An Approach to Construct Dynamic Service Mashups Using Lightweight Semantics

Dong Liu, Ning Li, Carlos Pedrinaci, Jacek Kopecký, Maria Maleshkova, and John Domingue

Knowledge Media Institute, The Open University
Walton Hall, Milton Keynes, MK7 6AA, UK
{d.liu,n.li,c.pedrinaci,j.kopecky,
m.maleshkova,j.b.domingue}@open.ac.uk

Abstract. Thousands of Web services have been available online, and mashups built upon them have been creating added value. However, mashups are mostly developed with a predefined set of services and components. The extensions to them always involve programming work. Furthermore, when a service is unavailable, it is challenging for mashups to smoothly switch to an alternative that offers similar functionalities. To address these problems, this paper presents a novel approach to enable mashups to select and invoke semantic Web services on the fly. To extend a mashup with new semantic services, developers are only required to register and publish them as Linked Data. By refining the strategies of service selection, mashups can behave more adaptively and offer higher fault-tolerance.

Keywords: Mashup, Semantic Web Services, Service Selection, Service Invocation.

1 Introduction

More and more companies and organisations expose their core functionalities as SOAP or RESTful services on the Web, so that third-party developers can create new Web applications atop of these services in a more agile way. Repositories and marketplaces such as ProgrammableWeb[1], Seekda[2] and Mashape[3], have been established to collect and publish descriptions of Web services. On the other hand, mashups integrate data, services and contents available online into a coherent application that creates new value [20]. Tools such as Yahoo Pipes[4] and IBM Mashup Center[5], have been available for assisting the development of mashups.

[1] http://www.programmableweb.com
[2] http://webservices.seekda.com
[3] http://www.mashape.com
[4] http://pipes.yahoo.com/
[5] http://www.ibm.com/software/info/mashup-center/

A. Harth and N. Koch (Eds.): ICWE 2011 Workshops, LNCS 7059, pp. 13–24, 2011.

However, the mashups built with these tools are essentially static, i.e. depending upon a predefined set of APIs and components. This has an impact on the extendibility and fault-tolerance. Developers of a mashup have to work on the programming code, even if they just want to extend it with services or APIs offering similar functionalities. For instance, there are a few companies now offering local business searching services, e.g. Scoot API[6], Yahoo Local Search API[7], Yelp[8] and CityGrid[9]. New local business searching services might also come to the market at some point. Combining new business searching services to an existing mashup requires both hard coding and re-deployment.

Online services might be unaccessible for reasons such as expiration of API keys, connection failures, request timeout, etc. End-users will suffer from the long response time of the mashups built with these unaccessible services. One possible way to overcome this issue is to switch the mashups to other alternative services. For example, although most of the local business services cover different regions of the world, they (e.g. Yelp and Yahoo Local Search) may have some overlaps with each other. When one of them is off-line, the mashups can use the other one instead.

To address these issues, we propose a novel approach to build dynamic mashups using Web services with lightweight semantics. The UI components interact with unified interfaces of each kind of services, rather than invoking those services directly. iServe, together with its extensions, performs service selection and invocation behind those interfaces. The services to be invoked by the mashups through iServe are controllable and determined at runtime. Therefore, mashups built following our approach are more flexible and robust.

The rest of the paper is organised as follows: Section 2 discuss principles related to semantic services and mashups. Section 3 details the proposed approach to build dynamic service mashups. Section 4 summaries related work. Finally, Section 5 concludes this paper and highlights our future work.

2 Services, Mashups and Semantics

A Web service is a set of operations on resources, which are accessible online. Accordingly, there have emerged two types of Web services: operation-oriented services (e.g. SOAP services) and resource-oriented services (e.g. RESTful services). Both SOAP and RESTful services can be exploited as reusable building blocks for new applications. Efforts made to integrate Web services are regarded as service composition [15]. Service composition is usually performed on the business logic layer, and results in executable workflows or plans that fulfil certain requirements of the new Web application. Mashup is an innovative way to develop Web applications by syndicating contents, data and functionalities from distributed sources on the Web. Different from service composition, mashup can

[6] http://www.scoot.co.uk/about-us/add-scoot/reference.html
[7] http://developer.yahoo.com/search/local/V3/localSearch.html
[8] http://www.yelp.com/developers/documentation/v2/search_api
[9] http://docs.citygridmedia.com/display/citygridv2/Places+API

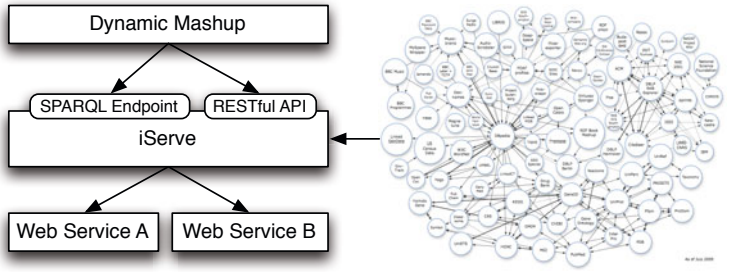

Fig. 1. Mashup model

be carried out on layers ranging from data to presentation. Rather than enacting a predefined workflow, components of a mashup are more loose-coupled, and can interact with end-users. In this paper, we focus on service mashups that are aim to bring them together mashups with easy-to-accomplish end-user service compositions [3].

Semantic technologies have been adopted to automate service annotation, discovery, composition and invocation [1]. Ontological models such as OWL-S [10] and WSMO [16] provide formal languages for semantically describing Web services, whereas annotation-based approaches (e.g. SAWSDL [19], hRESTS [7]) enable the creation of lightweight semantic Web services. We employ Minimal Service Model (MSM)[10] to capture the semantics of SOAP and RESTful services, which are essential for selection and invocation of hybrid services. In MSM, a `Service` is defined as a set of operations plus links to functional classifications and non-functional properties. An `Operation` is an atomic unit to be invoked, having properties like input messages, output messages, addresses, faults, etc. Instances of `MessageContent` are containers of the input and output messages exchanged during the invocation of services. A `MessageContent` may comprise a hierarchy of `MessagePart`. Additionally, `modelReference` borrowed from SAWSDL enables the linking of service elements to semantic models via URIs, while `liftingSchemaMapping` and `loweringSchemaMapping` are used to specify data transformations from a syntactic representation to its semantic counterpart and vice versa.

From all above, dynamic mashup is defined as a Web application implemented by selecting and invoking Web services described using MSM. Figure 1 shows the conceptual model of dynamic mashup. By querying against iServe's SPARQL endpoint, a dynamic mashup selects some relevant APIs, and then quests iServe to invoke them. Service invocation through iServe is always in a RESTful way, and some of the inputs might be from the Web of Data. Some of the key features of dynamic mashups are outlined as follows:

– **Dynamics** Mashups can determine which services to invoke on the fly.

[10] `http://cms-wg.sti2.org/ns/minimal-service-model`

- **Transparency** The technical details of service selection and invocation are transparent for the UI components of a mashup.
- **Configurability** Maintainers can easily control the behaviours of a mashup just by revising the strategy of service selection, yet without programming work.
- **Extendibility** To integrate more services offering similar functionalities, developers only need to formally describe those services, and publish them on the Web.
- **Robustness** When a service is temporarily unavailable, a dynamic mashup can smoothly switch to the alternatives.

3 Building Dynamic Mashups

In our previous work [13], we have presented iServe, a public registry for semantic services. It can import service descriptions conforming to heterogeneous schemas, and publish them as Linked Data on the Web. iServe exposes a set of Web APIs for manipulating the published service descriptions[11], as well as for service discovery on higher level of abstraction [12].

The proposed approach to build dynamic mashups centres upon iServe and its extensions for service invocation [8]. This section elaborates how to construct dynamic mashups by taking advantage of iServe's capabilities of service discovery and invocation, and also by following the steps listed below.

- **Semantic Services Authoring** This step includes 1) annotating service descriptions with concepts of the MSM and domain ontologies; 2) publishing them as Linked Data via iServe.
- **Specifying Strategies of Service Selection** This step can be done by either exploiting iServe's built-in service discovery mechanisms, or writing SPARQL queries to be executed against the RDF dataset of iServe.
- **Defining Lowering and Lifting Schema Mappings** This step outcomes XSPARQL [2] queries for translating RDF triples into parameters used to invoke services, and also for rewriting the invocation results as RDF statements.
- **Merging Service Invocation Results** This step deals with issues regarding to put together invocation results from different sources, e.g. eradicating any duplicated items, sorting by specific properties, etc.

In order to demonstrate the workflow of building a dynamic mashup, an example is given in this section, which visualises the local business search results on a map (see Figure 2). Besides the Web APIs for local business searching mentioned previously, the mashup also makes use of Google map API[13] and Google Web Toolkit[14].

[11] `http://iserve.kmi.open.ac.uk/wiki/index.php/IServe_RESTful_API`
[12] `http://iserve.kmi.open.ac.uk/wiki/index.php/IServe_Higher_Level_Discovery_API`
[13] `http://code.google.com/apis/maps/`
[14] `http://code.google.com/webtoolkit/`

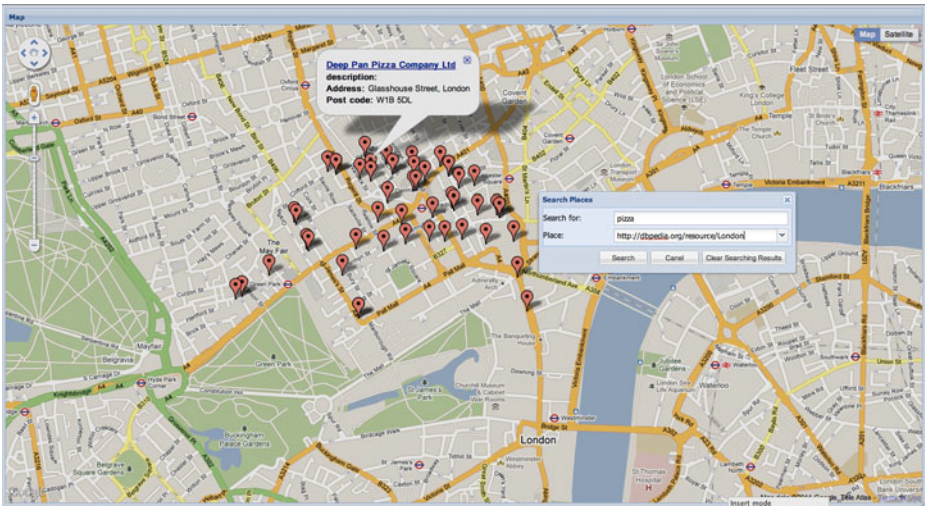

Fig. 2. Screenshot of the example of dynamic mashup

3.1 Semantic Services Authoring

Essentially, semantic services authoring is to add annotations to the original documents of service descriptions, so as to make them more understandable for machines. Tools such as SWEET and SOWER [9], have been developed to facilitate annotating both HTML and WSDL files. Although authors can arbitrarily annotate services descriptions, we argue that semantic services will be easier to be discovered and invoked, if being annotated following principles and patterns shown below.

- Service categories should be attached to services rather than operations or messages. This can simplify service discovery based on functional classifications.
- The addresses and types of HTTP methods, e.g. GET, POST, PUT, etc., should be declared, otherwise operations will not be able to be invoked.
- Information related to groundings of input messages should be provided.
- Lowering schema mappings must be associated with input messages. When an input message has a hierarchical structure, lowering schemas are usually utilised to annotated message parts on the lowest level. Section 3.3 shows how this can help in the preparation for service invocation.
- In principal, lifting schema mappings are for output messages only.
- In many cases, messages are annotated with concepts of domain ontologies, while their sub-parts are annotated with properties of such concepts. This can ensure the alignment of formal semantics of input/output messages and the ontological knowledge. In addition, it also gives hints on writing and understanding the lifting and lowering schema mappings.

For instance, Listing 1 shows the aforementioned Scoot API described in RDF, using the vocabulary of MSM and several domain ontologies such as DBpedia ontology[15], Service Categories ontology[16] and W3C WGS84 vocabulary[17].

```
service:Scoot a msm:Service;
   msm:hasOperation operations:search;
   sawsdl:modelReference finder:InternetSearch .
operations:search a msm:Operation;
   msm:hasInput inputs:query, inputs:place ;
   msm:hasOutput outputs:result ;
   hrests:hasMethod "GET" ;
   hrests:hasAddress "http://www.scoot.co.uk/api/find.php?format=xml&what={p1
        }&lat={p2}&long={p3}" .
inputs:query a msm:MessageContent;
   sawsdl:modelReference rdf:Literal ;
   hrests:isGroundedIn "p1" .
inputs:place a msm:MessageContent;
   sawsdl:modelReference dbp-ont:Place ;
   msm:hasPart types:lat , types:lng .
outputs:result a msm:MessageContent;
   msm:hasPart types:result-item .
types:lat a msm:MessagePart;
   sawsdl:modelReference geo-pos:lat ;
   sawsdl:loweringSchemaMapping lowerings:lat ;
   hrests:isGroundedIn "p2" .
types:lng  a msm:MessagePart;
   sawsdl:modelReference geo-pos:long ;
   sawsdl:loweringSchemaMapping lowerings:lng ;
   hrests:isGroundedIn "p3" .
types:result-item a msm:MessagePart;
   sawsdl:modelReference dbp-ont:Place ;
   sawsdl:liftingSchemaMapping liftings:result-item .
```

Listing 1. Description of Scoot API in RDF

As depicted by Listing 1, Scoot API is assigned to the category of Internet Search. It has one operation called "search", which takes keywords and an instance of `dbp-ont:Place` as inputs, and returns a list of relevant local businesses also as instances of `dbp-ont:Place`. Note that the input message `inputs:place` has one model reference `dbp-ont:Place` and two sub-parts `types:lat` and `types:lng`. And, the model references of `types:lat` and `types:lng` are respectively `geo-pos:lat` and `geo-pos:long`, which are two properties of the concept `dbp-ont:Place`.

3.2 Service Selection

This sub-section focuses on SPARQL-based service selection, which enables the on-the-fly refinement of the selection strategies. Listing 2 gives an example seeking for the services used to implement the mashup mentioned before, i.e. those under the category of "Internet Search", taking a `rdf:Literal` value and an instance of `dbp-ont:Place` as inputs, and returning instances of `dbp-ont:Place` as outputs.

[15] `http://wiki.dbpedia.org/Ontology`
[16] `http://www.service-finder.eu/ontologies/ServiceCategories`
[17] `http://www.w3.org/2003/01/geo/wgs84_pos`

```
SELECT DISTINCT ?s WHERE {
    ?s rdf:type msm:Service .   ?s sawsdl:modelReference ?c .
    ?c rdfs:subClassOf finder:InternetSearch .
    ?s msm:hasOperation ?o .    ?o msm:hasInput ?in1 .
    ?in1 sawsdl:modelReference rdf:Literal .
    ?o msm:hasInput ?in2 .  ?in2 sawsdl:modelReference ?in2mr .
    dbp-ont:Place rdfs:subClassOf ?in2mr .
    ?o msm:hasOutput ?out . ?out msm:hasPart ?outpart .
    ?outpart sawsdl:modelReference ?outmr .
    ?outmr rdfs:subClassOf dbp-ont:Place .
}
```

Listing 2. SPARQL query for service selection

By means of rewriting the SPARQL query above, the mashup can behave more adaptively, namely, dynamically choose the services to invoke. Three examples for the typical usage are listed as follows. Mashup developers can create more complex queries to satisfy their own requirements.

`FILTER (?s != service:Scoot)`
: When the Scoot service is now unavailable, this filter can avoid the attempts to invoke it. In this way, it also can meet the requirement for smoothly switching between services at runtime.

`?o hrests:hasAddress ?addr FILTER regex(str(?addr), ".uk") .`
: This clause with the regular expression can select services having addresses that contains ".uk", i.e. services provided by companies registered in the UK.

`LIMIT 3`
: The solution sequence modifier `LIMIT` can restrict the number of services to invoke, so as to reduce the response time of the mashup.

3.3 Service Invocation

The overall process of service invocation includes dereferencing, lowering, grounding, invoking and lifting. When identifiers of resources on the Web of Data are used as parameters for invoking services, iServe will first attempt to retrieve RDF triples describing those resources, i.e. dereferencing the resources. After that, RDF statements are lowered to literal values by executing XSPARQL queries. Those values are then used to instantiate requests to be sent to the service endpoints. Grounding refers the instantiation of service requests, which is the last step of the preparation for the actual invocation of services. After receiving the results in the format of XML, another set of XSPARQL queries will be executed to transform them into RDF.

As stated in the beginning of Sectionn 3, iServe provides RESTful APIs for publishing and removing the descriptions of semantic services. And, all the services stored in iServe are to be invoked as RESTful APIs. Therefore, an action resource [6], named "invoke", has been added to each operation of the services. In other words, the template of the addresses to invoke services stored in iServe is:

`http://.../services/service-id/operations/operation-id/invoke?`
`parameter1=`V_1`&parameter=`V_2`&...`

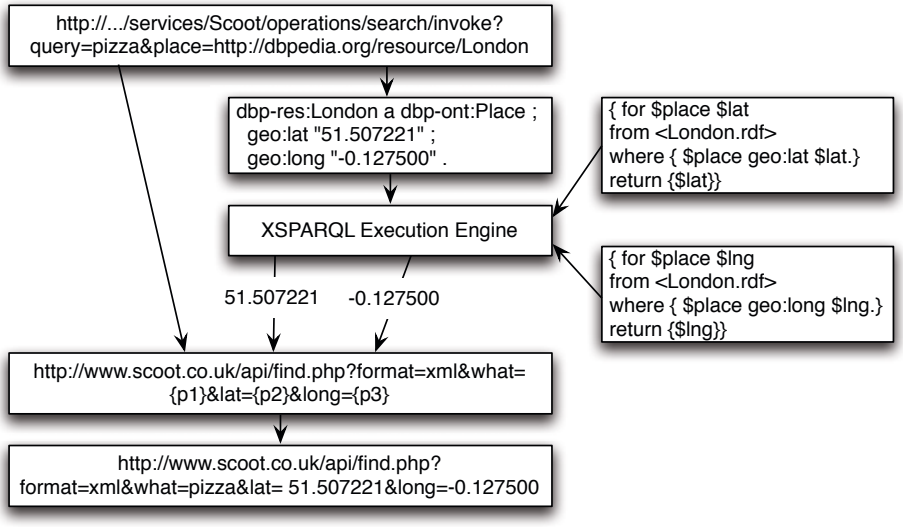

Fig. 3. An example for preparing the service request

Figure 3 illustrates the preparation for calling of the search operation of the Scoot service. iServe first found a dereferenceable URI, `http://dbpedia.org/resource/London` in the original request, and got a piece of RDF by sending there an HTTP `GET` with header `Accept: application/rdf+xml`. Then, the XSPARQL query engine loaded that piece of RDF as well as the lowering schema mappings shown on the right hand side of Figure 3, and extracted the latitude and longitude of London. Finally, variables in the URI template were replaced with the query keywords and the values of latitude and longitude. Part of the raw XML file returned by Scoot API is shown in the upper left of Figure 4, while the XSPARQL query guided the lifting of service invocation results is in the upper right. And, some of the generated RDF triples are shown in the lower part of Figure 4.

3.4 Extensions to Existing Mashups

As stated in Section 1, one of the key features of dynamic mashups is the extendibility. Developers can integrate new semantic services to built-up mashups without efforts on the modification of the source codes. Taking as an example CityGrid[18], another local business searching service, the following things have to be done to ensure being found by executing the SPARQL query in Section 3.2: firstly, put it into the category of Internet Search by adding a model reference to the service; secondly, use the DBpedia ontology and W3C WGS84 vocabulary to annotate the service description. Moreover, to make it invocable, developers have

[18] `http://docs.citygridmedia.com/display/citygridv2/Places+API`

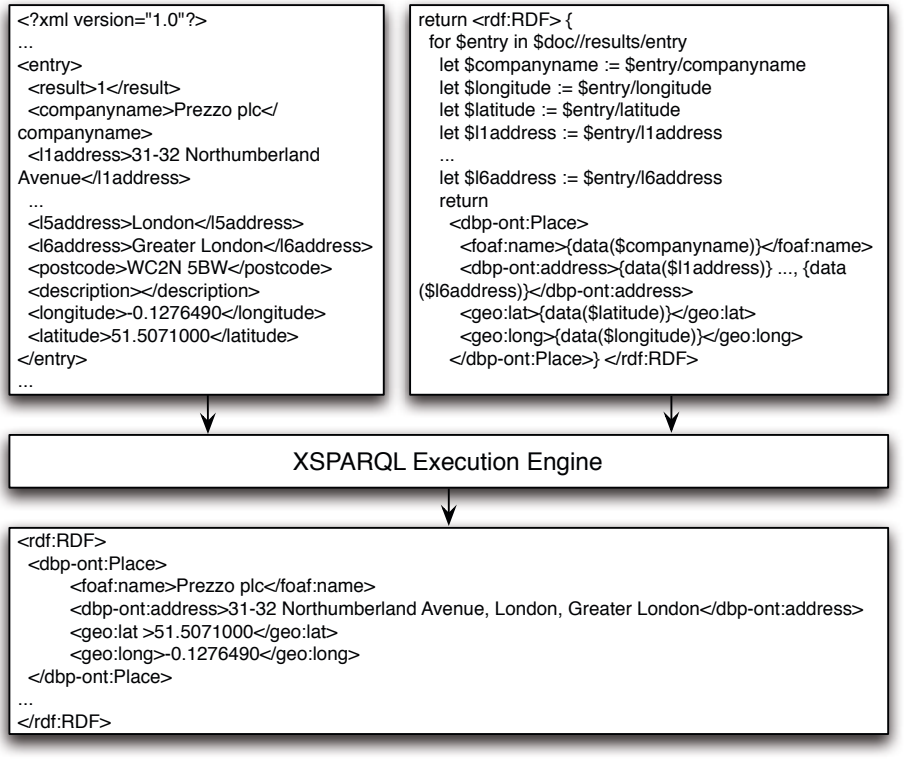

Fig. 4. An Example for Lifting

to specify the lowering and lifting schema mappings for the CityGrid service. The lowering schema for Scoot API is particularly reusable in this case. Therefore, the developers only need to write the XSPARQL query (see Listing 3) for data lifting on the basis of analysis on the sample results of invoking the CityGrid service.

```
return <rdf:RDF> {
 for $entry in $doc//locations/location
  let $name := $entry/name
  let $address := $entry/address
  let $street := $address/street
  let $city := $address/city
  let $state := $address/state
  let $postal_code := $address/postal_code
  let $longitude := $entry/longitude
  let $latitude := $entry/latitude
   return <dbp-ont:Place>
      <foaf:name>{data($name)}</foaf:name>
      <dbp-ont:address>{data($street)}, {data($city)}, {data($state)}, {data(
          $postal_code)}</dbp-ont:address>
      <geo:lat>{data($latitude)}</geo:lat>
      <geo:long>{data($longitude)}</geo:long>
  </dbp-ont:Place>} </rdf:RDF>
```

Listing 3. XSPARQL query for lifting CityGrid invocation results

All the results of service invocation are transformed into RDF through data lifting, and they conform to the same ontology. Thus, developers can easily merge them together by adding them to a common RDF model before serialising and sending them to the client side.

4 Related Work

Several platforms have been established to facilitate the design and development of mashups. For instance, IBM *Sharable Code* is an online platform to support the whole life-cycle of Web APIs and service mashups [11]. Mashup developers are required to use Domain Specific Language (DSL) to specify data mediation, process mediation and UI customisation. A lightweight framework, Mashlight, is proposed in [5], which is composed of four components: *Block Builder*, *Block Library*, *Mashup Builder* and *Run-time Engine*. Developers can use the Block Builder to encapsulate functionalities as Mashlight Blocks, and save them into the Block Library. Mashup Builder is a visual tool for defining the workflow, and the Run-time Engine is the execution environment to enact the mashups. The overall architecture of Mashlight is similar to our work, but lack of explicit semantics and effective discovery and selection mechanisms.

MatchUp built on top of the IBM Mashup Center provides a solution to the mashup autocompletion [4]. The proposed autocompletion algorithm can recommend relevant components and the connections between them to help users building mashups in a more convenient and intuitive way. Comparing with MatchUp, our approach intends to enable mashups to automatically select and invoke services at runtime, rather than design time.

In the context of Web services, RESTful service composition is another related topic to our work. Bite, a lightweight and executable language for RESTful service composition, is proposed in [17]. Bite offers a basic set of language constructs for specifying the business logics of Web-scale workflows, and inherits concepts from scripting languages such as dynamic data types. Bite has four runtimes to satisfy different requirements. In contrast to introducing a new language, minor extensions are made to BPEL for the composition of RESTful services [12]. Efforts also have been made to automate the process of RESTful service composition [21].

The work stated above adopts little semantic technology. SA-REST, an annotation-based approach to add semantics to RESTful services, is briefly described in [18]. Mashups created using SA-REST, denoted with *smashup* (semantic mashup), are to be hosted at a proxy server together with domain ontologies. Similar to our approach, the data mediation is carried out through lowering and lifting, but implemented with XSLT. To my best knowledge, SA-REST does not address the issues of service modelling, registry and discovery.

Apart form SA-REST, *Semantic Web Pipes* (SWP) proposed in [14] is a rapid prototype method of semantic mashups on the data layer. SWP provides a set of operators for merging, splitting and transforming RDF triples in the cloud of Linked Data. It extends SPARQL with workflows, and implements dynamic

transformations of RDF data using XQuery and XSPARQL. Our approach also follows the principles of Linked Data and applies SPARQL, XSPARQL for data mediation.

5 Conclusions and Future Work

In this paper, we present a novel method of building mashups using Web services with lightweight semantics, which is implemented based on iServe and its extensions for service invocation. By applying our approach, mashups gain the ability of selecting and invoking semantic services. Moreover, developers can easily extend a mashup without programming work, as well as switch it to the alternatives with a service is unaccessible. In short, our approach is effective for building mashups more flexible, robust and extendible.

Our future work will involve realising the caching mechanisms for service selection, and filtering services by QoS parameters, e.g. availability, response time, throughput, etc. In addition, we will also focus on the context-awareness of mashups, i.e. automatically change the strategies of service selection according the running state of the involving services. For example, when a service is down, mashups can be aware of it and automatically switch to alternative ones with high semantic similarity.

Acknowledgements. This work is partly funded by the EU project SOA4All (FP7-215219) and NoTube (FP7-231761). The authors would like to thank the European Commission for their support.

References

1. Agarwal, S., Handschuh, S., Staab, S.: Annotation, composition and invocation of semantic web services. Web Semantics 2(1), 31–48 (2004)
2. Akhtar, W., Kopecký, J., Krennwallner, T., Polleres, A.: XSPARQL: Traveling Between the XML and RDF Worlds – and Avoiding the XSLT Pilgrimage. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 432–447. Springer, Heidelberg (2008)
3. Benslimane, D., Dustdar, S., Sheth, A.: Services Mashups: The New Generation of Web Applications. IEEE Internet Computing 12(5), 13–15 (2008)
4. Greenshpan, O., Milo, T., Polyzotis, N.: Autocompletion for Mashups. Proceedings of the VLDB Endowment 2(1), 538–549 (2009)
5. Guinea, S., Baresi, L., Albinola, M., Carcano, M.: Mashlight: a Lightweight Mashup Framework for Everyone. In: Proceedings of 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009) at WWW 2009 (2009)
6. Hadley, M., Pericas-Geertsen, S., Sandoz, P.: Exploring Hypermedia Support in Jersey. In: Proceedings of the First International Workshop on RESTful Design, WS-REST 2010, pp. 10–14. ACM, New York (2010)
7. Kopecký, J., Gomadam, K., Vitvar, T.: hRESTS: an HTML Microformat for Describing RESTful Web Services. In: The 2008 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2008). IEEE CS Press (2008)

8. Li, N., Pedrinaci, C., Kopecký, J., Maleshkova, M., Liu, D., Domingue, J.: Towards Automated Invocation of Web APIs. In: Poster at the 8th Extended Semantic Web Conference, ESWC 2011 (to appear, 2011)
9. Maleshkova, M., Pedrinaci, C., Domingue, J.: Supporting the Creation of Semantic RESTful Service Descriptions. In: Workshop: Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2) at 8th International Semantic Web Conference (2009)
10. Martin, D., Paolucci, M., McIlraith, S.A., Burstein, M., McDermott, D., McGuinness, D.L., Parsia, B., Payne, T.R., Sabou, M., Solanki, M., Srinivasan, N., Sycara, K.: Bringing Semantics to Web Services: The OWL-S Approach. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 26–42. Springer, Heidelberg (2005)
11. Maximilien, E.M., Ranabahu, A., Gomadam, K.: An Online Platform for Web APIs and Service Mashups. IEEE Internet Computing 12(5), 32–43 (2008)
12. Pautasso, C.: RESTful Web Service Composition with BPEL for REST. Data and Knowledge Engineering 68(9), 851–866 (2009)
13. Pedrinaci, C., Liu, D., Maleshkova, M., Lambert, D., Kopecký, J., Domingue, J.: iServe: a Linked Services Publishing Platform. In: Proceedings of Ontology Repositories and Editors for the Semantic Web at 7th ESWC (2010)
14. Phuoc, D.L., Polleres, A., Tummarello, G., Morbidoni, C., Hauswirth, M.: Rapid Semantic Web Mashup Development through Semantic Web Pipes. In: Proceedings of the 18th World Wide Web Conference (WWW 2009), Madrid, Spain, pp. 581–590 (2009)
15. Rao, J., Su, X.: A Survey of Automated Web Service Composition Methods. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 43–54. Springer, Heidelberg (2005)
16. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology 1(1), 77–106 (2005)
17. Rosenberg, F., Curbera, F., Duftler, M.J., Khalaf, R.: Composing RESTful Services and Collaborative Workflows: A Lightweight Approach. IEEE Internet Computing 12(5), 24–31 (2008)
18. Sheth, A.P., Gomadam, K., Lathem, J.: SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. IEEE Internet Computing 11(6), 91–94 (2007)
19. W3C: Semantic Annotations for WSDL and XMLSchema (2007), `http://www.w3.org/TR/sawsdl/`
20. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding Mashup Development. IEEE Internet Computing 12(5), 44–52 (2008)
21. Zhao, H., Doshi, P.: Towards Automated RESTful Web Service Composition. In: Proceedings of 7th IEEE International Conference on Web Services(ICWS 2009), pp. 189–196. IEEE Computer Society (2009)