2009 International Symposium on Computing, Communication, and Control (ISCCC 2009)
Proc .of CSIT vol.1 (2011) © (2011) IACSIT Press, Singapore

Enhancing Mashup Composition with AuraGen

Reza Karimpour [1‡] and Fattaneh Taghiyareh [1]

[1] ECE Department, University of Tehran
Tehran, Iran

Abstract. With the wide spread use of Web in everyday life of people, demand for more flexible Web applications has lead to emergence of a new breed of Web Application: Web mashup. While creating mashups, a lot of burden is imposed on users, leaving the weaving of the inputs and outputs of services to users. In this paper we present AuraGen, a facilitated and easy to use Web service mashup composition system. AuraGen generates ReST Web service descriptions and human readable documentation automatically and directly from Web service objects without manual interpretation. With its technical features, we consider AuraGen an ideal solution for both developers as well as users to make mashup composition a lot easier without any extra cost on developer's side.

Keywords: Mashup, ReST, Web Service Description

1. Introduction

After wide spread usage of Web services in business to business scenarios, many began to investigate incorporating benefits of Web services in smaller environments like business to customer settings. These efforts led to the idea of Web mashups [1]. A mashup is a novel way of compositing Web services into a Web application that satisfies client's requirements. These Web applications are created (composed) of smaller Web services each of which designed particularly for a special purpose. Currently there is a great investment from companies like Amazon, Google and Microsoft toward this new breed of Web application [2]. Mashups are generally based on ReST (Representational State Transfer) [3]. ReST is an architectural style, native to the way Web and HTTP are functioning. The advantage of ReST that concerns clients is ease of use. The only requirement to invoke a ReST Web service is an ordinary Web browser with no special add-on, making ReST an appealing tool for a foundation of more flexible Web applications like mashups. Unfortunately ReST suffers from some problems. First there is a lack of development tools that directly target ReST Web services. Although in recent years some effort is appearing online like Restlet for Java [4] and WCF (Windows Communication Foundation) with ReST support for.Net framework [5]. Another issue regarding ReST is the lack of an agreed upon machine readable service description. One of the main factors that helped wide spread adaptation of SOAP [6] based Web services is the WSDL [7] de-facto standard that describes the interface of Web services. Currently there is no adopted equivalent for ReST, leaving the service composition task to manual inspection of human readable documentations.

In this research we present AuraGen, an **au**tomatic **R**eST service description (**a**nnotation) and documentation generation system based on reflexive languages. We use AuraGen for automatic generation of Web service descriptions and human readable documentation directly from Web services objects without manual interpretation. Furthermore we use those service descriptions along with related documentation to create a mashup composition tool that greatly simplifies the task of mashups creation. We consider AuraGen

[‡] Corresponding author. Tel.: + 989125501204; fax: +982144458095
E-mail address: r.karimpour@ece.ut.ac.ir.

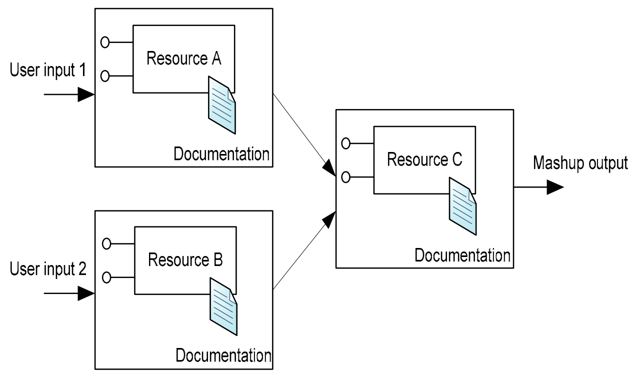


Figure 1. A typical data centric mashup

```
public class RegResult
{
    public bool Succeeded;
    public string Error;
    public int UserId;
}
[ServiceContract]
public class RegService
{
   [OperationContract]
   [WebInvoke(UriTemplate =
   "newuser/{name}", Method = "POST")]
   RegResult CreateUser(string name){
   …
}
```
Figure 2. A ReST Web service in WCF

an ideal solution for both developers as well as users to make mashup composition easier without any extra cost on developer's side. Using this system we will show how mashup composition process can be facilitated, leaving the user to think about what he/she wants rather than how.

2. Mashup Composition

Currently the building block of Web service mashups is lightweight Web services. The glue that relates theses Web services together is mainly JavaScript so services that are JavaScript friendly are preferred. XML-RPC and ReST are main participants. Being a more disciplined technology, ReST has gained much attention in recent years [8, 9].

Industry has started to develop tools that boost productivity of developers building ReST based solutions. One of the solutions that brought ReST to.Net framework is WCF, the new library for service interoperability from Microsoft. As shown in Figure 1, a typical mashup composition involves binding inputs and outputs of one or more Web service into a more specific and mature Web service. The user finds appropriate services based on her/his needs by inspecting the human readable documentation of Web services. Each service may have many inputs as well as many outputs (one output that constitutes many smaller parts). Once again, user is responsible to handle the mediation between ReST Web services. There are many problems with the human readable documentation of Web services. First, they are tailored toward people with some knowledge of programming. In addition, they should be aware of the conceptions like Data Type and argument, transactions and etc. Second, the aforementioned documentations are usually hand crafted by programmers and refined by others. Consider a situation in which the Web service is updated while the documentation is not. Such Inconsistency problems are due to storing one truth in multiple places as discussed by Hammer in his classic paper [10].

3. Automatic Description Generation

After proposing the AuraGen, we describe key design decisions along with their justification.

3.1. Description language

For annotating the Web services we have to define a description standard. Benefits of ReST service descriptions are (1) modelling and visualization support for development of modelling tools for relationship and choreography analysis and manipulation of resources, (2) automated generation of stub and skeleton code and code for manipulation of resource representations, and (3) configuration of client and server using a portable format [11]. There are many propositions but the two prominent description standards are WSDL 2.0 [7] and WADL [11]. Each of these specifications has advantages over the other but WADL seems a more natural choice in case of ReST because WADL is:

- Resource oriented as ReST is, while WSDL 2.0 is interface centric.
- More polished and refrains from having cluttered features which are useless as well as dangerous to ReST. For example using cookies as state management mechanism is considered a bad practice because it endangers the stateless nature of ReST whilst it is supported in WSDL 2.0. On the other hand since ReST descriptions are processed by lightweight clients backed by limited scripting

languages, making descriptions complex is not appealing. ReST gained much popularity due to the same reason in contrast to classic SOAP based ones.

In short, WADL is simple and more polished toward ReST.

3.2. Reflexive languages

- Dynamic type information and reflection enables a running program to access its internal structure and behavior and also to programmatically manipulate that structure, thereby changing its behavior at runtime [12]. For example, a program can ask for the class of a given object, find the methods on that class as well as enumerating the parameters and then invoke any of those methods by passing proper arguments which are instantiated at runtime. Many modern languages like C# allow programmers to attach meta-information to data type structures and later retrieve them via reflection at runtime. These metadata are called attribute or annotation. AuraGen employs attributes along with reflexive type system to implement dynamic extraction of description and documentation of Web services. AuraGen can be applied to any platform that uses a reflexive and attribute oriented language.

3.3. Service Class

- ReST services are usually represented in form of a service class. Service classes are instantiated by a host application which delegates the service object to clients as a ReST service. Each class has methods that are mapped as resource handlers, the atomic building block of ReST Web services. As depicted in Figure 2, a ReST resource in WCF is mapped to a method via WebInvoke attribute. When newuser/john resource is requested by typing http://websrv/newuser/john into Web browser, the service framework automatically maps the request to CreateUser method of RegService class. The underlying framework uses UriTemplate property of WebInvoke class to perform the mapping. The name parameter is also mapped to john using the template parameter denoted using curly brackets in UriTemplate. This way the programmer is abstracted away from the burden of mapping the URL to a method and required parameters.

4. Design

4.1. Service Description Generation Algorithm

- In order to automatically generate the service description directly from service object we need to perform some mapping from service object to WADL elements. We start from service object and retrieve all the methods annotated by OperationContract attribute. These are the methods that handle resource request from clients. Then each method is inspected for type of parameter as well as return values. For each data type that is not supported by W3C XML natively, an entry is created in grammar container element in root element of WADL. A sample is shown in Figure 3. Since complex data structures typically correspond to representation of resources, a representation element is also created. The only remaining element of WADL is method. Each method element has two child element named request and response. Request element describes the way resource should be requested and response element models the result of request. Each method element exactly matches to methods of service object. Again using reflection we turn each method in service object into a method element. A method element has two attributes, name and id. The name attribute specifies the HTTP method that should be used for requesting the resource while the id is an arbitrary. We extract the name attribute from Method property of WebInvoke annotation and the value of id is set to the name of the related method in service object. Each request can have query_variable child elements that specify the parameters passed by client in the query string of request. To extract query_variable element we parse the UriTemplate property of WebInvoke and any phrase enclosed in curly brackets after the question mark character is turned into a query_variable element. The response element is created in accordance to return value of service method. Note that all the steps described above are performed at runtime so the resulting service descriptions are always synchronized with actual service object. Mappings are summarized in Table.

4.2. Service Documentation and Manual Generation

- We propose two new additions to standard attributes used for annotating service classes in order to turn service documentation generation to an automatic process. The first attribute will be used to attach a human readable documentation to service methods and related parameters as well as parameter parts in case of complex types. The second attribute would be used solely on service methods and will be used to provide samples to service methods. The former is called ResDoc and the latter ResSample. ResSample is an extra documentation that can assist programmers and mashup designers to start using service resources easily. The distinction is valuable when creating resource stubs for clients. For example a WADL to client stub generator may use the ResDoc to make the generated stub easier to use. On the other hand ResSample is useful for those who have just started using the service resource. AuraGen dynamically extracts those attributes at runtime and provides the documentation to clients in two different formats: WADL service description and standalone HTM. Rich clients that invoke ReST Web service can take advantage of embedded documentation in WADL to provide useful hints and quick lookups to users whilst HTML documentation are considered more complete and thorough manual, useful for starters as well as future references.

5. Implementation

Two evaluate the suitability and effectiveness of AuraGen we have developed a mashup composition tool based on automatically generated WADL. Through investigations in literature, we found that most research choose to use a proxy service as mashup composition tool. Using proxy servers introduces the bottleneck problem making mashup dependent to availability of proxy. Moreover due to the same origin policy [13], many implementations refrain from rich client JavaScript implementations. AuraGen mashup composer is an add-on for the application platform provided by Mozilla. Mozilla application framework is a XML based cross platform toolkit that employs XUL for UI rendering and JavaScript for binding logic to user-interface elements. AuraGen mashup composer assists users by retrieving the description from service operations and lists them as necessities of retrieval of a resource. This way user can bind the inputs of an operation to proper data with a browse like mechanism which greatly facilitates the process while minimizing the mismatch errors at runtime. As shown in Figure 4, resources provided by a ReST service are listed along with documentation, allowing user to commit a request to a resource. Note that AuraGen decides the correct order of retrievals by inspecting the availability of required inputs. Furthermore AuraGen retrieves the resource in parallel to others whenever possible to boost the performance. The piping mechanism is also designed with the entry level users in mind. The parameters for resource retrieval are listed in separated views. Mashup developer can decide to enter those parameters directly or can bind them to predefined values. Predefined values are kept in a customizable collection called user items. Any retrieval is marked with an instance name which refers to an object that contains the inputs and outputs of a service. User can select to view the retrieved resource in form of human interpretable form by checking the Render output to HTML option. If the service is a plain text or XML, AuraGen will display it using a generic HTML rendering style.

6. Conclusion

ReST Web service have attracted much attention by the blossom of mashup composition tools. Mashups are an ideal solution contributing to the flexible nature generate WADL service descriptions and documentation directly from service objects. This approach has the following advantages:

TABLE 1 MAPPING SERVICE OBJECT TO WADL ELEMENTS

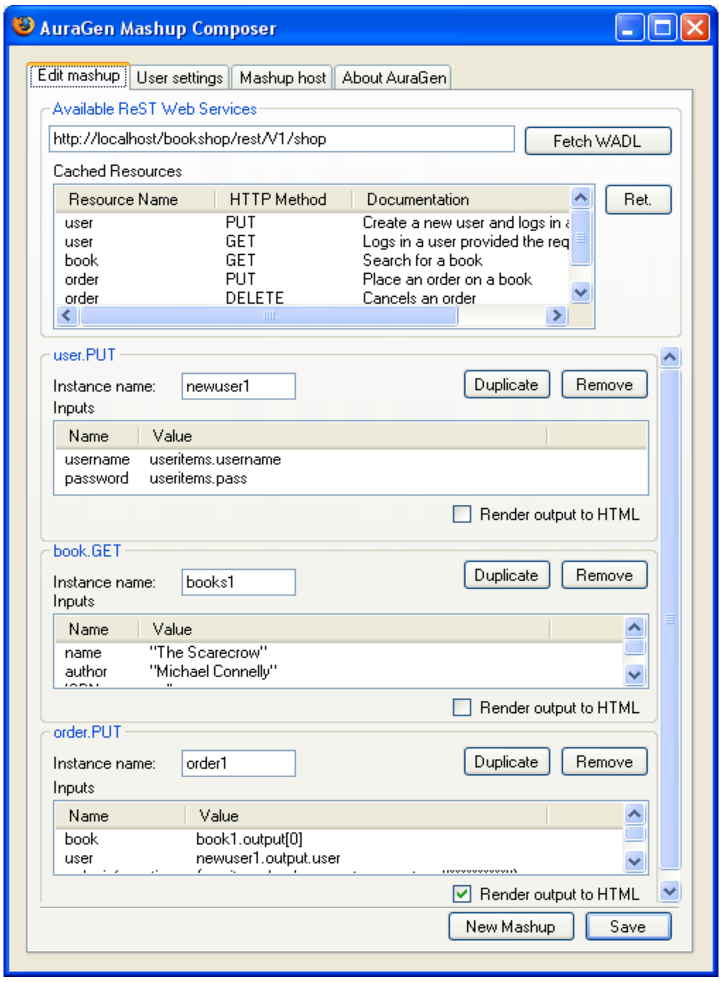
WADL Element	Service Object
grammar	Parameters and return values
resource	UtiTemplate
method	Methods annotated by OperationContract
request	Parameters of methods annotated by OperationContract
response	Return type of methods annotated by OperationContract
query_variable	UtiTemplate
documentation (AuraGen's addition to	ResDoc and ResSample

	WADL)

- Zero effort for service developers. Since the process is nearly invisible, the service developers are abstracted away from WADL service description. A developer only needs to provide the documentation in form of annotations on service objects via familiar attribute mechanism.
- Always in sync. The lazy generation of service description and manual documentation results in artifacts that are always synchronized with the real service. Changes to service are immediately reflected in service description and documentation which is rather slow and error prone in traditional scenarios in which service documentation is prepared by people other than those who developed the service.
- Facilitated mashup composition. Combining the machine readable service descriptions with human readable documentation can greatly benefit clients trying to weave Web services into what they want. AuraGen mashup composer supports the users by fetching and visualizing the ReST resources and guides them through the composition process and also detects the correct and optimum order in which ReST resources should be retrieved.

```
<xs:element name=" RegResult">
  <xs:complexType>
        <xs:attribute name="Succeeded" type="xs:boolean"/>
      <xs:attribute name="Error" type="xs:string"/>
  <xs:attribute name="UserId" type="xs:integer"/>
         </xs:complexType>
                </xs:element>
```

Figure 3. RegResult type represented in WADL



Available ReST Web Services

http://localhost/bookshop/rest/V1/shop

Cached Resources

Resource Name	HTTP Method	Documentation
user	PUT	Create a new user and logs in a
user	GET	Logs in a user provided the req
book	GET	Search for a book
order	PUT	Place an order on a book
order	DELETE	Cancels an order

user.PUT

Instance name: newuser1

Inputs

Name	Value
username	useritems.username
password	useritems.pass

☐ Render output to HTML

book.GET

Instance name: books1

Inputs

Name	Value
name	"The Scarecrow"
author	"Michael Connelly"

☐ Render output to HTML

order.PUT

Instance name: order1

Inputs

Name	Value
book	book1.output[0]
user	newuser1.output.user

☑ Render output to HTML

Figure 4. AuraGen mashup composer

Others have provided solutions for mashup composition use cases. hREST [14] and SA-REST [15] use embedded HTML mark-up for annotating and describing the ReST Web services. Our solution uses XML based WADL for the same purpose. Being a standard XML document, WADL is more easily processed by client applications which happen to be lightweight Web applications usually running inside a Web browser and limited to scripting languages. In addition, hREST and SA-REST need human readable documentation already available. In contrast, our research references the service object directly, eliminating the need for any additional artifact. In [15] Amit Sheth proposed a novel solution to semantic annotation of Web services. We also plan to add semantic annotations to WADL Web service descriptions (again at service object level) in a future work.

7. References

[1] X. Liu, Y. Hui, W. Sun, and H. Liang, "Towards service composition based on mashup," in IEEE Congress on Services, 2007, pp. 332-339.

[2] E. Griffin, "Introduction to Mashups " in Foundations of Popfly: Apress, 2008, pp. 1-20.

[3] R. Fielding and R. Taylor, "Principled design of the modern Web architecture," in International Conference on Software Engineering, 2000, pp. 407-416.

[4] http://www.restlet.org/, Lightweight REST framework.

[5] K. Scribner and S. Seely, Effective Rest Services Via. net: For. net Framework 3.5: Addison-Wesley Professional, 2009.

[6] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," May, 2000.

[7] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana, "Web services description language (WSDL) version 2.0 part 1: Core language," Technical report, World Wide Web Consortium 2007.

[8] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs." big"'web services: making the right architectural decision," in 17th international conference on World Wide Web, Beijing, China, 2008, pp. 805-814.

[9] R. Battle and E. Benson, "Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST)," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 6, pp. 61-69, 2008.

[10] M. Hammer and D. McLeod, "Semantic integrity in a relational database system," in Very Large Data Bases, NY, USA, 1975, pp. 25-47.

[11] M. Hadley, "Web Application Description Language (WADL)," Technical Specification, Sun Microsystems, 2006.

[12] G. T. Sullivan, "Aspect-oriented programming using reflection and metaobject protocols," Commun. ACM, vol. 44, pp. 95-97, 2001.

[13] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell, "Protecting browser state from web privacy attacks," in Proceedings of the 15th international conference on World Wide Web Edinburgh, Scotland: ACM, 2006.

[14] J. Kopecky, K. Gomadam, and T. Vitvar, "hRESTS: An HTML Microformat for Describing RESTful Web Services," Kno. e. sis tech. report, Wright State Univ, 2008.

[15] A. Sheth, K. Gomadam, and J. Lathem, "SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups," IEEE Internet Computing, vol. 11, pp. 91-94, 2007.