give the overview of the modules in the framework. We will elaborate them in details in the subsequence sections. As the name suggests, the *Interface* module manages the interactions between the framework and a mashup application, in our case, PowerMash. It allows the mashup to submit inputs to the framework and to receive

the converted user profiles produced by the framework. Next to the Interface is the *Controller* module that serves as the main controller of the framework. The *Model Builder* module processes the source user profile (in our case, Andrew's bookmarks) and maps it into a corresponding internal conceptual model using a set of mapping rules. Then, it generates RDF based on this conceptual model, and stores it in the *User Profile Repository*. A set of concepts has to be extracted from the source user profile. Extraction is necessary because the content of a user profile is typically text that is *meaningless* for machine. The *Concept Extractor* performs this task by using available knowledge bases, in our case, DBpedia. The conversion of a source user profile to a target user profile takes place inside the *Interoperability Engine*. This engine measures the distance between the concepts in the source user profile and the concepts related to target user profile schema. Finally, the *Result Builder* generates the user profile of the target application. To simplify the discussion in our running example we consider only to fill up the '*favorite music'* field of the Facebook profile.

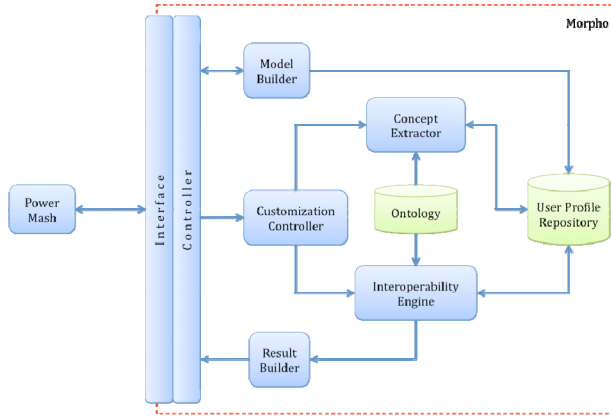


Figure 3: The Architecture of Morpho

4.1 Controller

After the mashup application sends the necessary input to the framework via the Interface, the Controller starts its task of managing the elicitation and transformation of user profiles. All inputs received from the mashup, except the configuration, are fed to the Model Builder. The configuration is sent to the Customization Controller. In some cases, the source and target applications are from the same application domain. For example, the source and target applications can be both social network sites. If the Controller detects such cases, several steps can be skipped. Let us elaborate further on this. Suppose the source application is Facebook and the target application is Hyves[7]. A set of mapping rules transforms a profile from Facebook into RDF based on Morpho internal conceptual model for social network sites. Then, this RDF is mapped into Hyves profile using another set of mapping rules that is

[7] http://www.hyves.nl/

specifically used to transform Hyves profile into Morpho internal conceptual model for social network sites and vice versa. Observe that even though two applications from the same application domain share the same Morpho internal conceptual models, the user profile transformation among them will not be perfect as during transformation process it is possible to lose data. It is also possible that required information in target profile is not available in the source profile.

4.2 Model Builder

The main task of Model Builder is to parse and map the source user profile to a specific internal conceptual model in Morpho, and to transform this user profile into RDF based on this internal conceptual model. The mapping functionality is application specific. This means that for a particular Web application a set of *mapping rules* has to be defined. The internal conceptual model is domain specific. We use the same internal conceptual model for all applications that belong to the same application domain. For example, if the source application is a social network site, then the internal conceptual model can be based on the OpenSocial/RDF[8] and FOAF specification[9]. Another example is that for bookmark data, we use our internal bookmark conceptual model that is inspired by the Annotea Bookmark Schema[10]. In our running example, the Model Builder transforms Andrew's bookmarks depicted in Figure 2 into RDF as follows:

```
<rdf:RDF ... >
  <morpho:Bookmarks>
    <morpho:hasItem rdf:resource="Bookmark_1" />
    ...
  </morpho:Bookmarks>
  <morpho:Bookmark rdf:ID="Bookmark_1">
    <a:created>2009-04-08T09:54:49+0100</a:created>
    <dc:title>Kelly Clarkson - Because Of You</dc:title>
    <dc:description>High quality video from official Kelly
         Clarkson channel at YouTube</dc:description>
    <b:recalls>http://www.youtube.com/watch?v=73-V2A3NuWo</b:recalls>
    <a:hasAnnotation>pop</a:hasAnnotation>
    <a:hasAnnotation>song</a:hasAnnotation>
  </morhpo:Bookmark>
  ...
</rdf:RDF>
```

In addition, the Model Builder also builds an RDF model for the target user profile based on the target application and annotates it with default pre-defined related concepts. For instance, the property '*music*' in the social network conceptual model that describes the user's favorite music is annotated with the DBpedia concept labeled '*music*' (e.g. dbpedia:Category:Music). This RDF model acts as the schema for the target user profile. Its instances are generated from the source user profile. The generated RDF is stored in an RDF repository (e.g. Jena [6] and Sesame [7]).

[8] http://web-semantics.org/ns/opensocial
[9] http://xmlns.com/foaf/spec/
[10] http://www.w3.org/2001/Annotea/

4.3 Customization Controller

Our framework employs various tools or external components that perform certain tasks in our modules, namely, in the Concept Extractor and Interoperability Engine modules. We shall discuss how these tools are utilized in the subsequent sections. One of the important features that we have is to allow the administrators to be able to guide how some modules should work by specifying the preferable combination of tools and components. For example, an administrator may prefer to aggregate the similarity scores that are returned by Levenshtein Distance [7] and Soundex [8] for the lexical matching step discussed in subsequent section. He/she might also want to specify, for example, which properties should be followed while expanding concepts. The Customization Controller takes the preferences and feeds the Concept Extractor and Interoperability Engine modules with necessary settings. The administrators do not specify their preferences, and then a default configuration is used.

4.4 Concept Extractor

The RDF data stored in the RDF repository is still *raw* and cannot be used directly to generate the target user profile. We need to connect this data to the concepts, in our case, the DBpedia concepts, such that it is meaningful for the machine. The Concept Extractor has to determine a set of concepts out of the RDF of the source user profile, in this case, Andrew's bookmarks. Note that this module is inspired by Relco [9], a tool for relating tags to concepts. The Concept Extractor works as follows.

Firstly, the Concept Extractor finds a set of keywords from the information available in Andrew's bookmark items. The keywords can be discovered using two tools for natural language processing, namely, Part-Of-Speech Tagger (POS Tagger) [10] and Named Entity Recognition (NER) [11] tools. A POS tagger is a piece of software that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective, etc., although generally computational applications use more fine-grained POS tags like 'noun-plural'. NER labels sequences of words in a text, which are the names of things, such as person and company names. It facilitates the framework to determine, for example, the person name. We consider words that are detected as noun and preceded by zero or more adjectives as the keywords. In addition, one or more words, which are determined as the entity names, are also considered as a keyword. For example, the POS tagger assigns parts of speech to each word in the description of the third bookmark item of Andrew as follows:

```
High quality video from official Kelly Clarkson channel at YouTube
 JJ    NN      NN   IN    NN       NN    NN       NN    IN   NN
```

There are seven keyword candidates: '*high quality*', '*video*', '*official*', '*Kelly*', '*Clarkson*', '*channel*', and '*YouTube*'. The NER returns '*Kelly Clarkson*' and '*YouTube*' as possible entity names. Combining the results of these tools, we have six keywords: '*high quality*', '*video*', '*official*', '*Kelly Clarkson*', '*channel*', and '*YouTube*'. Note that another sets of keywords are extracted from URL, title, and tags

of the bookmark items. We use the instance of the *morpho:Keyword* class to describe the discovered keywords:

```
...
<morpho:Bookmark rdf:ID="Bookmark_1">
   ...
   <morpho:Keyword ID="Keyword_1">
      <morpho:keywordValue>high quality</morpho:keywordValue>
   </morpho:hasKeyword>
   <morpho:Keyword ID="Keyword_2">
      <morpho:keywordValue>video</morpho:keywordValue>
   </morpho:hasKeyword>
   ...
```

The next step is to *lexically* match the discovered keywords from Andrew's bookmarks to a set of candidate concepts from DBpedia. A concept in DBpedia is of type *skos:concept* and usually has a property that describes the label of this concept, such as *rdfs*:*label*. In some cases, a concept can have multiple labels denoted by, for example, *skos*:*prefLabel* and *skos*:*altLabel*. These properties are in the SKOS vocabulary[11]. These matches give us a set of candidate concepts that may be syntactically related to the keywords together with their similarity scores. For example, in our running example, the DBpedia concept labeled '*Video'*[12] could be a good matching for the keyword '*video'*. Observe that the labels that are encoded in URIs are also considered [9]. In our implementation, the open source SimMetrics library[13] is used. This library employs many well-known similarity metrics such as Levenshtein Distance [7], Soundex [8], etc. The administrators can choose which metrics they want to use and how they are aggregated. The instance of *morpho*:*RelatedConcept* is used to describe the DBpedia concepts that are related to the keywords:

```
...
<morpho:Keyword ID="Keyword_2">
   <morpho:keywordValue>video</morpho:keywordValue>
   <morpho:RelatedConcept>
      <morpho:extConcept rdf:resource="DBpedia:Category:Video" />
      <morpho:similarityScore>1.0</morpho:similarityScore>
   </morpho:relatedToConcept>
   ...
```

The third step is to find other DBpedia concepts that are *semantically* related to the DBpedia concepts discovered in the previous step. This can be done by following and exploiting some properties of the concepts, for example, rdfs:subClassOf, *skos*:*related*, or *skos*:*broader*. The administrator can configure additional properties that the framework should follow during semantic structure exploitation. In our running example, the concept labeled '*Video'* is semantically related to the concepts labeled '*Music and video'* and '*Film'*. Thus, they can also be related to the keyword '*video'* extracted from Andrew's bookmarks. These found related concepts can be useful and might be a good alternative to the original concepts [9]. The *morpho*:*relatedToConcept* is also used to the new discovered concepts. The similarity

[11] http://www.w3.org/TR/skos-reference/
[12] http://dbpedia.org/page/Category:Video
[13] http://sourceforge.net/projects/simmetrics/

score of these new discovered concepts are based on the similarity score of the original concept, but lowered by a configurable reduction factor.

The previous step might result many related concepts for each keyword. These related concepts are ranked according to their similarity scores; however, by knowing the properties of the source user profile, the related concepts can be processed and refined further. In our running example, we have a set of bookmark items of Andrew's Del.icio.us bookmarks. Each bookmark item maintains information (e.g. URL, title, etc.) about *one* bookmarked website. By knowing this, we are able to disambiguate and to better select concepts that are most appropriate for each bookmark item. Intuitively, the probability is high that the keywords from a bookmark item will relate to concepts that are closed to each other. Consider the third bookmark item in Figure 2. For instance, the keyword '*video*' is related to DBpedia concepts labeled '*Video'*, '*Music and video'*, and '*Film'*. However, if we consider another keyword from the third bookmark item, for example, keywords '*pop*' and '*song*', then the concept labeled '*Music and video'* is more appropriate and relevant for the third bookmark. Observe that the keywords '*pop*' and '*song*' are related to the DBpedia concepts labeled '*Pop music*' and '*Songs*', respectively. These concepts are closer to the concept labeled '*Music and video'* than to the ones labeled '*Video'* and '*Film'*. Note that maximum distance between concepts to be considered as close to each other is configurable by the mashup application administrators.

4.5 Interoperability Engine

Section 4.4 discussed how the concepts are extracted from the source user profile (e.g. Andrew's Del.icio.us bookmarks). In this section, we elaborate on how these extracted concepts are related to the annotated DBpedia concepts in the conceptual model of the *target* user profile (e.g. DBpedia concept labeled '*Music*' that describes favorite music properties in the Morpho internal conceptual model of Facebook).

The Interoperability Engine measures the distance between the concepts that are extracted from the source user profile and the annotated DBpedia concepts in the conceptual model of the *target* user profile. To measure the similarity and relatedness, we employ DBpedia Relationship Finder [12] that is able to compute the distance between two objects/concepts in DBpedia. In our running example, the Interoperability Engine computes the distances between the DBpedia concepts related to Andrew's bookmark items and the DBpedia concepts labeled '*Music*'. The first bookmark item in Figure 2 is related to the DBpedia concept labeled '*Laptops*'. Recall that the third bookmark item is related to the DBpedia concept labeled '*Pop music*'. The path from the concept labeled '*Laptops*' to the concept labeled '*Music*' is much longer than the path from the concept labeled '*Pop music*' to the concept labeled '*Music*' and therefore the concept labeled '*Laptops*' is considered not relevant for the concept labeled '*Music*'. Then, the Interoperability Engine establishes link between the concept labeled '*Pop music*' and the concept labeled '*Music*' using a property *morpho:isRelevantTo*.

4.6 Result Builder

In the previous step, a set of concepts that is semantically related to the concepts in the target schema has been determined. The Result Builder exploits the *morpho*: *isRelevantTo* property and maps the target user profile (based on our internal conceptual model) to the one of the target application using a set of defined mapping rules. In our running example, the concepts '*Kelly Clarkson'*, '*music video'*, and '*pop music'* are relevant for the '*favorite music*' in Facebook and can be used to enhance it.

5 Conclusions and Future Work

In this paper we emphasize the importance of sharing and exchanging user profiles between underlying applications in personalized mashup applications. We suggest that by considering and sharing information about a user we can have a better mashup - a mashup that provides *more relevant* contents for the user. In addition, the sharing of user profiles facilitates a better integration and cooperation between applications in the mashup. We also propose a framework called Morpho that is a part of our *User Pipes* project. It is used to elicit a user profile from an application, and transform it to a profile for another application. It can be employed as a component in a mashup application and helps the mashup to perform user profile interoperability. Also, it is extensible and configurable as the mashup application administrator is able to specify settings that guide some processes in Morpho.

In the e-learning domain, in the context of the GRAPPLE project we are also working on a framework called GUMF (Grapple User Model Framework) for exchanging user model of various e-learning systems. The idea behind GUMF and Morpho is similar that is to enable user profile interoperability of various applications. However, GUMF is specifically designed and configured for the e-learning domain, while we intend to make Morpho applicable for various domains in which lightweight composition or mashups are relevant.

Even though the Morpho framework is able to provide the basic framework for user profile elicitation and transformation in the mashup environment, its performance depends on the algorithms/tools that are employed. Further evaluation and experimentation of the framework has to be done in order to see to what extend-user profile elicitation and transformation can be done. In addition, the evaluation has to study additional requirements and further extensions of Morpho. Our ongoing research on *User Pipes* also helps us in extending the functionality and portability of this proposed framework. For example, we can observe how WordNet can also be exploited to determine the semantic relatedness between two terms. Consequently, this can be used as an alternative of computing the semantic distance performed by the Interoperability Engine. Similarly, using other kinds of natural language tools in the Concept Extractor can be beneficial.

Acknowledgements: This work was partially supported by the European 7th Framework Program project GRAPPLE ('Generic Responsive Adaptive Personalized Learning Environment').

References

[1] D. Heckmann, T. Schwartz, B. Brandherm, M. Schmitz, and M. von Wilamowitz-Moellendorff. GUMO - The General User Model Ontology. In Proc. of 10th International Conference on User Modeling (UM 2005), Edinburgh, UK, Jul, 2005.

[2] F. Cena and L. Aroyo. A Semantics-Based Dialogue for Interoperability of User-Adaptive Systems in a Ubiquitous Environment. In Proc. of 11th International Conference on User Modeling (UM 2007), Corfu, Greece, Jun 2007.

[3] T. Kuflik. Semantically-Enhanced User Models Mediation: Research Agenda. In Proc. of 5th International Workshop on Ubiquitous User Modeling (UbiqUM'2008), workshop at IUI 2008, Gran Canaria, Spain, Jan, 2008.

[4] A. Kobsa. User Modeling in Dialog Systems: Potentials and Hazards. In AI and Society, 4(3):214-240, 1990.

[5] B. McBride. Jena: A Semantic Web Toolkit. In IEEE Internet Computing, 6(6):55–59, 2002.

[6] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Proc. of International Semantic Web Conference (ISWC 2002), Sardinia, Italy, Jun, 2002.

[7] F. Damerau. A Technique for Computer Detection and Correction of Spelling Errors. In Communications of the ACM, ACM, vol. 7, no. 3 (1964), 171-176.

[8] D. E. Knuth. The Art of Computer Programming Volume 3: Sorting and Searching. Addison-Wesley (1973), 394-395

[9] K. van der Sluijs and G.-J. Houben. Relating User Tags to Ontological Information. Proc. of 5th International Workshop on Ubiquitous User Modeling (UbiqUM'2008), workshop at IUI 2008, Gran Canaria, Spain, Jan, 2008.

[10] K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proc. of HLT-NAACL, 2003, 252-259.

[11] J. R. Finkel, T. Grenager, and C. Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In Proc. of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005), USA, 2005.

[12] J. Lehmann, J. Schüppel, S. Auer. Discovering Unknown Connections – the DBpedia Relationship Finder. In Proc. of 1st Conference on Social Semantic Web, CSSW2007, Leipzig, Sep 24–28, 2007.

[13] C. Stewart, A. Cristea , I. Celik , and H. Ashman. Interoperability between AEH user models. In Proc. of the Joint International Workshop on Adaptivity, Personalization & the Semantic Web, workshop at Hypertext 2006, Odense, Denmark, Aug, 2006.

[14] L. Aroyo, P. Dolog, G.-J. Houben, M. Kravcik, A. Naeve, M. Nilsson, and F. Wild. Interoperability in Personalized Adaptive Learning. Educational Technology & Society, 9(2):14-18, 2006.

[15] Yahoo! Pipes. http://pipes.yahoo.com/pipes/

[16] Mircosoft Popfly. http://www.popfly.com/

[17] Google Mashup Editor. http://editor.googlemashups.com/

[18] R. Ghosh, and M. Dekhil. Mashups for Semantic User Profiles. In Proc. of the 17th International Conference on World Wide Web (WWW 2008), Beijing, China, Apr, 2008.

[19] R. Ghosh, and M. Dekhil. I, me and my phone: identity and personalization using mobile devices, HP Labs Technical Report HPL-2007-184, Nov, 2007.

[20] H. Guo. SOAP: Live Recommendations through Social Agents. In Fifth DELOS Workshop on Filtering and Collaborative Filtering, Budapest, 1997.

Lightweight Data Integration using the WebComposition Data Grid Service

Ralph Sommermeier[1], Andreas Heil[2], Martin Gaedke[1]

[1]Chemnitz University of Technology, Faculty of Computer Science, Distributed and Self-organizing Computer Systems Group, 09107 Chemnitz, Germany
[2]Microsoft Research Cambridge, CB3 0FB Cambridge, United Kingdom
[1]{firstname.lastname}@cs.tu-chemnitz.de, [2]v-aheil@microsoft.com

Abstract. With the advent of Web 2.0, the user becomes a producer creating lots of data by consuming the functionality of the respective Web applications. Even though more and more valuable data is created, it is difficult to reuse it due to lack of structure. In this paper we discuss easing data integration by using the WebComposition Data Grid Service (WebComposition/DGS). Our approach separates technology and information space concepts in a flexible and extendable component model, which yields simplicity for the end user. The model facilitates this by creating, managing and embedding data in different formats and representations to their used applications. Furthermore, machine-readable metadata is implicitly supported and used to link the internal data and external data sources together.

Keywords: WebComposition, Data Grid Service (DGS), Resource Description Framework (RDF), Metadata, Representational State Transfer (REST), Simple Object Access Protocol (SOAP), Service-oriented architecture (SOA)

1 Introduction

A growing number of different data types arise within the scope of Web 2.0 applications, which yield a lot of interesting information. This information becomes even more interesting if multiple data sources are linked together. The power of linked data highlights more intriguing information [1], [2]. The current problem lies in the re-usability of this data. To address this issue, it is required to implement at least one interface for each data source. Obviously, this implementation is time consuming and costly thereby making the linking of different data sources hard to realize. The WebComposition/DGS approach addresses this issue by simplifying writing and reading data as produced or consumed by a Web 2.0 application [3], [4]. This approach enforces concepts of meaningful URIs [5], [6] when creating information spaces by allowing all data to be implicitly addressed by URIs. Beyond that, the WebComposition/DGS natively supports Resource Description Framework (RDF) statements related to these data objects so that they can be annotated with metadata described in a machine-readable format.

In section 2 we examine the state of the art influencing our research. Section 3 shows our approach divided into three subsections. These describe the supported protocols, data formats and data referencing mechanism in the information space concept. In section 4 we discuss our experience with the implementation of components around the WebComposition/DGS to gradually compose and integrate data. Finally, section 5 summarizes our work with a view on future research activities.

2 State of the Art

Many Web 2.0 applications are valuable data silos that mostly provide the corresponding data in very simple formats. This data can usually be accessed for reading by transfer or transport protocols. Often Web 2.0 applications even provide ways for adding and updating data. However, the data is mostly bound within the Web 2.0 application and linking the data in the Web is in most cases very difficult as the data is often not systematically addressable by any URI. In fact, the data produced by its users and held by the Web 2.0 application is its sole asset, distinguishing it from other business rivals. As such, major engineering challenges address the question of how to access data in such silos by using the "best" *protocol* for reading and writing it in a systematic way. In addition, simplicity in "working" with the data, i.e. the *data formats*, and its corresponding metadata is another challenge to be addressed in the context of the Web 2.0 domain and in the context of systematically integrating data.

Protocols. Protocols, within the context of Web 2.0, are usually built on the Hypertext Transfer Protocol (HTTP). They define a set of rules which allow different components of a Web application to communicate with each other. To integrate each other's data, each of the participating components must be capable of understanding the particular protocol and is, as such, limited to the protocols it supports.

The Atom Syndication Format (ATOM) [7], [8] defines a format based on the Extensible Markup Language (XML), using HTTP for publishing and editing data of related resources. ATOM is a well adopted format for aggregating data mainly used by weblogs and wikis providing data through feeds. However, the capability of the ATOM format for writing, modifying and deleting data is barely used. While the format itself is extendable, there is no support for serving multiple representations of a resource. Consequently, the potential consumers integrating data using the ATOM format are forced to support the particular representation.

The Google Data APIs [9] provide simple protocols for reading and writing data on the Web, based on the Really Simple Syndication (RSS) and ATOM formats. The four basic functions *Create*, *Read*, *Update* and *Delete* (CRUD) for working with data [10] are fully supported through an interface using HTTP. Metadata is provided in the form of additional feeds containing referential information using, for example, the Google Base schema. However, this strategy is limited in terms of the fixed semantic information provided by the metadata feeds. This is an issue the Semantic Web [11] aims to solve: data integration and interoperability.

Less data centric protocols include the Simple Object Access Protocol (SOAP) and XML Remote Procedure Calls (RPC). These protocols are not limited to the use of HTTP and can be applied on top of different transport or transfer protocols. They do

not focus on the resource as the primary unit and are often used on a procedural oriented data exchange. By calling business logic SOAP and RPC provide high flexibility in terms of data exchange, however, they are often used in ignorance how the underlying protocols (for example HTTP) work.

While HTTP as a protocol has many advantages, it becomes evident that a solution to the data silos challenge requires not supporting one sole protocol, but as many different protocols as possible.

Data Formats. Plenty of data exists on the Web – data, which could be shared or linked together. However, data in the context of Web 2.0 is mostly under the sole control of the particular Web application and stored in application-specific formats. Limited access to this data or even just subsets of this data is provided only through a small number of restrictive protocols including those described previously. Examples are, the common classical Web 2.0 services for sharing movies (www.youtube.com), pictures (www.flickr.com), private information (www.myspace.com) or private experience and knowledge (www.ciao.com).

Weblogs provide chronologically ordered data, consisting of entries and different views (e.g. by topic) on the data. Wikis provide data with extensive change histories and references to data items that even do not yet exist. Both solutions store their data in platform and vendor specific formats, barely able to exchange. Limited access to the data is often provided using the ATOM or RSS format only. Nevertheless, for writing and modifying the data, the standardized capabilities of these protocols are ignored. Instead, dedicated programming interfaces are offered to access identical functionality. First attempts to establish standardized formats to interchange data between different platforms exist [12] but are not yet recognized by a wider community. The community of DBpedia [13], for example, currently extracts data from various sources, changing this unstructured data into a machine-readable format according to linked data principles [14].

It becomes evident that the simple formats and restrictive mechanisms of these different approaches need to be supported by a solution for integrating as well as annotating this data. As such, a mix of simple data structures and more sophisticated, dedicated data structures, that facilitate reuse and annotation, is needed. We believe that this is a mix of application-specific data structures, usually based on XML, and RDF for annotating the application-specific data.

3 The WebComposition/DGS Information Space

The WebComposition/DGS addresses the proposed approach by adopting a local concept of the global information space. This information space enforces the co-existence of data and corresponding metadata using the abstract components of *containers*, *information stores* and *information items*. Each addressable by a distinct URI to integrate all provided data and metadata within the global information space.

Each WebComposition/DGS is accessible via a dedicated URI that identifies the service as a resource containing so called information stores. These information stores are accessible through nested URIs of the superjacent WebComposition/DGS container. Each information store in turn contains information items which can be addressed by a nested URI and extended by a user-defined path segment. The proposed solution provides the automatic creation of URIs in the information space every time a new information store or item is added. Fig. 1 depicts the composition of these URIs in the information space.

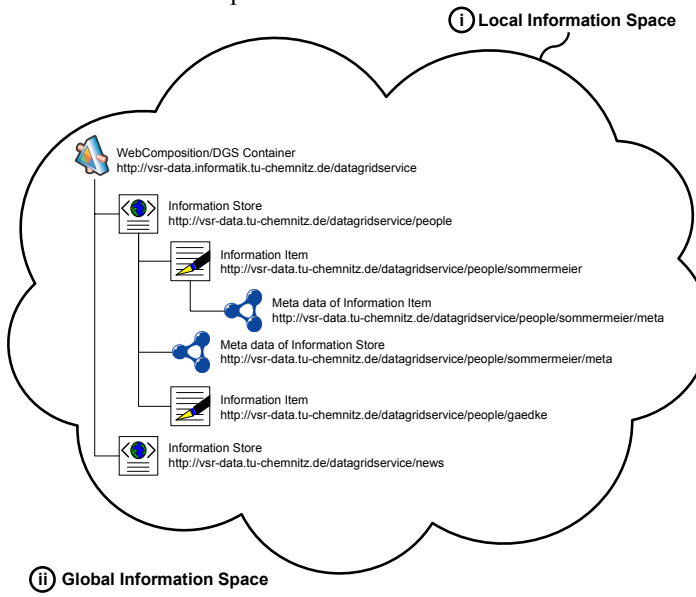


Fig. 1. Information space concepts within the WebComposition/DGS.

For each information store and item a corresponding store for metadata is maintained, which is referred to by extending the information space's URI with the additional path segment */meta*. This metadata describes the information store with all the relevant information semantically describing the data itself. The clear separation of data and metadata allows Web applications to easily access the data using simple protocols and mechanisms. Furthermore, the store for metadata stories is understood as a resource itself and can be addressed by its URI. Accordingly, data or metadata can be requested separately or be combined in terms of linked data [15].

Information Space. Each component within the local information space is addressed by its unique URI. Each sub-path of any particular URI, combined with the given authority, denotes a distinct URI identifying a unique resource within the path hierarchy thereby creating Semantic URIs [16]. Any resource within this information space is identified by its URI and the local information space in Fig. 1 (i) is spanned by one WebComposition/DGS container incorporated into the global information space of the World Wide Web in Fig. 1 (ii).

Container. The WebComposition/DGS service instance provides basic functionality to store, manipulate and easily query resources. The service is understood as a container comprising the functionality to be applied to the enclosed resource.

Information Store. The information store is a logical concept containing a set of related resources. Depending on the applied technology, the information store could be understood as a list, XML file, database or similar. Different implementations of information stores can be hosted within a single container at the same time. It is important to point out that the underlying technology and its evolution are transparent to the consumer of the service and do not affect the data integration.

Information Item. Information items represent the actual resources stored in an information store. Information items could be described in XML, a row in a relational database table, a file or an element out of a list. On a logical level, information items could even contain further information stores.

All components introduced so far outline the fundamentals of an easy data integration process. Besides the standard representation of data using XML, unstructured data, even binary data, is supported in the actual implementation. Enforcing a strict policy of how URIs are generated within the WebComposition/DGS results in any stored information, as well as its corresponding metadata, to be addressed by a dedicated URI. The possibility to access any data and metadata without exception is the fundamental concept that allows us to perform a standardized data integration lifecycle within the WebComposition/DGS.

4 Data Integration Lifecycle

One outstanding engineering challenge to overcome is to simplify handling data used by different types of common protocols in the context of Web 2.0. A data referencing mechanism is required for automatically creating data URIs for each information store, or item, which support the principles of linked data.

Data Referring. As information items are not necessarily bound by any entity (e.g., in form of a file), it is not possible to refer them natively by any URI. Therefore, the WebComposition/DGS provides the capability to create user-defined URIs by applying URI templates [17]. The URI of the information item's superordinate information store is extended with a path segment, which maps to any key that uniquely identifies the item within its native representation. This could be a primary key within a database, a line in a text file or a certain tag within an XML file. Similar to the information store's implementation this mechanism is transparent to the consumer of the service, which solely makes usage of the corresponding URI to integrate the corresponding data.

When using XML as a data format, we can make explicit use of XPath queries to retrieve a particular information item. The XPath query is mapped to the corresponding URI template and saved as metadata for the information store. Fig. 2 depicts the representation of a resource representing a person as XML using a Telnet

session to visualize the integration of the HTTP protocol and different content types. Fig. 2 (1) shows the execution of a HTTP *GET* to an information store accepting the content type *text/xml* request and the resulting XML data.

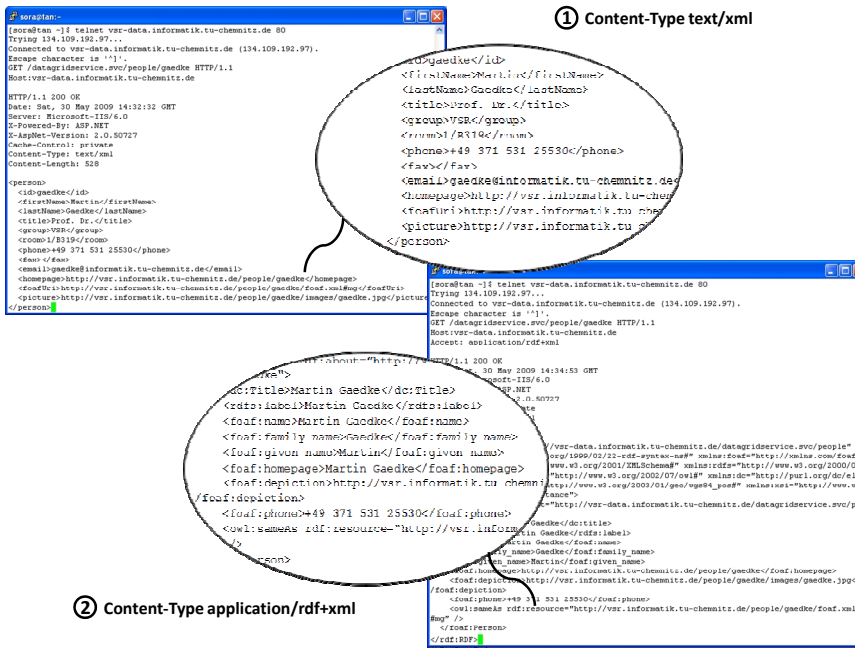


Fig. 2. Data referring using content negotiation.

Fig. 2 (2) shows the execution of a HTTP *GET* request to the same information store. However, here the difference is the accept header of *application/rdf+xml*. The resulting response provides a friend of a friend (FOAF) [18] resource (which is a RDF graph) and contains machine-readable data to be used in terms of linked data. The same result can be retrieved by adding */meta* to the original URI of the request without specifying the accept header. This allows the human user to retrieve the same representation of the data using a convenient mechanism. This mechanism, however, is not restricted to those two content types. Additional components can be implemented and specified to handle further representations of a resource. This characteristic is a fundamental capability to serve as many different data integrators as not all of them are capable of dealing with a single data format (cf. section 2).

Data Integration. The Telnet example above shows the technical realization of the data referencing mechanism using simple HTTP requests. The creation of information stores and information items, however, is not very convenient using these technologies. Data integration is more than simply providing structures of arbitrary data on the Web. Combining data from different sources as well as providing the user with a unified view of this data is an essential part of the data integration lifecycle.

For human use there is still a need for more user-friendly clients. The Telnet way is reasonable for demonstration but not for practical use. A dedicated component within

the WebComposition approach that addresses this issue is the WebComposition/Data Grid Service List Manager (DGSLM). This component, to be used in any Web browser allows creating, modifying or deleting data in the WebComposition/DGS information space. Using this component, we have the possibility to manage our information space via Web browsers without the need for programming. The component is build upon the unified interface of HTTP to read, write, manipulate and delete data. It uses data structures of information stores to dynamically create Web forms based on the metadata of that particular information store. Fig. 3 (a) illustrates the DGSLM displaying a list of information items from an information store.

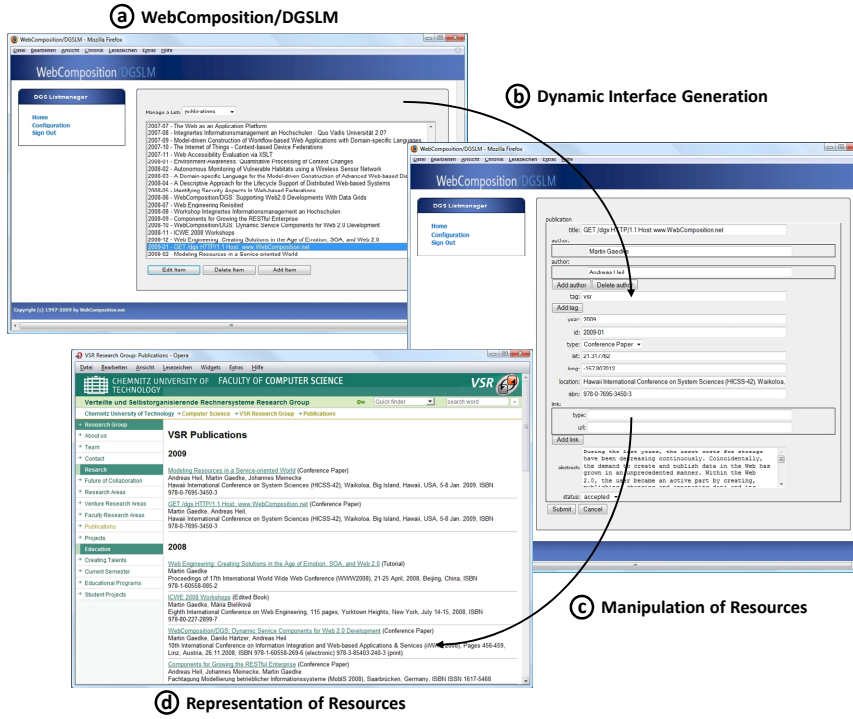


Fig. 3. Dynamic data integration lifecycle.

The corresponding responsibility of the WebComposition/DGSLM, is to provide the data representation independent from the underlying data structures. Data stored in databases, flat XML or binary files can be handled and accessed using a single, simple interface. By requesting a particular information item, a corresponding form is dynamically created Fig. 3 (b) that allows creating or manipulating new or existing information items.

To support the CRUD methods (cf. section 2) of the WebComposition/DGS, the WebComposition/DGSLM offers the complete functionality of HTTP to read, write manipulate and delete data. On behalf of the user the application creates corresponding HTTP request as defined by the endpoint, while the content of the request is dynamically allocated and sent to the WebComposition/DGS. Fig. 3 (c)

depicts representation of the previously created data. Extensible Stylesheet Language (XSL) transformations, also stored within the WebComposition/DGS information store are used to represent the data, in this example at the Website of the Chemnitz University of Technology.

The DGSLM provides the ability of easily managing information stores. It is not limited to a certain information store though. Hence, the WebComposition/DGSLM overcomes the typical difficulty of accessing multiple heterogeneous data sources from within a single Web application.

5. Conclusion and Future Work

For more than nine months, the WebComposition/DGS service is used in a production environment, using real, externally visible data of the research group Distributed and Self-organizing Computer Systems at the Chemnitz University of Technology, Germany. During this time, the data model was gradually exposed and extended with new resources, new representations and new components, according to the emerging needs of the group, and demonstrated the DGS's ability to deal with different representations of the data model for maximum reusability. Some parts of it were transformed from originally unstructured data, mainly managed with Wiki software before. This prior form of managing the data proved to be too hard for integrating and consuming outside of the Wiki itself. Therefore, a WebComposition/DGS implementation was used to successively expose data in accordance to Web standards and the REST principles. Over time, the data of publications, courses, projects, student projects and members of the research group were included. The data typically contains several hundred entries, describing historical and recent data. In addition, the data model was extended several times to accommodate for new information needs, to introduce links between different resources and to add new representations. The approach of encapsulating technologies in components appears to be an important factor for end user support. With the developed system, new data can be created ad-hoc. It is automatically editable in Web forms, without any scripting or code deployment. It can 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 transformations need to be specified when creating new data, the architecture allows adding more user-friendly components that automate this process in the future. The applied components also favor the reusability of the resources by automating the process of generating content representations according to the content negotiation. On the Website, 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 illustrates the system's potential for bottom-up data growth [19]. As demonstrated, components were gradually added to the WebComposition/DGS, while the service was in productive use, i.e. integrated into the group's Website.

Future work includes the development of a publish/subscribe mechanism for every information store to support event driven linked data concepts. Another interesting open issue is the transparent handling of URIs as references to other data entries or machine-readable information on the Web with corresponding user interfaces.

An online example of the WebComposition/DGS and its corresponding downloadable components can be found at http://www.webcomposition.net/dgs. The source code is available via http://www.codeplex.com/webcompostition.

References

1. Linked Data, http://www.w3.org/DesignIssues/LinkedData.html
2. How to Publish Linked Data on the Web, http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/
3. Heil, A. and Gaedke, M.: WebComposition/DGS: Supporting Web2.0 Developments with Data Grids. In IEEE International Conference on Web Services (ICWS), pp. 212-215. IEEE Computer Society, Los Alamitos (2008)
4. What Is Web 2.0, http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
5. Cool URIs don't change, http://www.w3.org/Provider/Style/URI
6. Cool URIs for the Semantic Web, http://www.w3.org/TR/2007/WD-cooluris-20071217/
7. The Atom Publishing Protocol – Requests for Comments: 5023, http://www.ietf.org/rfc/rfc5023.txt
8. The Atom Syndication Format – Requests for Comments: 4287, http://www.ietf.org/rfc/rfc4287.txt
9. Google Data APIs, http://code.google.com/intl/de/apis/gdata/
10. Kilov, H.: From Semantic to Object-oriented Data Modeling. In Proceedings of the First International Conference on Systems Integration, pp. 385-393. IEEE, Piscataway (1990)
11. W3C Semantic Web Activity, http://www.w3.org/2001/sw/
12. Völkel, M. and Oren, E.: Towards a Wiki Interchange Format (WIF) - Opening Semantic Wiki Content and Metadata. In First Workshop on Semantic Wikis (2006)
13. DBpedia, http://www.dbpedia.org/
14. W3C SWEO Linking Open Data Community Project http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
15. Linked Data, http://www.w3.org/DesignIssues/LinkedData.html
16. Sahoo, S.S., et al.: Knowledge Modeling and its Application in Life Sciences: A Tale of two Ontologies. In 15th International Conference on World Wide Web, pp. 317-326. ACM, New York (2005)
17. URI Template, http://tools.ietf.org/id/draft-gregorio-uritemplate-03.txt
18. FOAF Vocabulary Specification 0.91, http://xmlns.com/foaf/spec/
19. Heil, A., Meinecke, J., and Gaedke, M.: Components for Growing the RESTful Enterprise. In Fachtagung Modellierung betrieblicher Informationssysteme, pp. 273-283. Springer, Bonn (2008)

FREDDY: A Web Browser-friendly Lightweight Data-Interchange Method Suitable for Composing Continuous Data Streams

Shohei Yokoyama[1], Isao Kojima[2], and Hiroshi Ishikawa[1]

[1] Shizuoka University, Japan
[2] National Institute of Advanced Industrial Science and Technology, Japan

Abstract. As a remarkable lightweight data-interchange format for use with web browsers, JSON is well known. Recently, web browsers have come to support rich applications called Software as a Service (SaaS) and Cloud Computing. Consequently, data interchange between web servers and web browsers is an important issue. A singleton, an array, or a nested object (tree) can be represented by JSON, which is based on a subset of the JavaScript Programming Language. It is valuable for SaaS applications because JavaScript programs can parse JSON data without the need for special programs. However, web browsers and JSON are poorly designed to address large amounts of data and continuous data streams, e.g. sensing data and real time data. We propose here a novel data format and a data-interchange mechanism named "FREDDY" to address this deficiency. It is not merely a subset of the JavaScript; it can represent semi-structured data, just as JSON does. Moreover, FREDDY is suitable for composing a continuous data stream on web browsers. Using the small JavaScript library of FREDDY, web applications can access streaming data via a SAX-style API; it works on all major browsers. Herein, we explain FREDDY and evaluate the throughput of our implementation. We loaded 400 MByte streaming data using our 5 kByte library of FREDDY.

1 Introduction

Once it was believed that web browsers were useful merely to display static pages that web servers would provide one after another. However web pages are no longer static, as exemplified by Google Maps, which uses dynamic HTML and Asynchronous JavaScript + XML (Ajax)[15]. Using JavaScript, web pages can access Web servers, download data, and update pages themselves. This technique has laid the foundations for next-generation web applications such as SaaS[16], Web2.0[13], and cloud computing. In this case, the main logic of applications is partially located on a client side and partially located on a server side. We will address implementations of web applications with the question of how data that applications need can be accommodated. JavaScript applications on web browsers always run inside a security sandbox. Consequently, all data must be either on primary storage (main memory) of a local computer or on secondary

storage (hard disks) of web servers, which is not of the local computer. That is to say, data interchange between a web server and a web browser is an extremely important issue for web applications.

Ajax and JavaScript Object Notation (JSON: RFC4627)[5] are attractive solutions for interchange of data between web servers and web browsers. However, the combination of Ajax and JSON can download only a chunk of data simultaneously. They cannot handle continuous data streams, e.g. sensing data and real time data. Numerous sensors are installed in devices from toilets to satellites, but no lightweight integration method exists for handling sensing data on the Web.

The purpose of this paper is to propose a lightweight data stream interchange mechanism named FREDDY. Our implementation provides a *Simple API for XML (SAX)*[3] style programming. Therefore, FREDDY can accommodate not only a continuous data stream, but also semi-structured data, equivalently to XML. Using FREDDY, users can accommodate continuous data streams via a SAX event handler, which is written in JavaScript.

In this study, we also evaluate the throughput of our implementation of FREDDY when the application loads a large SAX event stream. The results of experiments show good throughput, about 1 MByte/s, of data interchange between a web server and a web browser. However, we do not emphasize the velocity of data interchange; also, FREDDY is a lightweight method in terms of the security sandbox of web browsers. The remainder of this paper is organized as follows. Section 2 expresses an overview of FREDDY. The data format and our implementations of FREDDY are described in Section 3 and Section 4. In Section 5, we present our experiments and evaluations. In Section 6, we describe related works. Finally, Section 7 concludes the paper.

2 Overview of FREDDY

The main contributions (Fig. 1) of this paper are: (a) a lightweight data format that is suitable for streaming data exchange using only JavaScript, (b) a streaming delivery mechanism on the Web, (c) a lightweight library, whose size is about 5 kByte, to realize SAX-style programming for handling both semi-structured data and continuous data streams.

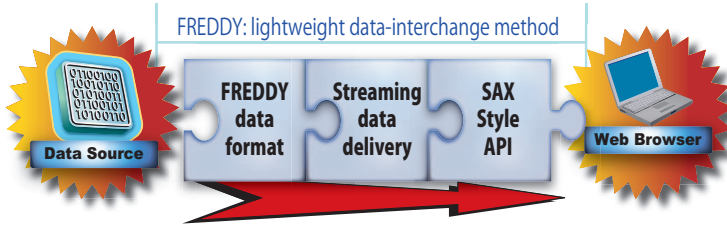


Fig. 1. Software components and data flow of FREDDY and JSON.

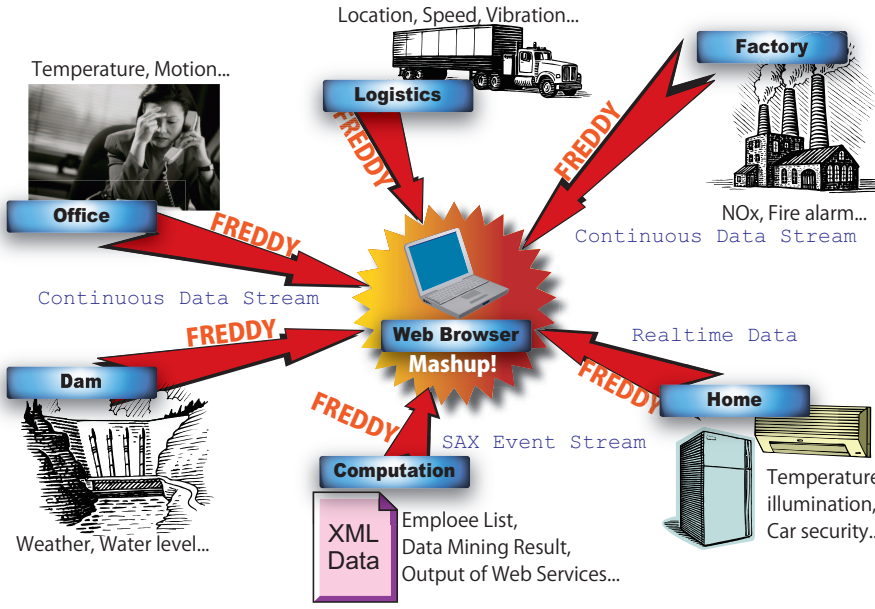


Fig. 2. Our Goal, Stream Data Mashups using FREDDY.

Figure 2 presents the goal of our research. FREDDY provides a lightweight JavaScript API that is equivalent to the SAX API of XML document processing. In fact, SAX API is the de-facto standard API. For that reason, we expect that many users have sufficient knowledge of SAX. Using the proposed FREDDY, users can easily develop Web mashups that compose web services to output continuous data streams. Because space is limited, we have concentrated on data interchange and have devoted scant attention to how to translate the output signal of a sensor into our proposed data format.

3 Data model

3.1 Outline of Data Format

The data format for FREDDY, FREDDY Format, can represent both a flat data stream and a semi-structured data stream. The FREDDY format is simple. Figure 3 portrays instances of the FREDDY format as an XML tree and a data stream representation. What the right of that figure readily clarifies is that the FREDDY format uses function calls written in JavaScript. Each line of FREDDY data expresses a type of SAX event and its property. We named each function call *event container* a generic name.

The main characteristic of the FREDDY Format, contrasted against that of JSON, is that it is *splittable* because it is a simple repetition of event containers. Actually, FREDDY realizes streaming between web servers and web browsers

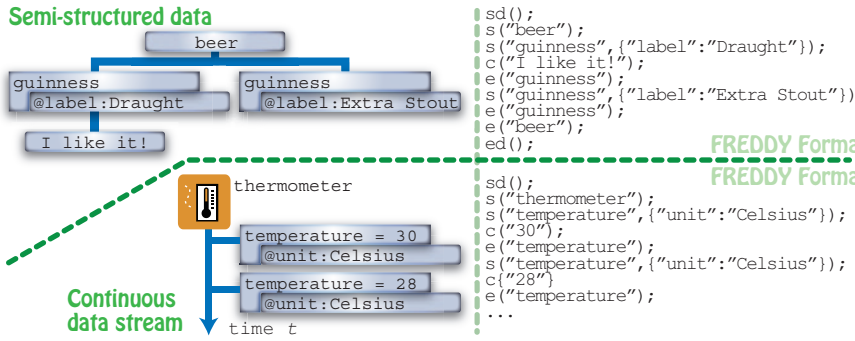


Fig. 3. Examples of semi-structured data representation: the four expressions are mutually equivalent.

Table 1. Event containers

SAX event	Usage	Event container
XML Document start	Data stream start	`ds();`
XML Document end	Data stream end	`ed();`
XML Text Node	Property	`c(`*value*`);`
XML Element start	Tuple start	`s(`*element-name* `,` *attr*†`);`
XML Element start‡	Tuple start‡	`S(`*simplified-name*`,` *attr*†`);`
XML Element end	Tuple end	`e(`*element-name*†`);`

†: optional argument
‡: with simplified element name

by sending and receiving small fragments of all data one by one. Later in this paper, a more precise account of the split FREDDY format is provided. We now address the event container in detail.

3.2 Event container

Table 1 portrays a list of all event containers. The events of the XML Document start and XML Document end are represented as `ds()` and `de()`. The data stream must start with a `ds()` event container and end with a `de()` event container. The two containers must appear only once.

The second argument of `c(...)`, which represents an XML Text node, is optional. It is always omitted from the representation of XML Tree because the XML Text Node has only a value.

The events of an XML Element start and end are represented as `s(...)`, `S(...)`, and `e(...)`. The argument of `e(...)` is optional. The element name is associated with an XML Element start event, which corresponds to that if no argument is given. The reason is data size reduction. For the same reason, the second argument of `s(...)` and `S(...)`, which represents XML Element Attributes, is optional.

3.3 Compression of verbose elements' name

Actually, XML is known to be verbose by design[12], particularly in terms of elements that appear many times. The SAX event stream has the same problem. For example, bibliographic information of Digital Bibliography and Library Project (DBLP)[1] has about 2,300,000 <author> tags, about 600,000 <inproceedings> tags, and about 350,000 <journal> tags.

Efficient SAX event stream interchange must tackle that data verbosity. In this context, XML compression is an important issue. In addition, XML compression is our concern: we proposed XML compression according to the simplified element name[7, 17]. The FREDDY Format tackles XML verbosity using the method of simplified element names. The reason that two event containers exist for the XML Element start event is data size reduction.

See Fig. 3. Two <Guinness> tags and two <temperature> tags exist. If the element name is <a> instead of <Guinness> and <temperature> then the amounts of data become small. This is the conceptual foundation of the compression method.

Algorithm 1 shows an algorithm for creation of an XML Element start event container from the element name. Whenever the parser captures the element start event, this procedure is called, where input T is a stack of visited element names and argument *name* is an element name. The procedure then returns a simplified element name derived from the original element name.

Algorithm 1: SimplifiedFREDDY($T, name$)

procedure GetSimpleName(idx)
  $X \leftarrow [a, b \ldots y, z, A, B \ldots Y, Z, 0, 1 \ldots 8, 9]$
  $len \leftarrow$ LengthOf(X)
  if $len \leq idx$
    then
      $y \leftarrow X[idx\%len]$
      $z \leftarrow idx/len$
      return (GetSimpleName(z) + y)
      comment: + means connection
    else return ($X[idx]$)

main
  if $T[name]$.IsExist()
    then
      $evtContainer \leftarrow$ "$S('" + sName + "')$"
      return ($T, evtContainer$)
    else
      comment: First appearance of the element
      $idx \leftarrow$ LengthOf(T)
      $sName \leftarrow$ GetSimpleName(idx)
      $T[name] \leftarrow sName$
      $evtContainer \leftarrow$ "$s('" + name + "')$"
      return ($T, evtContainer$)

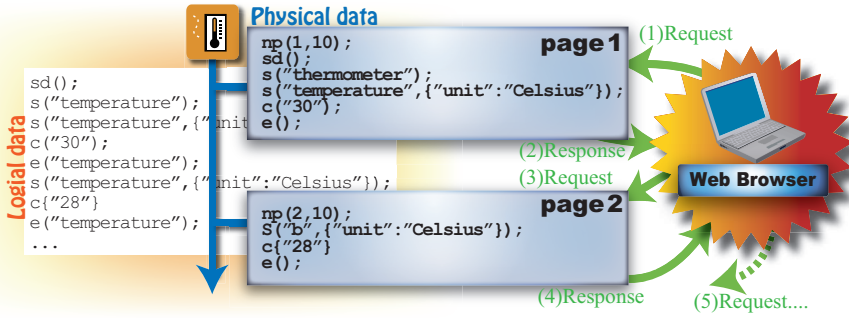


Fig. 4. Split of FREDDY data.

`GetSimpleName(...)` function generates the simple name for each new incoming element name. The original element name will be replaced with the simple name whenever this element re-appears later in this XML document. The original element name itself is kept within the XML document by not replacing the first entry of this element name.

3.4 Splitting data streams into small fragments

A salient difference between FREDDY and JSON is that FREDDY data can be split by line into valid fragments, which maintains a subset of the JavaScript programming language. All lines of FREDDY data are actually a subset of JavaScript.

Switching our attention to the continuous data stream, the data sources (e.g. thermometer and NOx sensor, etc.) always output data continuously, but HTTP, which enables any Web browser to communicate with any Web server, cannot handle continuous data streams. Therefore, FREDDY can split continuous data streams into small fragments, named Pages, as physical data.

Figure 4 presents an example of split data. The function-call `np(`*`pointer, sleep-time`*`);` in the first line of each Page is a pointer to the next Page. FREDDY can start downloading the subsequent page when the function `np` is called. The Pages are valid JavaScript code. For that reason, the web browsers can download them dynamically using HTTP.

Actually, FREDDY uses the dynamic <script> tag technique for downloading Pages. The dynamic <script> tag technique is a kind of Ajax based on Dynamic HTML. JavaScript applications can append HTML elements into the DOM tree of the HTML page. For example, if <img> tag with the src attribute is appended directly to the inside of the <body> tag of the DOM tree, then the image that the src attribute refers to is readily apparent. Similarly, using the dynamic <script> tag, one can access all HTTP-enabled resources.

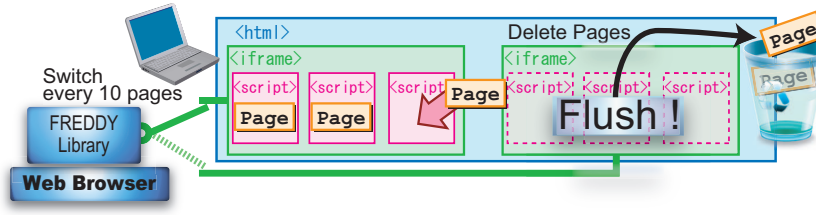


Fig. 5. Our client implementation.

4 Implementation

4.1 Dynamic <script> tag

In the following section, we describe our implementation for data interchange using FREDDY. The SAX-style programming makes only one pass through the document from top to bottom. For that reason, FREDDY deletes Pages after execution because they become unnecessary. An important problem was that web browsers did not release the <script> element from memory even when FREDDY deleted the element from the DOM tree. For that reason, our implementation adopts a hybrid of <iframe> and <script> because the web browser released the tags from the memory of the client PC when FREDDY flushed the <iframe> element.

Because of the hybrid implementation, FREDDY achieves both lower processing costs and higher throughput. Our client implementation is portrayed in Fig. 5. Actually, FREDDY uses two <iframe> elements alternately. FREDDY appends <script>, which includes a Page, as a child node of one <iframe> element at the beginning. It flushes the <iframe> and switches the other <iframe> area if the number of Pages containing the <iframe> area reaches 10. Two <iframe> are used because of the synchronism of the dynamic <script> tag technique. That is to say, when Pages are appended into a <iframe>, the other <iframe> element is flushed; then Pages are released from memory simultaneously.

4.2 Streaming delivery system

Figure 6 shows how web applications load a data stream over HTTP protocol. The data interchange between data source and web application consists of the following four steps:

1. Raw data are translated into Pages of FREDDY format by the gateway specializing in each data source.
2. FREDDY requests to the data source and receives Pages one by one using the dynamic <script> tag technique.
3. The downloaded page is executed using a JavaScript engine of a web browser.

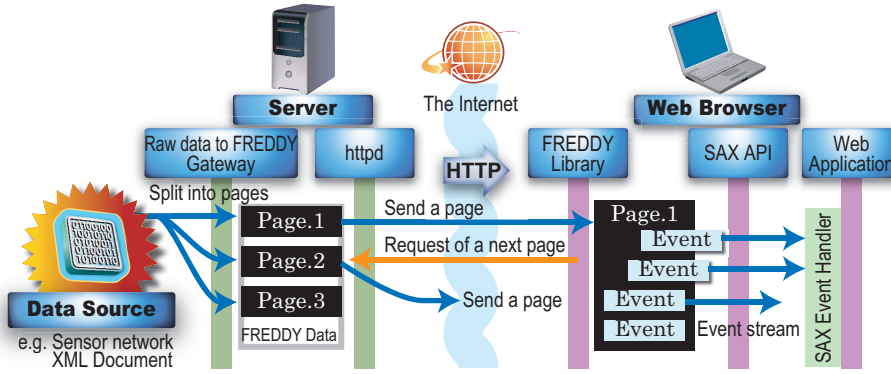


Fig. 6. Continuous Data Stream over HTTP.

4. The SAX event is noticed to the user defined event handler. If the function `np(...)` is called, then FREDDY requests download of the following Page.

It seems complex, but, in fact, it is very simple for users because the steps above are hidden by our implementation. Therefore, the web application can receive a continuous data stream easily via a SAX event handler. Furthermore, the size of the FREDDY library, which is written in JavaScript, is only 5 kByte because it is sufficiently lightweight to include into web applications.

4.3 JavaScript SAX API of FREDDY

Next, describing the usage of FREDDY, the FREDDY streaming delivery system is executed behind the SAX API, so that the SAX API is the only interface for FREDDY.

The usage of JavaScript SAX API has three steps. The first is creating methods of the SAX event handler. The name of the method is the same as that of the methods of Java DefaultHandler class. The next is creating instances of both the SAX Parser and event handler and setting the handler to the parser. Finally, execute and start parsing document.

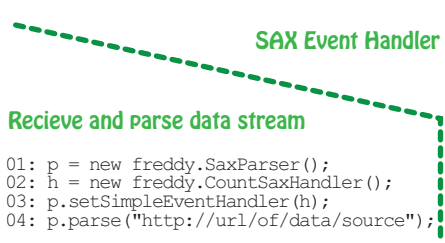
For example, if the events of a documents are counted, then an event handler can be created, as portrayed in Fig. 7 left. The right part of Fig. 7 shows the code to count up the events of the data.

This is a general procedure related to all SAX Parsers, so that FREDDY is not only accessible by web programmers; it is also easily applicable by XML programmers.

5 Experiments and results

5.1 Dataset and environment

For experiments, we used large-scale data of four XML documents up to 400 MByte. The three small XML documents *dataS.xml*, *dataM.xml*, and *dataL.xml*




```
01: CountEventHandler.prototype = {
02:   count : 0,
03:   startElement : function(name,attr){
04:     this.count++;
05:   },
06:   endElement : function(name){
07:     this.count++;
08:   },
09:   Characters : function(data){
10:     this.count++;
11:   }
12: };
```

Fig. 7. JavaScript code for using FREDDY.

Table 2. Machine environment

	Web Server	Client A	Client B
Hardware		IBM ThinkPad X41 Tablet	DELL Precision 390
CPU	Intel Xeon 2.33 GHz	Pentium M 1.6 GHz	Core2 Duo 2.4 GHz
Memory	4GB	1GB	2GB
OS	Linux (Fedora Core 7)	Windows XP	Windows Vista
HTTPD	Apache 2.2.4		

are 1 MByte, 10 MByte, and 100 MByte files created using the `xmlgen` from the XMark benchmark project[14]; the biggest XML document is a 400 MByte DBLP bibliographic information document.

The computers used for the experiments are described in Table 2. We used two different computers to estimate performance: a desktop computer (*Client A*) and a laptop computer (*Client B*). The client machines and the server machine are connected via a Giga-bit Ethernet network.

Regarding the case in which FREDDY is used as an intermediate form for XML handling with a web browser, the system has the following four tiers: (1) a client machine on which the web browser is running, (2) a web server which hosts web applications, (3) an SAX event stream-to-FREDDY gateway server, and (4) a web server which holds XML documents. However, we specifically address the interchange of FREDDY data between a server and a client. Therefore, the three servers described above are located on the same server.

5.2 FREDDY vs. JSON

Actually, JSON is well known as a browser-friendly, lightweight data-interchange format. As described earlier, JSON has a simple structure and great power of expression, but it has limited scalability. For large amounts of data, it is impractical to store all data into main memory using JSON. For this reason, we propose a novel data-interchange method, FREDDY, which is suitable for use with large amounts of data. We have performed measurements of FREDDY for comparison with JSON.

In this experiment, we measured the execution time for loading the whole XML document of each *dataS.xml*, *dataM.xml*, *dataL.xml*, and *dblp.xml* using

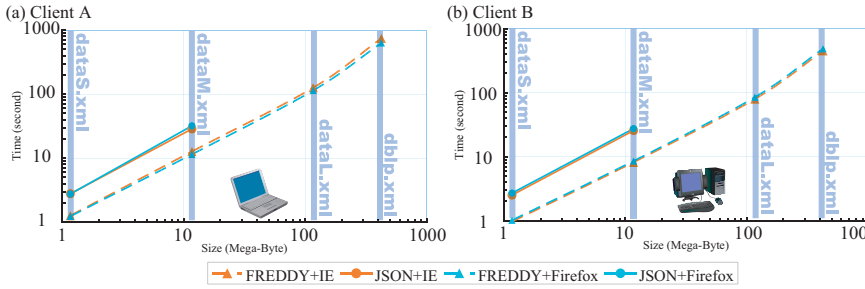


Fig. 8. Execution time of FREDDY and JSON.

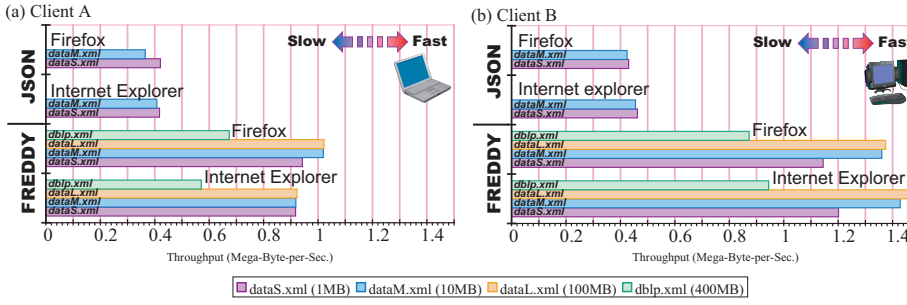


Fig. 9. Throughput (MByte/s) of FREDDY and JSON.

FREDDY and JSON on both Internet Explorer (Microsoft Corp.) and Mozilla Firefox of the client machines: *Client A* and the *Client B*. We used an XML-to-JSON gateway, which is implemented using IBM[11]. We measured all cases, which are combinations of 2 clients, 2 browsers, 2 systems, and 4 datasets, 10 times each. In this case, JSON was unable to load *dataL.xml* and *dblp.xml* because the server had exhausted the allowed memory size.

Results of this experiment are presented in Fig. 8 and Fig. 9. Figure 8 shows the average execution time of FREDDY and JSON. The throughput, the amount of loaded data per second, is presented in Fig. 9. The results of the experiment are that, irrespective of data size, the throughput of FREDDY is greater than that of JSON. We shall next examine the results more carefully.

– **Internet Explorer versus Firefox.**
  We can find no significant difference of loading times between Internet Explorer (Microsoft Corp.) and Mozilla Firefox, but Internet Explorer (Microsoft Corp.) sometimes failed to load Pages. Therefore, we developed a retransmission mechanism for FREDDY.
– **Client A versus Client B.**
  The result of FREDDY shows that *Client B* is faster than *Client A*. However, regarding the result of JSON, we can find no obvious difference between *Client A* and *Client B*. The result suggests that FREDDY is bottlenecked

by CPU power. On the other hand, we infer that the result of JSON is not influenced by client-machine specifications.

– **FREDDY versus JSON.**
Because JSON must store all data into the main memory, it is difficult for JSON to handle a large amount of data. The result also shows that JSON was unable to load datasets of 100 MByte and 400 MByte.
Figure 9 presents that FREDDY is faster than JSON.
We find that FREDDY has a feature resembling that of SAX in the context of XML processing.

6 Related Works

The proposed FREDDY splits large data into small fragments for data interchange. Indeed, splitting large data into small fragments is a common solution to the problem of data interchange. Nevertheless, some problems persist in implementation of data interchange on the web browser because a JavaScript web application program must always run inside of a security sandbox. Applying common problems of information technology to the web application domain is a recent trend of research.

Klein and Spector proposed distributed computation of genetic programming via Ajax[9]. In the context of data interchange, Huynh et al. proposed sophisticated user interfaces for publishing structured data on the Web[6], but that method merely addresses the contents' presentation. It includes no solution for large documents.

The target of comparison in this study, JSON, has spread quickly on the Internet. The W3C proposed an application for semantic web for representation of SPARQL query results[4]. The JSON-RPC[2] is a lightweight remote procedure call protocol, which resembles XML-RPC. Results of several studies suggest it as a future direction of FREDDY development.

Natarajan describes an innovative transport layer protocol for data interchange on the Web[10]. However, to the best of our knowledge, no method exists for the application layer. FREDDY is an application layer method. Consequently, users use FREDDY in an existing HTTP and TCP/IP environment.

An extremely important issue related to JavaScript applications is security management. Jackson and Wang tackle the security of cross-domain scripting[8]. We also devote attention to the security of web applications, but a more comprehensive study of security is beyond the scope of this paper.

7 Conclusions

As described herein, we have presented FREDDY, a browser-friendly lightweight data-interchange method that layers efficient data stream interchange between a web server and a web browser. We also proposed an SAX style API to load a continuous data stream. Results of our experiments show that FREDDY can be a good solution for data interchange on the Web.

The future direction of this research will be one that encompasses data sources. We seek to focus attention how to translate raw data into FREDDY format. We plan to extend the design to an infrastructure of a web mashup that can communicate between the Web and a sensor network. We believe that JSON and FREDDY can serve as a basis for data interchange for web applications.

References

1. Digital bibliography and library project (dblp). `http://dblp.uni-trier.de/`.
2. Json-rpc. `http://json-rpc.org/`.
3. Sax. `http://sax.sourceforge.net/`.
4. K. G. Clark, L. Feigenbaum, and E. Torres. Serializing sparql query results in json. `http://web5.w3.org/TR/2007/NOTE-rdf-sparql-json-res-20070618/`.
5. D. Crockford. Introducing json. `http://json.org/`.
6. D. F. Huynh, D. R. Karger, and R. C. Miller. Exhibit: lightweight structured data publishing. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 737–746, New York, NY, USA, 2007. ACM Press.
7. H. Ishikawa, S. Yokoyama, S. Isshiki, and M. Ohta. Project xanadu: Xml- and active-database-unified approach to distributed e-commerce. In *DEXA '01: Proceedings of the 12th International Workshop on Database and Expert Systems Applications*, pages 833–837, Washington, DC, USA, 2001. IEEE Computer Society.
8. C. Jackson and H. J. Wang. Subspace: secure cross-domain communication for web mashups. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 611–620, New York, NY, USA, 2007. ACM Press.
9. J. Klein and L. Spector. Unwitting distributed genetic programming via asynchronous javascript and xml. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1628–1635, New York, NY, USA, 2007. ACM Press.
10. P. Natarajan, J. R. Iyengar, P. D. Amer, and R. Stewart. Sctp: an innovative transport layer protocol for the web. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 615–624, New York, NY, USA, 2006. ACM Press.
11. S. Nathan, E. J. Pring, and J. Morar. Convert xml to json in php. `http://www.ibm.com/developerworks/xml/library/x-xml2jsonphp/`.
12. W. Ng, W.-Y. Lam, and J. Cheng. Comparative analysis of xml compression technologies. *World Wide Web*, 9(1):5–33, 2006.
13. T. O'Reilly. What is web 2.0. `http://www.oreilly.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20%.html`.
14. A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. Xmark: a benchmark for xml data management. In *VLDB'01: Proceedings of the International Conference on Very Large Data Bases*, pages 974–985, Hong Kong, China, 2001.
15. Wikipedia, the free encyclopedia. Ajax. `http://en.wikipedia.org/wiki/Ajax_%28programming%29`.
16. Wikipedia, the free encyclopedia. Software as a service. `http://en.wikipedia.org/wiki/Software_as_a_service`.
17. S. Yokoyama, M. Ohta, and H. Ishikawa. An xml compressor by simplified element name (in japanese). In *Proceedings of DBWeb2000*, 2000.

Introducing collaborative Service Mashup design

Martin Vasko and Schahram Dustdar
{vasko|dustdar@infosys.tuwien.ac.at}

Vienna University of Technology
Institute of Information Systems
Distributed Systems Group
Argentinierstreet 8, 1040 Vienna, Austria

Abstract. The adoption of REST - an architectural paradigm focusing on resources - by various providers like Google, Facebook and Yahoo strongly influences traditional Business Process design approaches layered atop of Web service description language (WSDL) and SOAP based Web services. Beside the architectural differences manifested in integration challenges for existing business process environments this new paradigm eases service development and enables lightweight integration in Web-oriented architectures. This paper introduces a model-driven approach to integrate different domains into Service Mashups: a) Orchestration information derived from WS-BPEL processes, b) Coequal integration of RESTful Web services and WS-* Web services and c) Role-based collaboration of process participants. The introduced concepts are implemented as a platform to enable collaborative Service Mashup design. The prototype is realized as a Rich Internet Application to maximize design performance and user experience.

1 Introduction

The emergence of Web services adhering to the REST (Representational State Transfer) paradigm - referred as RESTful Web services - and the widespread adoption and dispersion by different providers[1] strongly influences traditional business process environments. The benefit of the exposed services for individual business needs evolved over time from general services like for example the Google Search to specialized services like access to Social Network portals - refer to the Facebook services[2] as an example - or commercial services like Amazon's Web services. Beside the increasing number of services the ease of integration and the flexibility to changes awakes the interest to integrate these services into existing business process management approaches. The architectural differences between established Service Oriented Architectures and newly emerging lightweight resource oriented architectures complicate this endeavor.

[1] The programmable web - a directory of Web APIs, http://www.programmableweb.com, last accessed on April 2009
[2] http://developers.facebook.com, last accessed on April 2009

Different approaches [1], [2] try to solve the integration hurdles from the WS-BPEL (Web Service Business Process Execution language[15]) perspective (Pautasso [1]) whereas others provide an abstract simplified language to enable a unique service integration (Rosenberg et al. [2]). The latter is closely related to the approach introduced in this work. Whereas we do not try to provide an alternative language we encourage the model-driven approach to maintain extensibility. The abstraction of orchestration information, service integration information and participant integration information results into a simplified process abstraction. The orchestration information is derived from business processes defined in WS-BPEL. The service integration information is derived from WSDL (Web Service Description Language [16]) based Web services and RESTful Web services. Due to the lack of standards for human participant integration this work derives the role model information from BPEL4People (WS-BPEL Extension for People [14]). This enables a structured collaboration on Service Mashups. Beside the generic role model of human participants, newly emerging RESTful Web services enable the programmatic access to Social Network platforms and thus intensify the interactions with human participants. Currently different providers work on the definition of a set of functions summarizing a common access to Social Networks[3]. These trends facilitate the combination of such services with existing services to Mashups. As a consequence, not even the designers of Service Mashups might have access to collaborative Service Mashup design tools through Social Networks but also the Mashups themselves comprise Social Network Services as part of the Service Orchestrations. The underlying concepts of these correlations are described in Section 3. Existing platforms to create and administrate Mashups (like Google Mashup Editor, Intel Mash Maker, Microsoft Popfly and Yahoo Pipes to name the most prominent ones) currently provide limited support for collaborative Mashup design. Our work introduces a model-driven approach to collaboratively design Service Mashups. We developed a Rich Internet Application prototype to exemplify the introduced concepts and propose an approach to model Service Mashups.

The work is organized as follows: Section 1.1 tries to clarify the definition of Service Mashups by providing existing definitions and relating akin appellations like Mashups and Business processes. Section 2 summarizes existing approaches and relates them to the approach introduced in this paper. Section 3 introduces collaborative service orchestration based on a generic role model derived from BPEL4People, a established specification in Web service environments. Section 4 motivates the need for collaborative Service Mashup design approaches on the basis of a well-known process scenario. The combination of orchestration information, service integration information and collaboration information resulted in the Service Mashup Abstraction described in Section 5. This abstraction is realized in a framework which is implemented as a Web-based prototype and deploys a Rich Internet Application. Finally Section 6 concludes the paper and gives an outlook of the Future Work.

[3] `http://www.opensocial.org`, last accessed on April 2009

1.1 Service Mashups

The increasing number and diversity of RESTful Web services exposed on the internet blurs an adequate classification of orchestration paradigms. In the domain of Web applications lightweight integration approaches - summarized as RESTful Web services - dominate the service infrastructure. In the domain of Enterprise Computing the "Big" Web service technology stack (SOAP, WSDL, WS-* specifications, WSBPEL, etc.) delivers interoperability for heterogeneous service infrastructures. The architectural differences of these two worlds result in challenging combination efforts (outlined by Pautasso et al. [3]). From the orchestrational point of view both provide interesting techniques: WSBPEL is an XML based orchestration language to formulate business processes in Web service environments. The orchestration of services is achieved by a set of activities providing simple and complex Web service interaction capabilities. In contrast to this specification Mashups - refered as combinations of Web API calls - currently lack a comparable notation. Mashups indicate a way to create new Web applications by combining existing Web resources and Web APIs. According to [4] Service Mashups combine WS-BPEL based orchestrations with RESTful Web services. This work adheres to this definition and extends the idea of integrating new paradigms by the use of a model-driven approach.

2 Related Work

Hoyer et al. [5] sketch design principles for emerging Enterprise Mashups. The authors identify several shortcomings in established implementations of Service Oriented Architectures due to: a) high technical complexity of the relevant standards b) their inflexibility to react on changing requirements quickly and c) missing involvement of actual end-users. By allowing end-users to compose collections of services according to their individual needs, Mashups empower end-users to create their own workspaces which fit best to solve their heterogeneous business problems.

Swashup DSL introduced by Maximilien et al. [6] provides a domain-specific language to represent Mashups. The proposed language is focused on the description and propagation of data in Mashups. The approach is implemented by the use of the Rails framework and identifies three main concepts of mashups: data and mediatiation, service APIs and a means to generate Web applications from the resulting mashups. Curbera et al. [7] introduce Bite, a workflow-based composition model for Web applications. Bite allows the definition of interactive, asynchronous workflows with multiparty interactions and enables comprehensive data integration.

HOBBES [8] deploys an Adobe Flex - based WSBPEL designer and enables the collaborative modeling and administration of Business processes. The approach implements a Rich Internet Application to design and share WSBPEL processes. In contrast to HOBBES the approach introduced in our work focuses on the collaborative design of Service Mashups.

Tran et al. [10] introduce a novel approach to integrate existing process modeling languages at different abstraction levels using the concept of an architectural view. Their approach overcomes the divergence in term of syntax, semantics and levels of abstraction of existing process modeling approaches. Our work derives the concept of integrating different modeling languages at different abstraction levels using the concept architectural view and applies this idea on the domains of orchestration information, service integration information and collaboration integration.

3 Collaborative Service Orchestration

With the increasing number of services exposed by different providers on the internet the combination of these services to Service Mashups gains popularity. The broadening of functionality enables the recombination of Services to advanced Service Mashups orchestrating services from different domains. This trend leads to increasing specialization of Mashups and requires know-how from different domains: For example HousingMaps.com[4] combines data from craigslist.org[5] with Google Maps[6]. This Mashup requires knowledge in the domain of real estate markets and web application development. The trend to interdisciplinary combination of Services is encouraged by the effort to coequal integration of RESTful Web services and WS-* Web services as exemplified in the sample process scenario in the next section. This evolution of process environments from closed company-intern systems to open Web service environments crossing organizational boundaries requires the collaboration of different experts on the design of Service Mashups. Existing platforms to administrate Mashups currently lack support for this endeavor. Even simple role models are not supported.

The importance of role models in collaborative environments was depicted by Ellis et al. [9]. In the domain of Service Oriented Architectures BPEL4People[7], a specification proposed by IBM and SAP, shape up as an effort to integrate and structure human participants based on roles in WS-BPEL based Business processes. The specification defines a generic role model consisting of three human roles:

- The *process initiator* triggers the process instance at its creation time
- *Process stakeholders* can influence the progress of a process instance, for example, by adding ad-hoc attachments
- *Business administrators* are allowed to perform administrative actions on the business process, such as resolving missed deadlines.

The role model covers the whole business process lifecycle from design time to runtime issues. To continue the combination of Service Oriented Architecture principles with REST based web application paradigms the approach presented

[4] `http://www.housingmaps.com/`, last accessed on April 2009
[5] `http://www.craigslist.org`, last accessed on April 2009
[6] `http://maps.google.com`, last accessed on April 2009
[7] WS-BPEL Extension for People, BPEL4People

in this paper introduces a role model derived from the BPEL4People roles. In contrast to BPEL4People the role model introduced in this paper covers design time issues only as the integration of runtime issues is part of the future work described in section 6. The derived role model consists of the following roles:

- *Creator* - the *Creator* is determined automatically by the infrastructure during design time and refers to the participant creating the initial Service Mashup design
- *Stakeholder* - the *Stakeholder* may influence the progress of the Mashup design by adding attachments, process notes or forwarding tasks but has no privileges to change the Mashup design
- *Administrator* - the *Administrator* is allowed to perform administrative actions on the Mashup design. In addition to the privileges granted to *Stakeholders Administrators* are able to change the Service Mashup design

Beside the role model the visibility and propagation of changes to the instance edited by different members is crucial in collaborative service orchestration. In contrast to real-time collaboration systems enabling the concurrent editing and session sharing between members the approach introduced in this work is realized regarding relaxed WYSIWIS (What-You-See-Is-What-I-See). A Service Mashup instance is locked exclusively by the member editing the instance. After propagating all changes to the server the instance is released and the involved members are able to check changes by loading the instance. This roundtrip approach adheres to the REST paradigms as Service Mashups are exposed as resources by the server. By updating the changes of the process model only, the introduced architecture minimizes server processing load. The disadvantages of late propagation of changes is compensated by full access to the Mashup design by the processing member.

4 Process Scenario

The coequal integration of RESTful Web services and WS-* Web services into established business process environments neglects the different underlying architectures of these two concepts. The growing interest and the low integration hurdles of RESTful Web services result in the dispersion into non-technical domains. This trend amplifies the coequal integration of mostly publicly available RESTful Web services and existing company-internal WS-* Web services into daily business processes.

To exemplify the previously stated assumptions we introduce a sample process scenario illustrated in figure 1. The process executes a revisited example of a travel agency using existing Web 2.0 services: A user submits a holiday request through the Web frontend containing details of the start date, the end date and the destination. This order is submitted automatically to the agency-internal Order Processing Web service to track incoming orders. After the successful completion of this Web service operation the order is assigned to a responsible travel agent by the agency-internal HR Service. This service administrates all

travel agents and automatically assigns the appropriate travel agent to the user request on the basis of the agent's expertise. The assigned travel agent prepares the user order. This is modeled in a separate process administrated by the travel agent. The travel agent searches for the best photos of the destination, plans on-site trips and searches for the cheapest hotels. After the travel agent arranged the on-site trip details and ordered them he assembles all details into the resulting trip. This step concludes the Process Order under his responsibility. He propagates the package containing the order details to the main process. The Order Manager responsible for the main process publishes the trip and responds to the user request.

The Web Application Expert is responsible for the appropriate execution of the process. He maps the service requirements defined by the travel agent and the Order Manager to available Web services. As already mentioned the task *Assemble Order* is performed by the Order Service. The assignment of the order request to an adequate travel agent is done by the HR service. Both services are company-internal WS-* Web services hosted on the Travel Agency servers. To enable the search for the best photos of the destination the Web Application expert decides to integrate the Flickr Web service. To plan on-site trips and calculate the distances he decides to use the Google Maps Web service. The search for the cheapest hotels is enabled by the Hotels Combined Web service. All of these services expose their APIs as RESTful Web services.

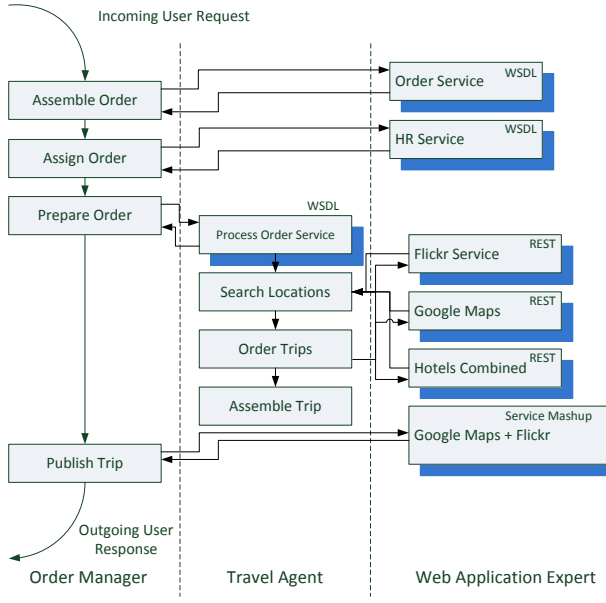


Fig. 1. Travel Agency scenario revisited

As a special service for the customer all on-site trips packaged in the resulting trip are exposed to the travel agency web site. The user is able to retrace the arranged trips on-site by the combinatorial use of Google Maps and Flickr. This Service Mashup was created by the travel agency to present a reproducible booking process to the customer. Before the user starts his trip he may familiarize with local details of the desired destination.

The process outlined in figure 1 exemplifies the coequal consumption of RESTful Web services and WS-* Web services. In the following section the abstraction is outlined to examine both Web service concepts coequally and motivate the use of role-based collaboration in the design of Service Mashups.

5 Service Mashup Abstraction

The scenario illustrated in the previous section outlines the blur of distinction of service integration into conventional process environments. Beside the coequal integration of RESTful Web services and WS-* based Web services the demand of human participant integration rises complexity of the process landscapes. In the SOA paradigm BPEL4People shape up as an accepted approach to introduce generic role models to structure human participant integration. As until now in the domain of Mashups no comparable approach is widely accepted this work proposes a role model derived from the BPEL4People model as introduced in section 3. Beside the two mentioned domains (Service Integration and Participant Integration) the integration of orchestration information into Service Mashups is of vital interest.

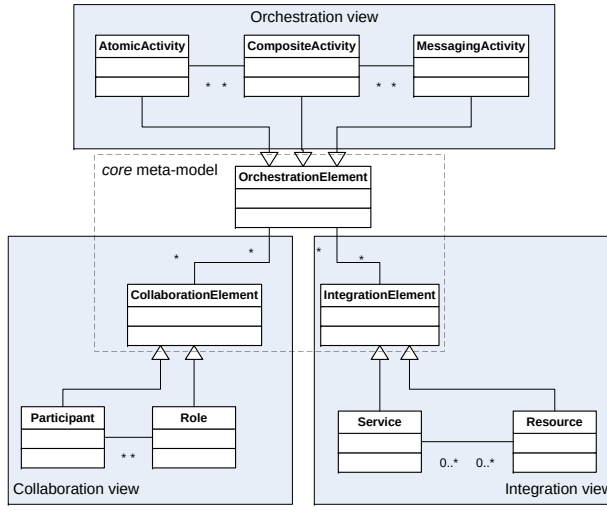


Fig. 2. The Service Mashup Meta-Model

In the WS-* based Web service environment WS-BPEL is the standard to describe and design the orchestration of Web services. WS-BPEL is layered atop of WSDL and decouples the orchestration logic from the service invocation logic by the use of Partner Links. The consequent separation of invocation details is reflected in the use of basic activities which refer to Partner Links and hide the concrete invocation mechanisms from the process definition. The Service Mashup Abstraction continues this separation of invocation details and extends the approach introduced by Tran et al. [10]. Tran et al. use the concept of an architectural view to integrate business process modeling languages at different abstraction levels. The approach maps process descriptions onto appropriate high-level or low-level views. In contrast to the approach elaborated by Tran et al. our work introduces a Control flow view and a Collaboration view on a higher abstraction level from WS-BPEL. This concept maps to the architectural views introduced by Tran et al.

The Service Mashup Abstraction emerges from the model-based integration of different domains into one model. Figure 2 illustrates the underlying meta-model.

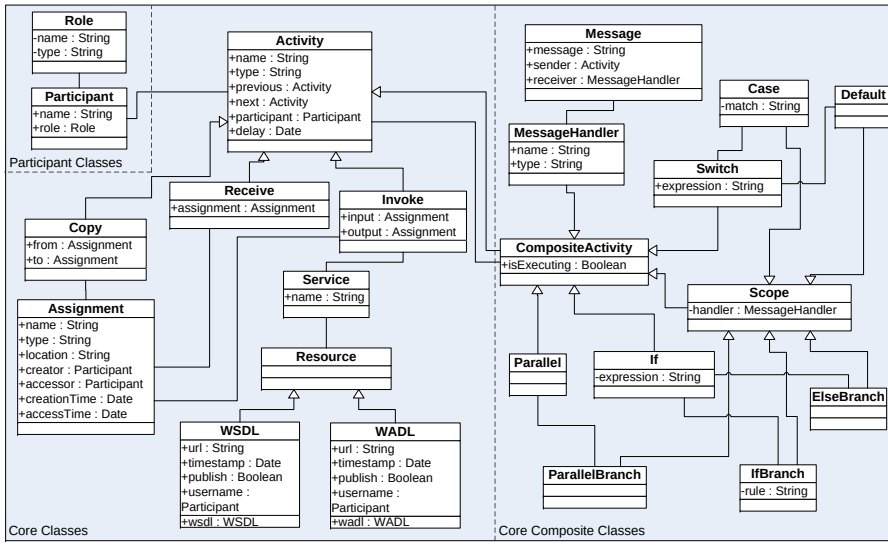


Fig. 3. The Service Mashup Abstraction represented using UML

The core meta-model consists of three elements each representing the base element for the particular view. This language design relates the views and enables a flexible combination of different views. Consider a combination of an Orchestration element like an Activity with a Collaboration Element like a Participant. As Orchestration Element is the parent of each Activity and is related to the Collaboration Element which represents the basis for Participant, each Activity

might have one or more Participants. The Integration Element is not directly connected to the Collaboration Element as the Service Mashup Abstraction reflects the decoupling principle of WS-BPEL to hide invocation details. The relation of Collaboration Elements with Service Invocations is achieved by linking elements of Orchestration Elements. A detailed class structure of the emerging Service Mashup Abstraction is illustrated in figure 3.

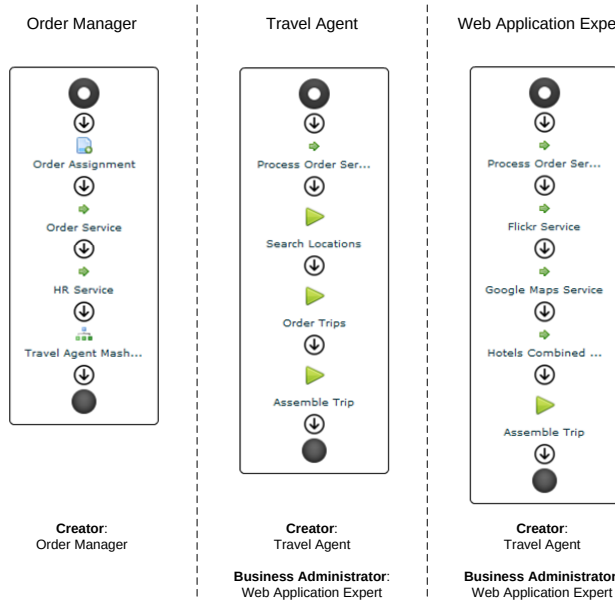


Fig. 4. Process Scenario resolved

```
<Process name="Web Application Expert Process" creator="Travel Agent">
  <Variables>
    <Variable name="Variable1" />
    ...
  </Variables>
  <Invoke ... type="WSDLInvoke" input="Variable1" output="Variable2">
    <WSDL id="1c21aefb-78d2-4963-af0f-9ba8f16df294" operation="GetValues"/>
  </Invoke>
  <Invoke ... type="RESTInvoke" input="Variable5" output="Variable6">
    <Resource uri="http://api.flickr.com/services/..." />
  </Invoke>
  <Invoke ... type="RESTInvoke" input="Variable7" output="Variable8">
    <Resource uri="http://maps.google.com/maps?..." />
  </Invoke>
  <Invoke ... type="RESTInvoke" input="Variable9" output="Variable10">
    <Resource uri="http://www.hotelscombined.com/api?..." />
  </Invoke>
  <Activity name="Assemble Trip" type="Activity" user="martin"/>
</Process>
```

Listing 1.1. Web Application Expert Process

Applying Service Mashup Abstraction on the process scenario introduced in section 4 results in the models illustrated in figure 4. The Order Manager orchestrates two WS-* Web services and refers to the Travel Agent process. The Travel agent models the abstract actions Search Location, Order Trips and Assemble Trip. After modeling this sequence she adds the Web application expert as Business Administrator to the Service Mashup. The Web Application Expert has now full access to the Service Mashups and refines the abstract activities by the according Service invocation elements. The resulting Service Mashup is depicted in listing 1.1. The process model is exposed by the prototype as a resource and might by accessed by HTTP GET operations.

5.1 Service Integration Pitfalls

The abstraction of Service descriptions comes with a risk to neglect the original Service integration paradigm. This might lead to a flippant orchestration of Web services. To prevent misleading orchestrations of Web services this work introduces pitfalls occurring in the coequal integration of RESTful Web services and WS-* Web services.

REST Service enforcement According to Fielding [12], a central concept in a resource oriented architecture is the focus on resources. To adhere to the REST paradigm a uniform interface provides stateless access to resources. Requests to RESTful Web services should include all data needed to fulfill a certain service function. The current trend to expose services of existing Web applications by the use of RESTful Web services to a broader audience blurs these conventions. Consider the RESTful access to a Photo sharing portal[8] as an example: The RESTful Web service client requests a list of 10 photos and exposes these photos on a Web site. The visitor of the web site can navigate through the lists of photos. Each navigation step triggers the RESTful Web service client to request the next list of 10 photos from the photo sharing portal. Figure 5 a) illustrates a stateful service design: The Web service increments and stores the lists to be able to respond to the next request. The second invocation trace in figure 5 b) illustrates a stateless RESTful Web service: The Web service client includes in the invocation, which list of photos it requests. All data needed to fulfill the request is included. This design ensures scalability (Web services are distributable across different load-balancers) by shifting state management responsibility to the client.

The trend to enforce a remote procedure call alike behavior by exposing RESTful Web services must be kept in mind during the design of Service Mashups consuming REST APIs.

[8] f.e. flickr services `http://www.flickr.com/services/api/`, last accessed on April 2009

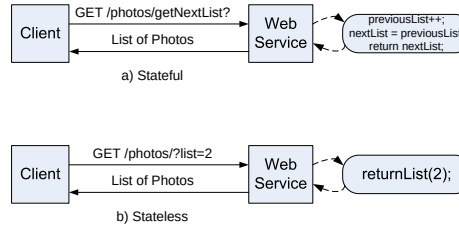


Fig. 5. REST Service enforcement

Protocol neglect HTTP provides different methods for requests[9]. To minimize integration hurdles existing RESTful Web services tend to reduce the HTTP protocol method support to GET and POST requests only. This results in exposing functions like sending a POST request to a RESTful Web service to delete an existing resource. Pautasso et al. [3] refer to the classification of supported HTTP verbs as *Hi-REST* and *Lo-REST*. The former refers to the support for four verbs (`GET`, `POST`, `PUT`, `DELETE`) and the latter refers to the use of `GET` and `POST` verbs only.

Concerning Service Mashup design *Hi-REST* architectures ease the integration of RESTful Web services by adhering stricter to the REST paradigm. The support for `PUT` and `DELETE` verbs indicate more clearly the operation on the resource and therefore the implications on the overall Service Mashup. RESTful Web services adhering to the *Lo-REST* architecture do not provide transparent access to the resources and therefore bear the risk to misleading orchestrations in Service Mashups.

These Service Integration pitfalls might be expanded to a collection of Antipatterns [13] as part of a Service Mashup framework in future work.

6 Conclusion and Future Work

This work introduces a model-driven integration approach towards: a) coequal integration of RESTful Web services and WS-* Web services, b) coequal Web service orchestration and c) collaborative Service design of Service Mashups. The approach derives concepts from established standards and provides an emerged Service Mashup Abstraction. The approach is exemplified in a web-based prototype. The implementation enables the asynchronous design of Service Mashups by implementing a Rich Internet Application. The prototype is reachable under `http://www.expressflow.com`.

Section 3 introduces the role model derived from BPEL4People. This role model is currently limited to design time issues of collaborative Service Mashup design. The expansion of this role model to runtime issues is planned for future work.

[9] For an exhaustive list of methods the interested reader may refer to RFC 2616

Section 5 introduces the Service Mashup Abstraction from the consequent model-driven abstraction of different Web service domains. Whereas the WS-BPEL based business process abstraction seems to be straight forward the co-equal integration of WS-* Web services and RESTful Web services comes with diverse risks. The pitfalls described in this work might be expanded to the definition of Antipatterns as part of the future work.

References

1. Pautasso, C.: BPEL for REST. 7th International Conference on Business Process Management (2008)
2. Rosenberg, F., Curbera, F., Duftler, M. J., Khalaf, R.: Composing RESTful Services and Collaborative Workflows: A Lightweight Approach. IEEE Internet Computing (2008) 24–31
3. Pautasso, C., Zimmermann, O., Leymann, F.: RESTful Web services vs. "Big" Web services: making the right architectural decision. Proceedings of the 17th international conference on World Wide Web (2008) 805–814
4. Benslimane, D., Dustdar, S., Sheth, A.: Service Mashups: The New Generation of Web Applications. IEEE Internet Computing (2008) 13–15
5. Hoyer, V., Stanoesvka-Slabeva, K., Janner, T., Schroth, C.: Enterprise Mashups: Design Principles towards the Long Tail of User Needs. IEEE International Conference on Services Computing (2008) 601–602
6. Maximilien, E. M., Ranabahu, A., Tai, S.: Swashup: situational Web applications Mashups. Conference on Object-oriented programming systems and applications (OOPSLA) (2007) 797–798
7. Curbera, F., Duftler, M., Khalaf, R., Lovell, D.: Bite: Workflow Composition for the Web. Proceedings of the 5th international conference on Service-Oriented Computing (2007) 94–106
8. Held, M., Blochinger, W.: Collaborative BPEL Design with a Rich Internet Application. 8th IEEE International Symposium on Cluster Computing and the Grid (2008) 202–209
9. Ellis, C. A., Gibbs, S. J., Rein, G.: Groupware: some issues and experiences. Communications of ACM (1991) 39–58
10. Tran, H., Zdun, U., Dustdar, S.: View-Based Integration of Process-Driven SOA Models at Various Abstraction Levels. Model-Based Software and Data Integration (2008) 55–66
11. Van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases (2003) 5–51
12. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine, PhD Thesis (2000)
13. Brown, W., Malveau, R., Mowbray, T.: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. Wiley (1998)
14. WS-BPEL Extension for People, IBM and SAP Specification `http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/` last accessed April 2009
15. Web Services Business Process Execution Language, OASIS Standard `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`, last accessed April 2009
16. Web Service Description Languag, W3C Technical Report `http://www.w3.org/TR/wsdl`, last accessed April 2009

Service Composition at the Presentation Layer using Web Service Annotations

Tobias Nestler[1], Marius Feldmann[2], Andre Preußner[1], and Alexander Schill[2]

[1] SAP Research CEC Dresden, 01187 Dresden, Germany
{tobias.nestler, andre.preussner}@sap.com
[2] Technische Universität Dresden, Department of Computer Science, Institute for Systems Architecture, Computer Networks Group
{marius.feldmann, alexander.schill}@tu-dresden.de

Abstract. In the field of Service-Oriented Architectures the implementation of business logic and business processes is well-understood and covered by existing development approaches, but concepts for a lightweight service consumption in order to build interactive service-based applications are still in a preliminary phase. This lack of service-consumer-orientation prevents users with limited IT skills to get easy access to services and their offered functionalities. The paper presents an approach that follows the idea of integration at the presentation layer enhanced by user interface (UI) related service annotations. It describes the relationship of these ideas to already existing mashup approaches and gives an insight into how services can be composed to complex interactive applications in a visual manner without the need to write any code.

1 Motivation and Background

Service-Oriented Architectures (SOA) promise to break former monolithic applications into loosely coupled services that can be distributed across several systems. These services are composed to implement business applications and processes. Even though service composition is well-understood and covered by existing approaches for technical developers using languages such as BPEL, tools and methodologies for enabling end-user service composition have been largely ignored [1]. A promising approach for bridging this gap are mashups that focus on a user-centric and lightweight UI integration [2] by combining the philosophy of SOA and approaches of End-User Development [3]. The need for such situational applications to address individual and heterogeneous needs as well as the shift to more flexible and dynamic business environments encourage the idea of integrating mashup concepts into the enterprise. Our approach shows a way of overcoming limitations of existing mashup approaches [4] in order to build complex interactive service-based applications. Following our preliminary investigations and description of related work [5], this paper discusses the following contributions:

- We propose the usage of UI related service annotations to ease service integration and composition. This limits the effort for the development of

service-based applications to a purely model-driven, visual composition of annotated services that can even be done by end-users. Although existing approaches, such as Dynvoker [6], already cover the generation of user interfaces for single web services dynamically, no solution is available for service composition.
- We adopt the idea of integration at the presentation layer [7] to compose services by combining their presentation front-ends, rather then their application logic or data [2]. Typically, web services are integrated into the application layer of a composite application via their well-defined service interfaces. The service annotations add the missing information about the UI of a single service to lift the integration to the presentation layer.
- We propose a tool environment that allows the creation of interactive service-based applications to nonprogrammers. Most of the existing lightweight composition approaches (overview provided by [8]) support the user only in building complex data representations in form of widgets or feeds, but lack sophisticated concepts following the idea of process mashups [9].
- We propose a automatic Model-driven generation approach for the designed interactive service-based applications. The models used within the approach can be applied for representing applications for various target platforms and different sorts of application partitioning.

2 Towards Visual Service Composition

This section presents our idea of composing services in a visual manner to ease and speed up the development of interactive applications that goes beyond existing visual mashup solutions. These applications combine concepts like multi-page support, dynamic UI behavior (e.g. input suggestion functionality, client-side input validation). Therefore, we introduce the concept of UI related service annotations. These are reusable information fragments attached to the service description, which are typically not available for the application developer. They cover static UI aspects, the behavior of UI elements, and relations between services. Annotations facilitate e.g. the grouping and ordering of UI elements, completion of forms, continuous field updates, or data conversion (more examples in [5]).

UI development is usually a very time consuming task and cannot be done by the targeted end-user group. A trained application developer has to build the UI and integrate the services manually. The developer has to understand the offered interface to integrate the service in an application. This is not necessary anymore, since the integration at the presentation layer is done on a much higher level of abstraction. The user (in the role of the service composer) only works with the presentation front-end of a service instead of an abstract representation in form of a predefined service widget. Therefore, UI fragments are automatically generated for the interaction with the services and represent the interfaces for service input and expected output. The fragments can be inferred from technical details such as the data types of parameters, and be further improved by leveraging the annotations attached to the service. UI fragments consist of freely arrangeable

UI elements like input fields or buttons. A manual implementation of a service wrapper, as usually required in existing visual mashup environments, is not needed anymore.

Our approach facilitates the development of interactive single- and multi-page applications. A page acts as a container for UI elements and represents a screen in the final application. The integration of services (as described above) and the actual service composition can be done for each page separately. The service composer can define data flows between the integrated service operations on a single page (intra-page flow) and between pages (inter-page flow). These data flows can be partially derived from service dependencies defined in the annotations or modeled manually by the service composer in a visual way. Different approaches to support this specific task are currently under investigation. One solution could be the selection of a specific output field of service operation A and drawing a line to the input field of the service operation B. Another way could be that each generated UI fragment offers all of its outputs and the user can select the associated service operation via a context menu or wizard. To transport the idea of multi-page applications to nonprogrammers we use a metaphor which most of the people are familiar with - Microsoft PowerPoint. Each page (or screen) in the final application will be presented like a slide in PowerPoint. Furthermore, it is possible to define a master layout that all pages will use. To build multi-page applications, the pages can be linked to each other by specifying a navigation path.

3 End-User Centric Tool Support

Our visual composition editor which implements the concepts introduced in Sec. 2 is currently under development in the frame of the EU funded project ServFace [10]. The main focus of the composition editor is the empowerment of end-users to develop interactive applications. Multiple design decisions were made based on this requirement. The tool is designed as a rich internet application (RIA) which runs in the web browser of the user and makes an installation dispensable. The annotations facilitate the understanding and simplify the composition of web services. Finally, the visual composition concepts guide the user through the development process by providing intuitive metaphors and hide the complexity of the actual programming task. Our user-centric implementation approach involves iterative evaluations with end-users.

The composition editor is integrated into a three step methodology for the development of interactive applications as explained in [11]. The annotations are created by an IT expert and stored in an annotation model based on a formally defined Meta-model. The visual service composition tool imports in a first step the functional interface descriptions of the web services and their attached annotation models. The result is a platform-specific object model structure kept in the tool representing the complete designed application.

Figure 1 shows a mockup of the envisioned composition editor. The user can import annotated services that shall be used in the application. These services

are displayed with their operations in the `Service Operations` palette. The user can drag service operations from the palette to the composition area. The editor displays the UI fragment inferred from the operation interface and the service annotations. The user can refine the layout, delete unwanted UI widgets or add additional ones from the `Widgets` palette, and define intra-page data flows. Besides this basic mashup editor functionality our composition editor pro-

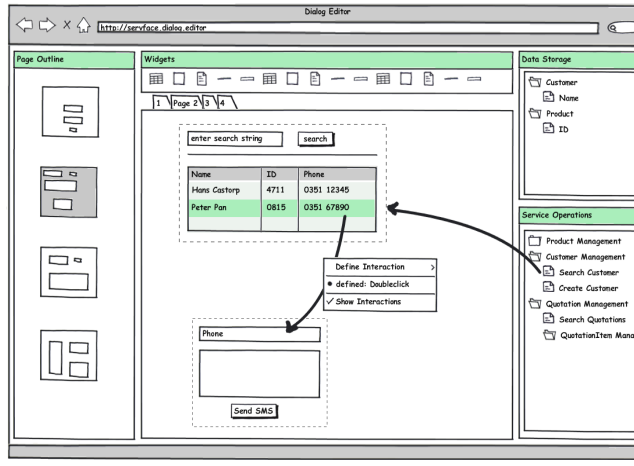


Fig. 1. Mockup of the Visual Composition Editor

vides innovative features especially designed towards the development of process mashups. The user can define inter-page data flows by dragging parameters or return values to the `Data Storage` and use them to fill UI elements at other pages. The editor supports the definition of the navigation flow of multi-page applications. This can be done either implicitly by separating the input and output of an operation to different pages, or explicitly by creating a new page and adding navigation buttons for the page transition. Concepts for an end-user friendly design of features like the inclusion of additional operations for data filtering or conversion, and the merging of UI elements to call multiple operations with one user interaction are under investigation.

After finishing the application development, the mentioned object structure is serialized to a model coined Composite Application Model (CAM). Its underlying Meta-model is reused for representing applications for various platforms. A serialized CAM is used as storage format for the composition tool and as input for generating executable application as described in the next section.

4 Generating Applications

In regards of bringing the composed interactive application to execution, the decision has been made to use a code generation mechanism. In comparison

to deploying the design-time outcome on a specific interpreter, code generation promises a higher efficiency. The chosen approach is realized in a model-driven manner. In order to bring the instance of the CAM Meta-model closer to the executable application and to resolve the annotations that are explicitly represented within the CAM to runtime information, a Model-to-Model transformation is applied in a first step. As it is the case for the CAM, the target Meta-model (named PROSAIC) can be reused for representing applications for a variety of platforms. A major challenge in realizing this approach has been to define a *reference architecture* for service-based interactive applications reflected in this Meta-model that can be used as an abstraction from concrete platforms and frameworks. Figure 2 shows an example of the control and data flow within the reference ar-

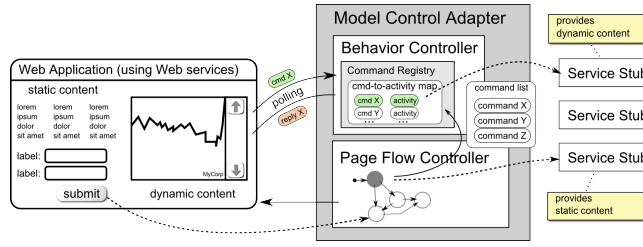


Fig. 2. UI- to Service-interaction within the reference architecture

chitecture developed for our approach. To support single page applications (e.g. RIAs) as well as multi-page applications, a differentiation between a page flow controller and a behavior controller has been introduced. Both controllers are located within the Model-Control-Adapter (MCA). Its major task is to coordinate the interaction between the user interface and the service infrastructure. The page flow controller contains a set of states and transitions between states. On state activation the state registers a set of commands within the behavior controller. These commands are mapped to a set of activities where an activity can contain actions such as the invocation of a service or assigning values to global variables. The commands are used for realizing the behavior of the user interface of the page associated with the state. For example if a widget displaying stock information should be updated in regular intervals, it triggers a command in a loop and sends it to the MCA (e.g. via Ajax functionalities). The MCA calls the service that returns the stock data and sends a reply to the UI that includes the new stock values into the widget.

This proposed reference architecture is reflected within the PROSAIC Metamodel. During several tests it has been evaluated that instances of this Metamodel can be transformed via Model-to-Model and Model-to-Code transformation to several platform and framework specific source code. Besides transforming it to Web applications (Dojo toolkit and the Spring framework) it has been proven that it can be applied for generating fat clients e.g. for mobile devices (Google Android applications).

The generation of the resulting application is done *completely automatically* by using an automatic build script for Ant that is triggered by the composition editor and that starts the M2M transformation implemented in ATL and the M2C transformation implemented by using openArchitectureWare. Furthermore this script enables the packaging and deployment of web applications.

Yet an open issue is the formal definition of the relations between the service annotations *kept explicitly* within the CAM and the PROSAIC Meta-model. This formal definition promises a starting point for a simplification of the template creation for the M2M transformation.

5 Conclusion and Future Work

The concept of presentation integration can be seen as the next major step in the integration area [7]. Our paper presented an approach to lift the service composition to the level of presentation integration via UI related service annotations. The presented visual composition concepts as well as the associated tool will empower nonprogrammers to create composite applications, which suit their requirements and individual needs. The active involvement of the actual service consumer in the integration and composition process can result in a more sufficient usage of knowledge, specific for their domain, and raise their productivity.

References

1. Ro, A., Xia, L.S.Y., Paik, H.Y., Chon, C.H.: Bill Organiser Portal: A Case Study on End-User Composition. In WISE (2008)
2. Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R.: Understanding UI Integration: A survey of problems, technologies, and opportunities. IEEE Internet Computing (May/June 2007)
3. Hoyer, V., Stanoevska-Slabeva, K.: The Changing Role of IT Departments in Enterprise Mashup Environments. In 2nd International Workshop on "Web APIs and Services Mashups" (Mashups08) (2008)
4. Nestler, T.: Towards a Mashup-driven End-User Programming of SOA-based Applications. In 10th International Conference on Information Integration and Web-based Applications & Services (iiWAS) (2008)
5. Nestler, T., Feldmann, M., Schill, A.: Design-Time support to create user Interfaces for service-based applications. In International Conference WWW/Internet (2008)
6. Spillner, J., Feldmann, M., Braun, I., Springer, T., Schill, A.: Ad-hoc Usage of Web Services with Dynvoker. Towards a Service-Based Internet, First European Conference, ServiceWave 2008, Madrid, Spain (2008)
7. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M.: A Framework for Rapid Integration of Presentation Components. In WWW'07 (2007)
8. Hoyer, V., Fischer, M.: Market Overview of Enterprise Mashup Tools. In ICSOC (2008)
9. Young, O.: The Mashup Opportunity. In Forrester Research Report (May 2008)
10. ServFace Consortium: ServFace Research Project (2008) http://www.servface.eu/.
11. Feldmann, M., Janeiro, J., Nestler, T., Hübsch, G., Jugel, U., Preussner, A., Schill, A.: An Integrated Approach for Creating Service-Based Interactive Applications. In INTERACT 2009 (to appear)

Towards Flexible Integration of Any Parts from Any Web Applications for Personal Use

Hao Han, Junxia Guo and Takehiro Tokuda

Department of Computer Science, Tokyo Institute of Technology
Meguro, Tokyo 152-8552, Japan
{han, guo, tokuda}@tt.cs.titech.ac.jp

Abstract. Mashup has brought new creativity and functionality to Web applications by the integration of Web services from different Web sites. However, most existing Web sites do not provide Web services currently, and the Web applications are more widely used than Web services as a method of information distribution.
In this paper, we present a method to integrate any parts from any Web applications for personal use. For this purpose, we propose a flexible integration method by the description and extraction of Web application contents. Our implementation shows that we can integrate any parts easily from not only the ordinary static HTML pages but also the dynamic HTML pages containing Web contents dynamically generated by client-side scripts.
Keywords: Web Application, Web Service, Information Extraction, Information Integration, Mashup, DHTML, JavaScript

1 Introduction

More and more information/knowledge is available on the Web with the development of the Internet, but they are not always in the forms that support end-users' needs. Mashup implies easy and fast integration of information in order to enable users to view diverse sources of data in an integrated manner. However, the contents used in most of the current mashup Web applications are typically obtained from the third party sources through the public Web service APIs, and the integration is limited to the Web sites that provide the open Web service APIs mostly. Although there are widely popular and successful Web services such as Google Maps API [1] and YouTube Data API [2], unfortunately, most existing Web sites do not provide Web services. Web applications are still the main methods for the information distribution. For example, the famous global news site CNN [3] provides the online news search function at site side for general users. However, this news search function can not be integrated into other systems because CNN does not open search function as a Web service. Similarly, BBC Country Profiles [4] does not provide the Web service APIs and it is difficult for the developers to integrate it with other Web services.

In this paper, we present a method to integrate any parts from any Web applications for personal use. For this purpose, we propose a flexible integration method by the description and extraction of Web application contents. We

defined WACDL (Web Application Contents Description Language), an XML-based language that provides a model for describing Web application contents, to configure the locations and scopes of target contents from Web applications. We also constructed an extraction and integration system, which extracts the target contents and executes the contents integration to generate the mashup Web applications. Our implementation shows that we can integrate any parts easily from not only the ordinary static HTML pages but also the dynamic HTML pages containing Web contents dynamically generated by client-side scripts.

The organization of the rest of this paper is as follows. In Section 2 we give the motivation of our research and an overview of the related work. In Section 3 we construct an example of mashup Web application, and explain our Web application contents description and integration system in detail. We give an evaluation of our approach in Section 4. Finally, we conclude our approach and give the future work in Section 5.

2 Motivation and Related Work

Most integration technologies are based on the combination of Web services or Web feeds. Yahoo Pipes [5] and Microsoft Popfly [6] are the composition tools to aggregate, manipulate, and mashup Web services or Web feeds from different Web sites with a graphical user interface. Mixup [7] can quickly build complex user interfaces for easy integration by using the Web service APIs available. Mashup Feeds [8] supports integrated Web service feeds as continuous queries. It creates the new services by connecting the Web services using join, select and map operations. Like these methods, Google Mashup Editor [9] is also limited to the combination of existing Web services or Web feeds.

For the integration of parts from Web applications without Web service APIs, the partial Web page clipping method is widely used. The users clip a selected part of Web page, and paste it into a personal Web page. Internet Scrapbook [10] is a tool that allows users to interactively extract components of multiple Web pages by clipping and assembles them into a single personal Web page. However, the extracted information is a part of static HTML document and the users can not change the layout of the extracted parts. C3W [11] provides an interface for automating data flows. With C3W, the users can clip elements from Web pages to wrap an application and connect wrapped applications using spreadsheet-like formulas, and clone the interface elements so that several sets of parameters and results may be handled in parallel. However, it does not appear to be easy to realize the interaction between different Web applications and needs a special Web browser. Extracting data from multiple Web pages by end-user programming [12] is more suitable to generate mashup applications at client side. Marmite [13], implemented as a Firefox plug-in using JavaScript and XUL, uses a basic screen-scraping operator to extract the content from Web pages and integrate it with other data sources. The operator uses a simple XPath pattern matcher and the data is processed in a manner similar to Unix pipes. However, these methods can only extract Web contents from static HTML

pages as Mashroom [14] and Dapper [15]. MashMaker [16] is a tool for editing, querying, manipulating and visualizing continuously updated semi-structured data. It allows users to create their own mashups based on data and queries produced by other users and by remote sites. However, they do not appear to support the integration of dynamically generated Web pages like the result pages from form-based query.

These current methods are based on the existing Web service APIs or Web feeds, or need the end-user programming, or have other limitations. They have realized the integration of Web applications to some extent, but still can not extract and integrate the Web contents dynamically generated by client-side scripts that become more and more in Web applications with the development of Web 2.0. To address these problems, we propose a novel approach to integrate any parts from any Web applications by the description and extraction of the target Web application contents. Compared with the developed work, our approach has the following features.

- We extend the range of Web contents from Web services to the general Web applications. Any parts from any Web applications are available.
- We propose WACDL to describe the target parts of Web applications. The WACDL file is generated easily and does not need the programming.
- We integrate any parts from not only the ordinary static HTML pages but also the dynamic HTML pages containing Web contents dynamically generated by client-side scripts.

We explain our approach by constructing an example of mashup Web application, and give an evaluation in the following sections.

3 Web Application Contents Description and Integration

Our integration is based on the description and combination of target parts of Web applications. In our approach, the target parts of Web applications are the visible contents in the Web pages such as the text, link, graph, video, flash and etc. As shown in Figure 1, the whole process includes the following steps.

1. We describe the target parts of Web applications in a WACDL file.
2. We get the request from client side and send it to the target Web applications. We search for the target parts from the response Web pages according to the description in WACDL.
3. We extract the contents and control the visibility of them.
4. We integrate the extracted contents and arrange their layouts to generate a resulting page of mashup Web application.

We generated an example of mashup Web application. It integrates the parts from the following five Web applications and realizes the search function of country information. As shown in Figure 2, after the users input the country name and send the request, the mashup Web application sends the request to

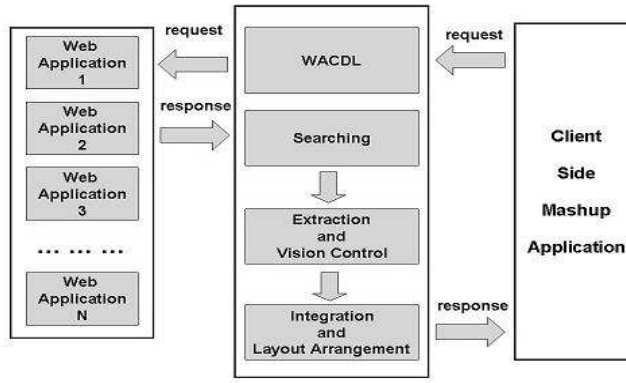


Fig. 1. The outline of our integration approach

each target Web application and receives the response Web pages. It searches for the target parts from the Web pages and shows them in an integrated resulting Web page.

- Part A: Country name and country flag from Country Fast Facts of CBS News [17].
- Part B: Weather information from Weatherbonk [18], which is a mashup application integrated by weather service and Google Maps service. The part of weather information is created by client-side scripts, which can respond to click and span events.
- Part C: The country's location, basic information and leader's photo from BBC Country Profiles [4].
- Part D: The latest corresponding news articles from BBC News [19].
- Part E: Pictures from Trippermap [20] shown with the map, which can respond to click event and show the relevant pictures.

We explain our integration approach based on the actual generation process of this example.

3.1 Web Application Contents Description Language

We need a model to describe Web application contents if we want to use their functionalities and contents like a Web service. Compared with the Web services, it is not easy to use Web applications by the end-user programming. Without the interface like SOAP or REST, we have to use the extraction and emulation technologies to interact with Web applications. The extraction is used to find and extract the target contents from Web pages, and the emulation is used to realize the process of sending request and receiving response.

We propose Web application contents description language (WACDL). It is XML-based as shown in Figure 3, and used to describe the necessary information

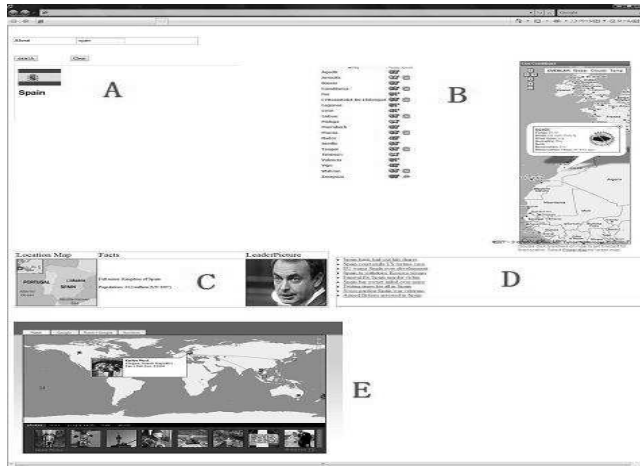


Fig. 2. The example of mashup application

for the extraction and emulation. A WACDL file represents a configuration of mashup Web application and includes the following items for each target Web application.

- StartPage: StartPage is a Web page of target Web application. From this StartPage, the request of end-user is submitted. The value of StartPage is the URL of Web page.
- InputArea: InputArea is the position information of request-input element in the StartPage. If there are other elements with the same InputType in a StartPage, we have to define the InputArea. For example, we need to select one InputBox as the request-input element if there are more than one InputBoxes in StartPage. The value of InputArea is the XPath-like expression of request-input element. Otherwise, the InputArea value is set as null.
- InputType: InputType is the type of request-input element in the StartPage. Usually, the value is InputBox (text input field), or OptionList (drop-down option list in selectbox), or LinkList (anchor list).
- ContentArea: ContentArea is the position information of target contents in the response Web page and used by the extraction. The value of ContentArea is the XPath-like expression of target parts.
- ContentType: ContentType is the type of target contents. There are two types of contents in a Web page: static Web contents and dynamic Web contents. The static Web contents are the unchangeable parts shown on the Web pages after the pages are fully loaded and during the viewing process. They include two kinds of information: property and structure. Property is text, image, link or object. Text is the character string in Web pages such as an article. Image is one instance of the graph. Link is a reference in a hypertext document to another document or other resource. Object is one instance of the video or other multimedia file. Structure is single

occurrence or continuous occurrence. A single occurrence is a part without similar ones such as the title of an article. A continuous occurrence is a list of parts with similar ContentArea values such as result list in a search result page. The dynamic Web contents are the parts dynamically generated or changed by client-side scripts in dynamic HTML pages according to the users' operations.

– ContentStyle: ContentStyle is the layout of target contents in the integrated resulting Web page. It is limited to the static Web contents usually. For the static Web contents, the extraction results are in XML format and the ContentStyle refers to XSLT [21] files defined by end-user. For the dynamic Web contents, the extracted parts are shown in their original styles and the ContentStyle value is null.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<channel>
    <target>
        <StartPage>
            URL of Web page where request is submitted
        </StartPage>
        <InputArea>
            Path of request-input element in HTML document of StartPage
        </InputArea>
        <InputType>
            Type of request-input element in StartPage
        </InputType>
        <ContentArea>
            <content>
                Path of target part in HTML document of response Web page
            </content>
            ...
            <content> ... </content>
        </ContentArea>
        <ContentType>
            <type>
                Type of target contents
            </type>
            ...
            <type> ... </type>
        </ContentType>
        <ContentStyle>
            <type>
                Layout of target contents in resulting Web page
            </type>
            ...
            <type> ... </type>
        </ContentStyle>
    </target>
    <target>
    ...
    </target>
</channel>
```

Fig. 3. Format of WACDL file

These six items describe how to get and integrate the target Web contents. Like a batch file, each WACDL represents a series of Web contents from different sources. Table 1 gives the description of our example mashup Web application shown in Figure 2. We developed Path Reader [22], a tool to read the path of target part by GUI, which is modified from Mouseover DOM Inspector [23]. The users can get the paths easily by mouse clicking the target parts, and do not need to read the HTML source codes manually.

3.2 Target Parts Searching

According to the description in WACDL file, we search for the target Web contents from the Web applications. There are two steps during this process. First, we get the response Web pages as the target Web pages. Then, we search for the target parts in the response Web pages.

Web applications provide the request-submit functions for the users. For example, search engine applications provide the text input field in the Web page for keywords inputting by the users. The users give the query keywords and submit the requests to server sides. There are three basic types of methods to send requests and get the response Web pages. The first type is to click an option in drop-down list of selectbox in a Web page by mouse to view a new Web page. The second type is to enter the query keywords into a form-input field by keyboard and click the submit button by mouse to send the query. The third type is to click a link of link list in a Web page by mouse to go to the target Web page. For the request submitting, there are POST and GET method, and some Web sites use the encrypted codes or randomly generated codes. In order to get the response Web pages from all kinds of Web applications, we use HtmlUnit [24] to emulate the submitting operation instead of URL templating mechanism. The emulation is based on the event trigger of the element of *InputType* within the *InputArea* of *StartPage* as follows.

- In the case of InputBox, the text input field is found according to the InputArea and the query keywords are inputted. Then the click event of the submit button is triggered to send the request and get the response Web page.
- In the case of LinkList, the text contained inside each link tag within InputArea is compared with keyword until the matched one is found. Then the click event of link is triggered to get the target Web page.
- In the case of OptionList, the text of each option within InputArea is compared with keyword until the matched one is found. Then the select event of option is triggered to get the target Web page.

ContentArea is used to find the target parts from the Web page. In the tree structure of HTML document, each path represents a root node of subtree and each subtree represents a part of Web page. Usually, the response Web pages have the same or similar layouts if the requests are sent to the same request-submit function. During the node searching, if a node can not be found by a path, the

Table 1. Description of our mashup application example

Target	Item	Value
A	StartPage	{[http://www.cbsnews.com/stories/2007/08/30/country_facts/ main3221371.shtml]}
	InputArea	{[null]}
	InputType	{[LinkList]}
	ContentArea	{[BODY:0:www-cbsnews-com/DIV:0:frame/TABLE:3:content/ TR:0:null/D:1:centerColumn/TDIV:0:centerColumnContent/ DIV:1:null/DIV:4:null/]}
	ContentType	{[dynamic]}
	ContentStyle	{[null]}
B	StartPage	{[http://www.weatherbonk.com]}
	InputArea	{[BODY:0:page/DIV:3:bonkLeftNavColumn/DIV:1:null/DIV:0:null/ FORM:0:searchForm/]}
	InputType	{[InputBox]}
	ContentArea	{[BODY:0:page/DIV:17:bonkForecastColumn/DIV:2:grid/], [BODY:0:page/DIV:18:bonkMapColumn/]}
	ContentType	{[dynamic],[dynamic]}
	ContentStyle	{[null]}
C	StartPage	{[http://news.bbc.co.uk/1/hi/country_profiles/default.stm]}
	InputArea	{[null]}
	InputType	{[OptionList]}
	ContentArea	{[BODY:0:body/DIV:0:null/DIV:5:null/TABLE:0:null/TR:0:null/ TD:1:null/TABLE:2:null/TR:1:null/TD:0:null/TABLE:1:null/ TR:0:null/TD:0:null/DIV:0:null/IMG:0:null/], [BODY:0:body/DIV:0:null/DIV:5:null/TABLE:0:null/TR:0:null/ TD:1:null/TABLE:2:null/TR:1:null/TD:0:null/DIV:20:quickguide/ DIV:0:null/DIV:1:content/UL:0:null/], [BODY:0:body/DIV:0:null/DIV:5:null/TABLE:0:null/TR:0:null/ TD:1:null/TABLE:2:null/TR:1:null/TD:0:null/P:28:null/ TABLE:0:null/TR:0:null/TD:0:null/DIV:0:null/IMG:0:null/]}
	ContentType	{[image],[text list],[image]}
	ContentStyle	{[bbc-country-layout.xslt]}
D	StartPage	{[http://search.bbc.co.uk/search?tab=ns&scope=all]}
	InputArea	{[BODY:0:null/DIV:1:blq-container/DIV:1:blq-mast/FORM:1:null/]}
	InputType	{[InputBox]}
	ContentArea	{[BODY:0:null/DIV:1:blq-container/DIV:2:blq-main/ DIV:0:blq-content/DIV:2:primary/DIV:0:null/UL:2:null/]}
	ContentType	{[text list]}
	ContentStyle	{[search-result-layout.xslt]}
E	StartPage	{[http://www.trippermap.com]}
	InputArea	{[null]}
	InputType	{[null]}
	ContentArea	{[BODY:0:null/DIV:0:wrapper/DIV:6:maptabs/]}
	ContentType	{[dynamic]}
	ContentStyle	{[null]}

similar paths would be used to try searching for the node. The definition and usage of similar path are described in [25] in detail.

3.3 Contents Extraction and Visibility Control

The target Web contents are mixed with other unnecessary elements such as the advertisements in a Web page, and shown in different fonts, sizes or colors if they come from different Web applications. In order to get a well designed resulting page, the users may define a customizable layout for the Web contents. After the target parts are found, the Web contents are extracted from the nodes in text format excluding the tags of HTML document according to the corresponding *ContentType* for the static Web contents. The detailed extraction algorithm can be found in [25]. The extracted static contents are in XML format, and would be transformed into HTML document by *ContentStyle*.

For the dynamic Web contents, we use a novel hide-and-display method to control their visibility instead of the static contents extraction method because we need to keep the functionalities of client-side scripts. The scripts use DOM operation to control the dynamic parts of Web pages usually, and sometimes access the elements outside the target parts such as the hidden values. If we remove the other parts from the target parts, the original execution environment of scripts would be broken and the scripts could not run normally. Here, we keep all the parts of each Web page and change the visibility according to the following steps in order to show the target parts and hide the other parts of Web page.

1. We create a node list L, and push the nodes found in Section 3.2 into L.
2. We create a node list L', and push the ancestor nodes and descendant nodes of each node in L into L'.
3. We push all the nodes in L' into L.
4. We hide all the parts of target Web page by setting the property "display" of attribute "style" of all the nodes to "none" (style.display="none") except the nodes in L.
5. We modify the HTML source to accelerate the Web page loading procedure. It is not necessary to load the external files such as the image files or video files if they are not within the target Web contents. These files are not shown in resulting Web page and would cost the loading time. For example, we set the attribute "src" of <img> to null and the image files would not be loaded.
6. We add the <base> between the <head> and </head> tag. The value of attribute "href" is the URL of the target Web page. By adding the <base> tag, all files invoked by relative paths can be found correctly including the external image files, flash files, script files and etc.

By the Web contents extraction method and the hide-and-display method, we can get any parts from any Web applications, and maintain the functionalities of dynamic Web contents.

3.4 Integration and Layout Arrangement

Finally, we integrate the parts from different Web applications in a resulting page. We use iframe [26] as the Web content container of each part.

IFrame (inline frame) is an HTML element which makes it possible to embed an HTML document into another HTML document. While regular frames are typically used to logically subdivide the Web contents of one Web page, iframes are more commonly used to insert Web contents from another Web site into the current Web page. Moreover, iframe is supported by all popular browsers.

According to the WACDL file, we create as many iframes as the number of <target> tags when the resulting page is loading. We show the Web contents from each Web application in an iframe. Each iframe runs in an independent manner, and does not exchange the data or events between each other. When the users click a link in a iframe, a new window pops up to show the target contents if the target of this link is another Web page or document. Our iframe supports the layout arrangement of users. In the resulting Web page, end-users can move iframes by dragging and dropping operations to adjust the locations as shown in Figure 4, which is more compact than the default layout arrangement of iframes in Figure 2.

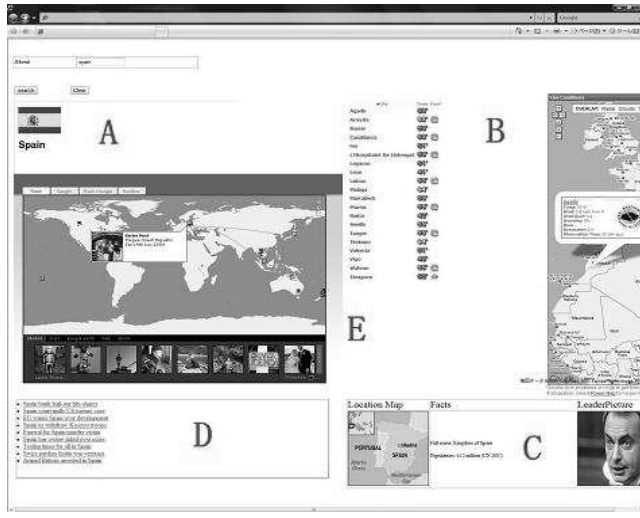


Fig. 4. The resulting page of mashup application

4 Evaluation

Our mashup Web application is constructed by the integration of our Web contents extraction method and hide-and-display method. It is developed by Java

and JavaScript, and works well on the Internet Explorer 7 and JDK 1.6 under Windows XP SP3 and Windows Vista SP1.

We integrated various parts from different kinds of Web applications, and prove our approach is applicable to the general Web applications [22] including the CNN News [3], Wikipedia [27] and Yahoo Finance [28]. However, the emulation of HtmlUnit is slow for some Web sites and costs more time than URL templating mechanism.

Our extraction method is based on the fact that the response Web pages from the same Web application use the same or similar layouts. If the response Web pages use the different layouts, the extraction precision would become low because the paths of the target parts vary with the layouts of Web pages. Moreover, if the layout of the Web page is updated, the users have to change the value of ContentArea in WACDL file. Our WACDL file is still manually generated now. The users have to analyze the structure of target Web applications and fill the corresponding information for each item though they do not need to read the source codes of HTML documents.

Although we add the <base> to deal with the relative paths, unfortunately, the client-side scripts of some Web applications use the relative path as the parameter of function as follows, and the external file can not be loaded correctly.

```
var flashfile = new ObjectLoad("flash.swf", "300", "400");
```

Our integration approach makes it possible for end-users with no or little programming experience to implement the integration of Web contents from various Web applications without Web service APIs. The range of Web contents are extended from Web services to the general Web applications. Any parts from any Web applications are available, not only the ordinary static HTML pages but also the dynamic HTML pages containing Web contents dynamically generated by client-side scripts, even the parts from mashup Web application. Compared with the programming, the WACDL is easy to read, write and update.

5 Conclusion

In this paper, we have presented a novel approach to integrate any parts from any Web applications for personal use. Our approach uses the WACDL to describe the Web application contents and functionalities, and realizes the integration by Web contents extraction method and hide-and-display method. By our extraction and integration system, the users can construct the mashup Web applications without the programming.

As future work, we will modify our approach to propose a friendly GUI for users to generate the WACDL file more easily. Moreover, we would like to explore more flexible ways of integration of Web applications, Web services and other Web contents. Additionally, besides the current developed Java-based emulation and extraction system, we will develop a JavaScript-based system in future.

References

1. Google Maps API: http://code.google.com/apis/maps/.
2. YouTube Data API: http://code.google.com/apis/youtube/.
3. CNN: http://www.cnn.com.
4. BBC Country Profiles: http://news.bbc.co.uk/2/hi/country_profiles/.
5. Yahoo Pipes: http://pipes.yahoo.com/pipes/.
6. Microsoft Popfly: http://www.popfly.com.
7. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M.: A framework for rapid integration of presentation components. In: The Proceedings of the 16th International Conference on World Wide Web. (2007)
8. Tatemura, J., Sawires, A., Po, O., Chen, S., Candan, K.S., Agrawal, D., Goveas, M.: Mashup feeds: Continuous queries over Web services. In: The Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. (2007)
9. Google Mashup Editor: http://editor.googlemashups.com.
10. Koseki, Y., Sugiura, A.: Internet scrapbook: Automating Web browsing tasks by demonstration. In: ACM Symposium on User Interface Software and Technology. (1998) 9–18
11. Fujima, J., Lunzer, A., Hornbaek, K., Tanaka, Y.: C3W: clipping, connecting and cloning for the Web. In: The Proceedings of the 13th International World Wide Web conference. (2004)
12. Han, H., Tokuda, T.: A method for integration of Web applications based on information extraction. In: The Proceedings of the 8th International Conference on Web Engineering. (2008)
13. Wong, J., Hong, J.I.: Making mashups with marmite: Towards end-user programming for the Web. In: The Proceedings of the SIGCHI Conference on Human factors in computing systems. (2007)
14. Wang, G., Yang, S., Han, Y.: Mashroom: end-user mashup programming using nested tables. In: The Proceedings of the 18th International Conference on World Wide Web. (2009) 861–870
15. Dapper: http://www.dapper.net.
16. Ennals, R., Garofalakis, M.: MashMaker: Mashups for the masses. In: The Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. (2007)
17. CBS News: http://www.cbsnews.com/stories/2007/08/30/country_facts/main3221371.shtml.
18. WeatherBonk: http://www.weatherbonk.com.
19. BBC News: http://www.bbc.co.uk.
20. Trippermap: http://www.trippermap.com.
21. XSL Transformations: http://www.w3.org/TR/xslt20/.
22. Guo, J., Han, H., Tokuda, T.: A new partial information extraction method for personal mashup construction. In: The Proceedings of the 19th European - Japanese Conference on Information Modelling and Knowledge Bases. (2009)
23. Mouseover DOM Inspector: http://slayeroffice.com/content/tools/modi.html.
24. HtmlUnit: http://htmlunit.sourceforge.net/.
25. Han, H., Tokuda, T.: WIKE: A Web information/knowledge extraction system for Web service generation. In: The Proceedings of the 8th International Conference on Web Engineering. (2008)
26. iframe: http://en.wikipedia.org/wiki/iframe.
27. Wikipedia: http://www.wikipedia.com.
28. Yahoo Finance: http://finance.yahoo.com.