ENRICHING HETEROGENEOUS SERVICE COMPOSITION WITH A SEMANTICALLY ENHANCED SERVICE DESCRIPTION LANGUAGE

Feifei Hang, Liping Zhao
School of Computer Science
The University of Manchester
Manchester, UK
{hangf, lzhao}@cs.man.ac.uk

Abstract

The emergence of Web 2.0 and its related technologies such as HTML5 has empowered end-users and made it possible for them to compose their own Web applications. Yet, most of the current development has mainly concentrated on the support of the composition of enterprise-oriented services and scientific workflows and little effort has been made to support the composition of end user-oriented services. In addition, the lack of machine-readable and high-level composite service description languages has prevented the end-users from sharing the service composition knowledge. To overcome these limitations, this paper introduces "HyperMash", a service composition approach for end-users. HyperMash supports the composition of both RESTful and SOAP-based Web services, and allows both types of service to be freely combined. A description language, called "Semantic-UiSDL", is used to automatically generate machine-readable and processable descriptions of composite services. Through this language, HyperMash can provide service recommendations to end-users as a way of sharing and reusing their service composition knowledge. This paper presents and illustrates the HyperMash approach and its major concepts and components through real-life examples and empirical study.

Keywords: Heterogeneous Service Composition; RESTful Web Services; SOAP-Based Web Services; Mashup; HyperMash; Semantic-UiSDL; Semantic Web; End-User Development

1. INTRODUCTION

With the advent of Web 2.0 and its related technologies such as HTML5, more and more non-professional programmers (thereafter called the "end-users" for short) have participated in the Web development by creating their own applications based on existing Web resources and services. In the past years, these users have created a large number of widget-based "mashups", typically in the form of Web pages, which combine, visualize or aggregate other Web services (Wikipedia, 2013). Web browsers have been the most popular means for end-users to access Web resources and consume Web services (NetMarketShare, 2013).

To date, support for service mashups by end-users has mainly focus on creating dashboard-like Web pages such as personal Web portals, for displaying different types of information, such as weather forecast, stock price report and online world clock. By contrast, little effort has been made to support end-users for creating data-oriented, ad-hoc Web service mashups, such as travel navigators and personal work plans. Our study shows that this shortfall is caused by four major limitations of current service composition approaches. The first limitation, as stated by Obrenovic and Gasevic (2008), is that most of the existing development environments for service-oriented solutions are not appropriate for end-users as they require the expertise of the professional programmers.

The second limitation, according to Hoang et al. (2010), is that most of today's service composition approaches only support either RESTful or SOAP-based Web services. Moreover, SOAP-based Web services are mainly developed for the enterprise applications. According to the statistics provided by Programmable Web (www.programmableweb.com), 22% of the Web services are SOAP-based. For example Programmable Web and WebserviceX.Net (www.webservicex.net) provide a large number of SOAP-based Web services such as postcode finders, weather forecast and currency converters, which have the great potential to be composed by end-users to assist their daily work or personal needs. Yet, the lack of support for end-user friendly composition approaches has restricted these services from being fully used by individual users.

Third, although a number of data-oriented service composition platforms are available, they only support a limited number of composition features. For example, the users can compose RESTful Web services by using Yahoo! Pipes (Yahoo!, 2013). However, until now, most of the existing Web-based service composition platforms only provide end-users with features for collecting and aggregating the data retrieved from Web services. For example, RESTful Web services expose their functions through a set of uniform interfaces: GET, POST, PUT, and DELETE (Fielding, 2000; Richardson & Ruby, 2007). But, Yahoo! Pipes only supports the GET interface. Consequently, it restricts the users from composing services by using other interfaces.

Finally, most of the existing Web-based service composition platforms do not automatically generate semantic descriptions for user defined composite services. In Yahoo! Pipes, for example, if an end-user is to compose a composite service, he needs to manually write the description of his composite service. According to Danielsen and Jeffrey (2013), this shortfall can result in at least two drawbacks: First, without automatically generated semantic descriptions, the service composition platform will not be able to understand the relationships between the composed services and will therefore not be able to automatically reuse the composition knowledge from the existing composite services. Second, manual descriptions of composite services may be incomplete.

This paper presents our attempt at addressing these four limitations. We describe HyperMash, a heterogeneous service composition approach with semantic enhancement. HyperMash attempts to overcome the first limitation by providing a Web-based, end-user friendly composition platform, which enables end-users to specify their desired services through a workflow diagram. To overcome the second limitation, HyperMash supports the composition and combination of both RESTful and SOAP-based services. To overcome the third limitation, HyperMash provides the full set of RESTful interface features. To address the fourth limitation, HyperMash uses a semantic composite service description language, called Semantic-UiSDL, for automatically describing user-defined composite services.

HyperMash intends to make two major contributions to service composition and description:

- The support of an on demand heterogeneous service composition platform, which allows end-users to create their own composite services by combining RESTful services with SOAP-based services at runtime.
- The provision of a semantically enhanced composite service description language (i.e. Semantic-UiSDL), which enables the sharing and reusing of the existing service composition knowledge.

To validate and demonstrate HyperMash, we have developed a prototype system and successfully tested this system on a large number of examples.

This paper proceeds as follow: Section 2 discuses some of closely related work. Section 3 presents the architectural framework of the HyperMash approach. Section 4 and 5 present the underlying concepts, enabling technologies, and working principles of two major HyperMash system components - Service Recommender and Service Composer - respectively. The HyperMash prototype is illustrated in Section 6, whereas the accuracy of service recommendation in HyperMash is evaluated in Section 7. Finally, Section 8 draws some conclusions to our work.

2. RELATED WORK

To further highlight the limitations of current service composition approaches, this section reviews some well-known work in the field.

2.1 SERVICE COMPOSITION APPROACHES

With the development of Web 2.0 and SOA, a great number of service composition approaches have been developed. In this paper we present and discuss the strengths and weaknesses of some of the existing approaches in the contexts of service composition and end-user development (EUD).

AMICO:CALC. To support end-users with native calculating abilities, several spreadsheet-based approaches (Hoang et al., 2010) have been proposed over the past years. As stated by Obrenovic and Gasevic (2008), AMICO:CALC has been designed as a plugin to several existing spreadsheet systems to provide users the ability of composing services by means of creating spreadsheets. When building an ad-hoc composite service, users are required to fill table cells with the predefined AMICO:CALC formulas. Meanwhile, the original formulas of each spreadsheet system are remained in the approach, which means that users can also manipulate the received data from the component services by using the native formulas and functions such as finding the minimum and maximum value among all.

However, we identified the following two major shortcomings of AMICO:CALC:

- Due to the nature of spreadsheet, users have to learn a certain number of formulas before being able to utilize them in composing Web services. In other words, as concluded by Hoang et al. (2010), this spreadsheet-based approach can only be helpful to skilled users rather than novices.
- To use AMICO:CALC, users have to install corresponding plugins to extend the original spreadsheet systems to gain the ability of composing Web services. However, it would increase the barrier of using AMICO:CALC as plugin installations are strictly not allowed on many types of mobile devices.

Yahoo! Pipes. As a well known and end-user friendly approach, Yahoo! Pipe (Yahoo!, 2013) allows users to create ad-hoc composite services by drawing workflow diagram. Since Yahoo! Pipes becomes a very developed service composition platform over the past few years, it is nowadays being recognized as a benchmark approach by a large number of researchers and developers.

By using “pipes”, users can easily aggregate data provided by different services in Yahoo! Pipes. However, we witnessed the weaknesses of this approach in the following two aspects:

- Because Yahoo! Pipes is primarily designed to compose RESTful services, users are not able to consume SOAP-based services in Yahoo! Pipes.

Table 1. A Comparison of Some Common Service Composition Approaches

Approach	**User group**			**Web services & features supported**			**Cross-device**
					RESTful		
	IT-expert	Skilled user	End-User	SOAP-based	Uniform Interfaces	HATEOAS	
AMICO:CALC	X	X	-	X	-	-	-
SOA4ALL	X	X	-	X	O	N/A	O
Jopera	X	X	-	-	X	-	-
Yahoo! Pipes	X	X	-	-	O	-	X
SOA Extension with Mashup	X	X	-	X	-	-	N/A
Marmite	X	X	X	-	O	-	N/A
DashMash	X	X	X	-	O	-	N/A
IBM DAMIA	X	X	-	-	O	-	N/A

(X) means the dimension is directly supported or required; (O) means the dimension is partially supported; (-) means the dimension is not supported; (N/A) means the dimension is not mentioned in related publication.

- Due to the lack of the full range support of RESTful interfaces, Yahoo! Pipes can only access Web services through the HTTP verb "GET".

Other Approaches. To support end-users, Marmite (Wong & Hong, 2007) and DashMash (Cappiello et al., 2011) are proposed to facilitate the creation of ad-hoc composite services. As can be seen from Table 1, both of these approaches are focusing on RESTful Web services. Yet, only an incomplete set of HTTP verbs is supported by these approaches. In other words, none of them provides sufficient features to support the full feature set of RESTful services.

Similarly, IBM DAMIA (Simmen, Altinel, Markl, Padmanabhan, & Singh, 2008) also provide similar functions and support to cover a limited set of RESTful features. However, it requires the expertise of professional users to utilize this approach. By contrast, JOpera (Pautasso, 2009) is proposed to provide full range of support to all the HTTP verbs used by RESTful services. However, as an Eclipse-based system, JOpera cannot be used on many types of mobile devices.

Meanwhile, SOA Extension with Mashup (Liu, Hui, Sun, & Liang, 2007) is proposed to support the composition of SOAP-based services. According to Liu et al. (2007), end-users are once again neglected by this approach.

SOA4ALL (Krummenacher, Norton, Simperl, & Pedrinaci, 2009; Lecue, Gorronogoitia, Gonzalez, Radzimski, & Villa, 2010) makes the giant leap towards the combination of both RESTful and SOAP-based Web services. However, the approach requires users to be at least skilled and does not support all the RESTful interfaces.

The approaches discussed in this section are summarized in Table 1.

2.2 Service Description Languages

Service descriptions are important part of Web services as they describe the specifications, behaviors and other aspects of the services. Service descriptions can be either machine generated or human-crafted (Danielsen & Jeffrey, 2013).

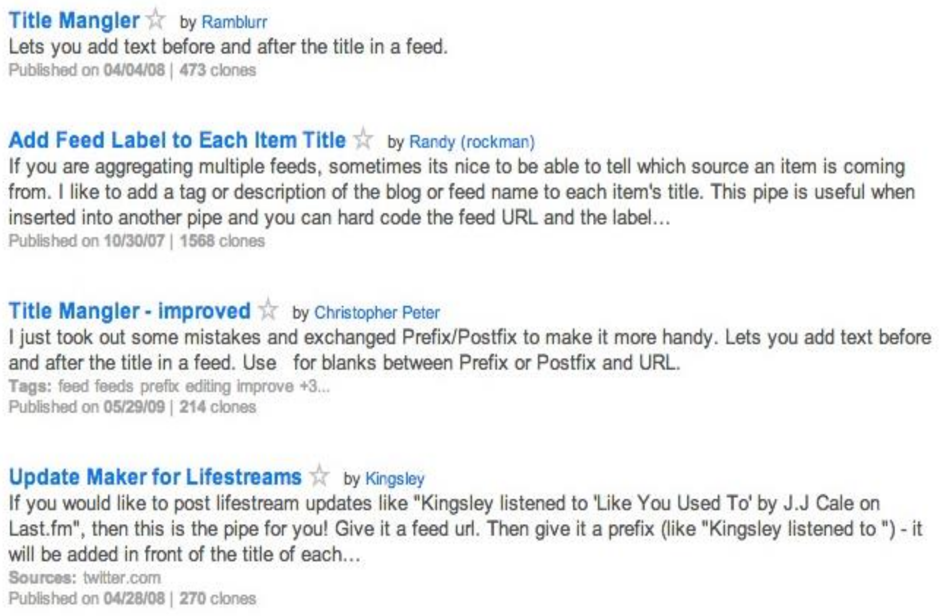
With the development of Semantic Web, more and more companies and organizations have started to enhance their originally human-readable service descriptions by adding machine-readable semantic elements. However, this shift always requires the expertise of professional programmers and therefore could be extremely challenging for end-users.

In this section, we present five commonly used service description languages.

Natural Language. In the realm of end-user service composition, natural language is playing a major role for describing services. Due to the nature of natural language, people can easily describe a service in their native language or even whatever language they can use. Figure 1 shows a set of composite services that described in natural language by end-users on Yahoo! Pipes.

However, as stated by Danielsen and Jeffrey (2013), natural language description are primarily written for human. In other words, as claimed by Semantic Web community (Bizer, 2009; Torma S, March 2008), natural language descriptions normally are not machine-readable, which means machines, precisely computers, can learn nothing from those human-readable descriptions, especially the relationships between each component services.

WSDL & WADL. As a very well developed and mature service description language, WSDL (W3C, 2001) allows developers describing SOAP-based Web services systematically. Based on XML, this structured language



Title Mangler ☆ by Ramblurr
Lets you add text before and after the title in a feed.
Published on 04/04/08 | 473 clones

Add Feed Label to Each Item Title ☆ by Randy (rockman)
If you are aggregating multiple feeds, sometimes its nice to be able to tell which source an item is coming from. I like to add a tag or description of the blog or feed name to each item's title. This pipe is useful when inserted into another pipe and you can hard code the feed URL and the label...
Published on 10/30/07 | 1568 clones

Title Mangler - improved ☆ by Christopher Peter
I just took out some mistakes and exchanged Prefix/Postfix to make it more handy. Lets you add text before and after the title in a feed. Use for blanks between Prefix or Postfix and URL.
Tags: feed feeds prefix editing improve +3...
Published on 05/29/09 | 214 clones

Update Maker for Lifestreams ☆ by Kingsley
If you would like to post lifestream updates like "Kingsley listened to 'Like You Used To' by J.J Cale on Last.fm", then this is the pipe for you! Give it a feed url. Then give it a prefix (like "Kingsley listened to ") - it will be added in front of the title of each...
Sources: twitter.com
Published on 04/28/08 | 270 clones

Figure 1. Natural Language Service Descriptions on Yahoo! Pipes

enables machines to understand the core aspects of Web services such as the specifications, the exposed API, and even QoS information. Nevertheless, due to the lack of semantic annotations, WSDL cannot elaborate the relationships between each of the related descriptive information of Web services. Moreover, WSDL normally is not capable for describing composite services as it can hardly describe the relationships and interactions between the components services of a composite service.

Similarly, WADL (W3C, 2009) is proposed as a description language for RESTful Web services since the dramatic development of REST related techniques. However, its drawbacks could also be observed in the absence of semantic annotations for services and the lack of abilities on describing composite services.

OWL-S. The Semantic Web is always being defined as the approach of bringing a machine-readable mechanism to Web resources including Web services. The DARPA Agent Markup Language (a.k.a. DAML) (Office, 2006) extends XML and Resource Description Framework (RDF) (W3C, 2004) to provide a set of constructs for creating machine-readable ontologies and markup information. The contribution of DAML program in Semantic Web is the Web Ontology Language for Services (OWL-S) (Coalition, 2003). OWL-S is a Web ontology which enables automatic service discovery, invocation, composition, interoperation, and execution monitoring (Ankolekar et al., 2002).

However, since OWL-S models services by using a three-party ontology (i.e. service profile, service model, service grounding), the data payload in transferring OWL-S descriptions at runtime could be way too large for end-users. Also, as claimed by industry players, it is still extremely difficult to embrace OWL-S in real world nowadays.

WIfL. As proposed by Danielsen and Jeffrey (2013), WIfL leverage the power of RDFa (W3C, 2013a) to introduce semantic annotations into hypertext-based descriptions of RESTful Web services. By adopting WIfL, the originally human-readable hypertext descriptions also become machine-readable.

However, since WIfL is developed for annotating RESTful services, it cannot describe SOAP-based services. Furthermore, as another description language for single raw service, WIfL has yet to be equipped with the ability to describe composite services.

3. THE ARCHITECTURAL FRAMEWORK OF HYPERMASH

HyperMash is a Web-based system that can be used through a Web browser. The HyperMash architectural framework consists of three layers (Figure 2): the Web-based User Interface Support layer, the Service Composition Engine layer and the Middleware System layer. These layers and their major system components are described in the following sections.

3.1 WEB-BASED USER INTERFACE SUPPORT

This layer consists of a GUI, a visual editor and a set of supporting tools. Through the GUI and visual editor, end-users can define a sequence of Web services on the fly by means of drawing graphical workflow diagrams. Figure 3 shows a workflow diagram the end-user can create by using the GUI and visual editor of HyperMash. A detailed illustration of using HyperMash for service composition is given in sections 6.

This layer provides the following tools for Web service composition: (1) a local database for keeping personal data and information of end-users; (2) a geo-location detector for retrieving the physical location of end-users at runtime; (3) a collection of multimedia widgets for rendering and displaying online videos or audios; and (4) a set of data processors for manipulating the runtime dataflow of the composite services.

3.2 Service Composition Engine

This layer consists of two systems called "*Service Recommender*" and "*Service Composer*". The function of *Service Recommender* is to automatically generate machine-readable composite service description documents and help end-users to retrieve the described composition knowledge on the fly. The *Service Composer* is designed to compose a set of Web services according the end-user defined workflow and provide runtime monitoring and substituting supports.

The *Service Recommender* consists of two components – *Semantic Composite Service Description Generator* (SCSDG) and *SPARQL Semantic Querying Engine* (SSQE). By adopting Semantic-UiSDL (see Section 4), SCSDG can automatically describe the composite services defined by end-users in a machine-readable manner. By doing so, the composition knowledge involved in the composite services can be therefore retrieved by SSQE to enable the reuse of the existing composition knowledge.

The *Service Composer* can compose RESTful and SOAP-based services separately as well as together. Meanwhile, inside of the *Service Composer*, the three connectors – *M4REST Connector*, *M4SOAP Connector* and *Availability Checker Connector* are respectively responsible for invoking and communicating with the corresponding components in the middleware system, described in Section 5, to correctly monitor, consume and manipulate the primitive Web services at runtime. When a primitive service is detected as unavailable by the *Service Monitor* at runtime, the *Service Substitutor* will help end-users to find alternative services to replace the failed one. Also, the *Service Composer* can differentiate different types of Web services according to Semantic-UiSDL, presented in Section 4.

3.3 Middleware System

This layer consists of the following three subsystems: *M4REST*, *M4SOAP* and *Availability Checker*. *M4REST* (Middleware for RESTful Web Services) supports the full set of the RESTful service interface functions.

M4REST consists of four separate components, respectively supporting the GET, POST, PUT, and DELETE functions exposed by RESTful Web services.

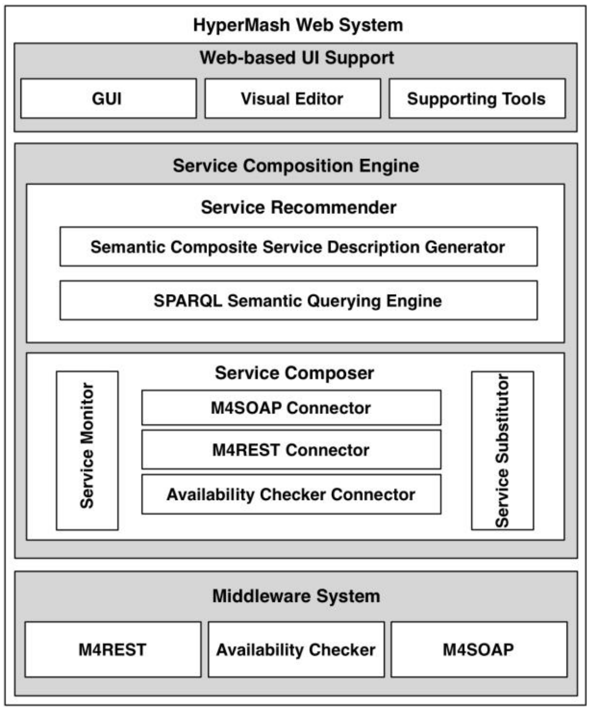


Figure 2. The Architecture of HyperMash

M4SOAP (Middleware for SOAP-based Web Services) aims at wrapping and unwrapping SOAP messages and invoking RPC-like procedures for invoking SOAP-based Web services at runtime.

The *Availability Checker* checks the availability status of the requested Web services and sends this information to the *Service Monitor*.

Since the *Service Composition Engine* is the most important layer in the entire HyperMash architecture, the underlying concepts, enabling technologies, and working principles of its inside systems (i.e. *Service Recommender* and *Service Composer*) are presented in detail in Section 4 and 5, respectively.

4. Service Recommendation based on Semantic-UiSDL

To better support end-users in the HyperMash approach, the *Service Recommender* is designed to help end-users to retrieve and reuse the composition knowledge of the existing composite services on the fly. By utilizing Semantic Web standards and technologies, we developed a Resource Description Framework (RDF) based service description language, called Semantic-UiSDL, to semantically describe the composite services defined by end-users. Since Semantic-UiSDL is adopted in SCSDG in HyperMash, all the composite services can be automatically and semantically annotated.

In this section, we present the requirements and vocabulary of Semantic-UiSDL, and illustrate how Semantic-UiSDL can be queried by a standard SPARQL semantic querying engine to provide service recommendations.

4.1 Language Requirements of Semantic-UiSDL

As stated in Section 2, by looking at example service composition platforms and the service description languages they use, we discovered some features of composite service descriptions in the existing end-user oriented service composition platforms.

Firstly, the authoring workflow of composite service descriptions is being done manually. For example, in Yahoo! Pipes, all the composite services have to be manually described by end-users in the form of natural language. The common artifact of this manual authoring workflow is human-readable hypertext page itself, which motivates our interest in making the composite service descriptions machine-readable.

Secondly, the details of the composite services do not necessarily align with the contents of the human-crafted descriptions. In a large number of extreme cases, according to our observation, the descriptions provided by the author (i.e. an end-user) of the composite services do not even mention any relevant information on the behaviors or purposes of the services.

Thirdly, it is quite common that with the changes made on the composite services, their descriptions became out-of-date. Some end-users always neglect the importance of keeping their composite service descriptions up-to-date.

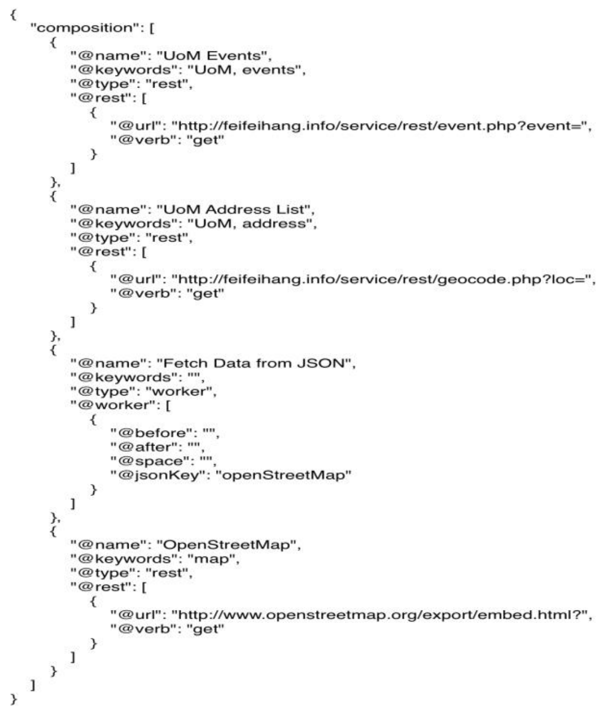
Based on these features, our requirement on Semantic-UiSDL is that it should automatically generate and update machine-readable descriptions of the composite services created in HyperMash.

4.2 Language Vocabulary of Semantic-UiSDL

Semantic-UiSDL extends UiSDL (Hang & Zhao, 2013) with semantic annotations. UiSDL is a Service Composition Ontology Description Language. It captures high-level service composition concepts used for describing the composite services, which may contain either RESTful or SOAP-based services or both. An example of the UiSDL service description is given in Figure 3.

However, due to the lack of semantic annotations, UiSDL is not machine-readable. This means the service composition system cannot understand the relationships between each of the description concepts exiting in the composite services. To overcome this drawback, we have enhanced UiSDL with semantic annotations based on standard Resource Description Framework (RDF). Semantic-UiSDL is the result of the enhancement.

Figure 4 illustrates the vocabulary of Semantic-UiSDL. As shown in the figure, we utilize the standard Dublin Core Metadata Element Set (a.k.a. dc) (Initiative, 2012) to describe the general information (e.g. title, creator, identifier) of each composite service, while the more detailed and concrete concepts are described by Semantic-UiSDL (SU) vocabulary.



```
{
  "composition": [
    {
      "@name": "UoM Events",
      "@keywords": "UoM, events",
      "@type": "rest",
      "@rest": [
        {
          "@url": "http://feifeihang.info/service/rest/event.php?event=",
          "@verb": "get"
        }
      ]
    },
    {
      "@name": "UoM Address List",
      "@keywords": "UoM, address",
      "@type": "rest",
      "@rest": [
        {
          "@url": "http://feifeihang.info/service/rest/geocode.php?loc=",
          "@verb": "get"
        }
      ]
    },
    {
      "@name": "Fetch Data from JSON",
      "@keywords": "",
      "@type": "worker",
      "@worker": [
        {
          "@before": "",
          "@after": "",
          "@space": "",
          "@jsonKey": "openStreetMap"
        }
      ]
    },
    {
      "@name": "OpenStreetMap",
      "@keywords": "map",
      "@type": "rest",
      "@rest": [
        {
          "@url": "http://www.openstreetmap.org/export/embed.html?",
          "@verb": "get"
        }
      ]
    }
  ]
}
```

Figure 3. An Example of UiSDL Description

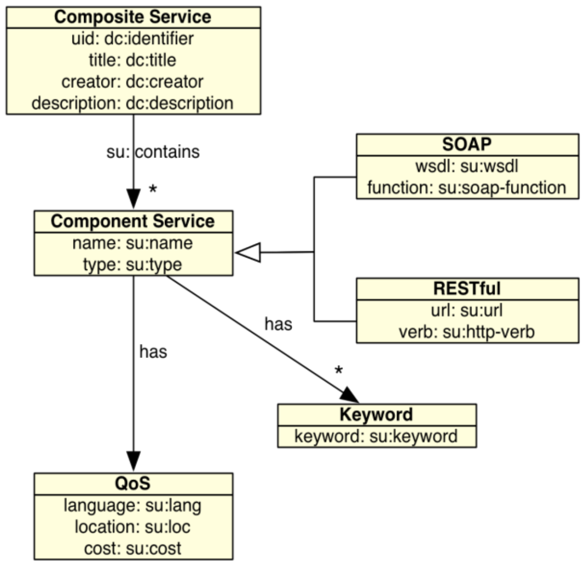


Figure 4. Semantic-UiSDL Vocabulary

As elaborated in Figure 4, each composite service in HyperMash consists of at least one component service that can be either RESTful or SOAP-based. Also, whatever

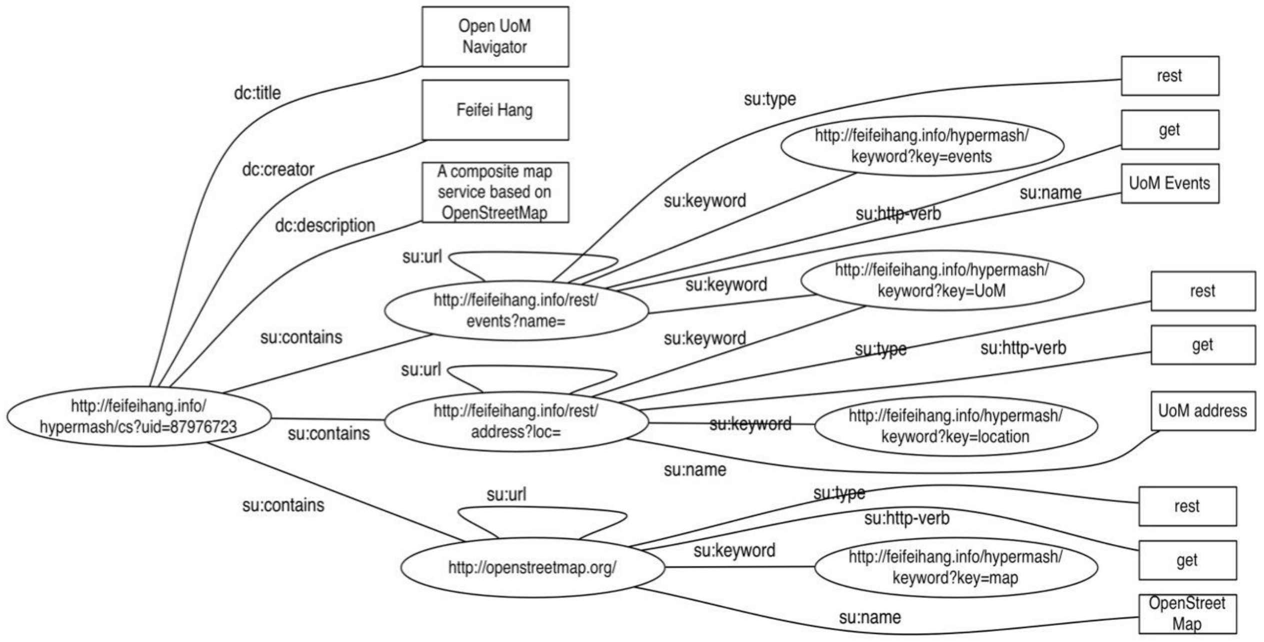

Open UoM Navigator
Feifei Hang
A composite map service based on OpenStreetMap
dc:title
dc:creator
dc:description
http://feifeihang.info/hypermash/cs?uid=87976723
su:contains
su:contains
su:contains
su:url
http://feifeihang.info/rest/events?name=
su:type
su:keyword
su:http-verb
su:name
http://feifeihang.info/hypermash/keyword?key=events
rest
get
UoM Events
su:keyword
http://feifeihang.info/hypermash/keyword?key=UoM
su:url
http://feifeihang.info/rest/address?loc=
su:keyword
su:type
su:http-verb
rest
get
su:keyword
http://feifeihang.info/hypermash/keyword?key=location
UoM address
su:name
su:url
http://openstreetmap.org/
su:type
su:http-verb
su:keyword
http://feifeihang.info/hypermash/keyword?key=map
su:name
rest
get
OpenStreet Map

Figure 5. The RDF Graph of a Semantic-UiSDL document

types the component service is, it always contains a set of descriptive keywords (a.k.a. tags) to briefly categorize the service and a group of user-level QoS information (Jingwen & Nahrstedt, 2004) to fulfill the context-awareness requirements of end-users.

Figure 5 shows the RDF graph of the example composite service described in Semantic-UiSDL. In this graph, each rectangle represents a concrete descriptive value of the composite service such as *su:title*. Each circle indicates the properties that is described as a resource such as *su:url* for RESTful services.

By describing certain descriptive properties as resources, it conceptually starts to build a web of composite services with linked resources (a.k.a. linked-data) among them. Figure 6 shows a visualized web of composite services and the relationships between their component services after adopting Semantic-UiSDL.

As can be seen from Figure 6, in the Semantic-UiSDL web the composite services are semantically connected together through the shared component services in between. For example, a component service "Google Maps" could be composed by a composite service "Earthquake Locator" and another composite service such as "Post Office Finder". Therefore, the detailed composition knowledge of the existing composite services can be discovered and reused by HyperMash.

4.3 Querying Semantic-UiSDL with SPARQL Engine

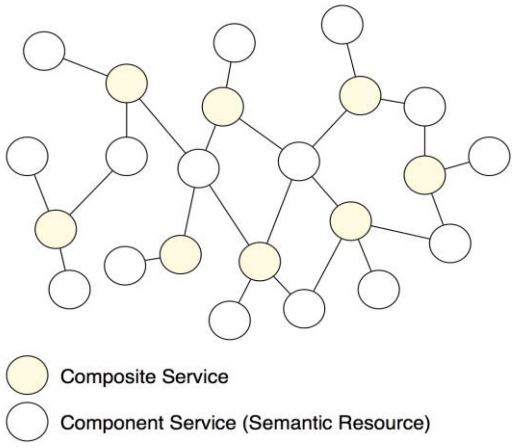


Figure 6. A Semantic-UiSDL Composite Services Web

As one of the most important features of semantic service description languages, Semantic-UiSDL enables both end-users and machines to understand the relationships (a.k.a. composition knowledge) between each one of the composite and primitive services.

Since Semantic-UiSDL is based on RDF, we use SPARQL (W3C, 2013b), a widely accepted language for querying RDF documents, to retrieve the composition knowledge from Smenatic-UiSDL documents in HyperMash.

Figure 7 shows a part of the Semantic-UiSDL document of the composite service example as illustrated in Figure 3.

To query the URL, for example, of all the component services involved in the composite services, we use the SPARQL code as shown in Figure 8 to fetch the URL resources from the document. The result of the querying is also shown in Figure 8.

With Semantic Web and SPARQL, we can perform more complex tasks on querying Semantic-UiSDL documents. Figure 9 elaborate how we can find out the URLs of all the component services that contain the keyword “UoM” in the example composite service.

5. The Process & Working Principles of Service Composer

The *Service Compoer* enables service composition in five processing steps. This section describes the process of *Service Composer* and the working principles underlying this process.

1) Defining a Workflow

The runtime working process of *Service Composer* starts from the creation of the workflow diagram made by end-users. To create a workflow diagram in HyperMash, end-users can easily define the sequence of the workflow of their ad-hoc composite service by means of drag-n-drop graphical representations of the primitive Web services and connecting the Web services by using pipelines in the visual editor as Figure 10 shows. Thereafter, the Service Composer will assemble the primitive services according to the end-user defined workflow, and inform SCSDG in the *Service Recommender* to generate Semantic-UiSDL documents to record the composition knowledge of this composite service.

2) Starting the Runtime Synchronization Mechanism

When the user-defined workflow is executed, the *Service Composer* begins to iterate through all the selected Web service and follow the HTML5-enabled runtime synchronization mechanism elaborated in Figure 11.

As one of the basic requirements of composing Web services, the procedure of accessing and consuming the composed Web services should be asynchronous. By default, JavaScript, which is widely used to create dynamic Web pages and Web applications, has already obtained the ability of raising HTTP request asynchronously with the help of XMLHttpRequest (W3C, 2013c). However, there is a significant shortcoming in accessing multiple Web services directly through XMLHttpRequest, as it causes the creation of a large number of threads at runtime. For example, if there are 20 Web services involved in a composite service, it will generate 20 threads when consuming these Web services.

```
...
<rdf:Description rdf:about="http://feifeihang.info/service/rest/event.php?event=">
   <su:name>UoM Events</su:name>
   <su:keyword rdf:resource="http://feifeihang.info/hypermash/projects/keyword.php?key=UoM"/>
   <su:keyword rdf:resource="http://feifeihang.info/hypermash/projects/keyword.php?key=events"/>
   <su:type>rest</su:type>
   <su:url rdf:resource="http://feifeihang.info/service/rest/event.php?event="/>
   <su:http-verb>get</su:http-verb>
</rdf:Description>

<rdf:Description rdf:about="http://feifeihang.info/service/rest/geocode.php?loc=">
   <su:name>UoM Address List</su:name>
   <su:keyword rdf:resource="http://feifeihang.info/hypermash/projects/keyword.php?key=UoM"/>
   <su:keyword rdf:resource="http://feifeihang.info/hypermash/projects/keyword.php?key=address"/>
   <su:type>rest</su:type>
   <su:url rdf:resource="http://feifeihang.info/service/rest/geocode.php?loc="/>
   <su:http-verb>get</su:http-verb>
</rdf:Description>

<rdf:Description rdf:about="http://www.openstreetmap.org/export/embed.html?">
   <su:name>OpenStreetMap</su:name>
   <su:keyword rdf:resource="http://feifeihang.info/hypermash/projects/keyword.php?key=map"/>
   <su:type>rest</su:type>
   <su:url rdf:resource="http://www.openstreetmap.org/export/embed.html?"/>
   <su:http-verb>get</su:http-verb>
</rdf:Description>
...
```

Figure 7. Example Semantic-UiSDL Document

```
PREFIX su: <http://feifeihang.info/hypermash/semantic-usdl#>
SELECT ?url
WHERE {
   ?o su:url ?url.
}
```

SPARQLer Query Results

url
<http://www.openstreetmap.org/export/embed.html?>
<http://feifeihang.info/service/rest/geocode.php?loc=>
<http://feifeihang.info/service/rest/event.php?event=>

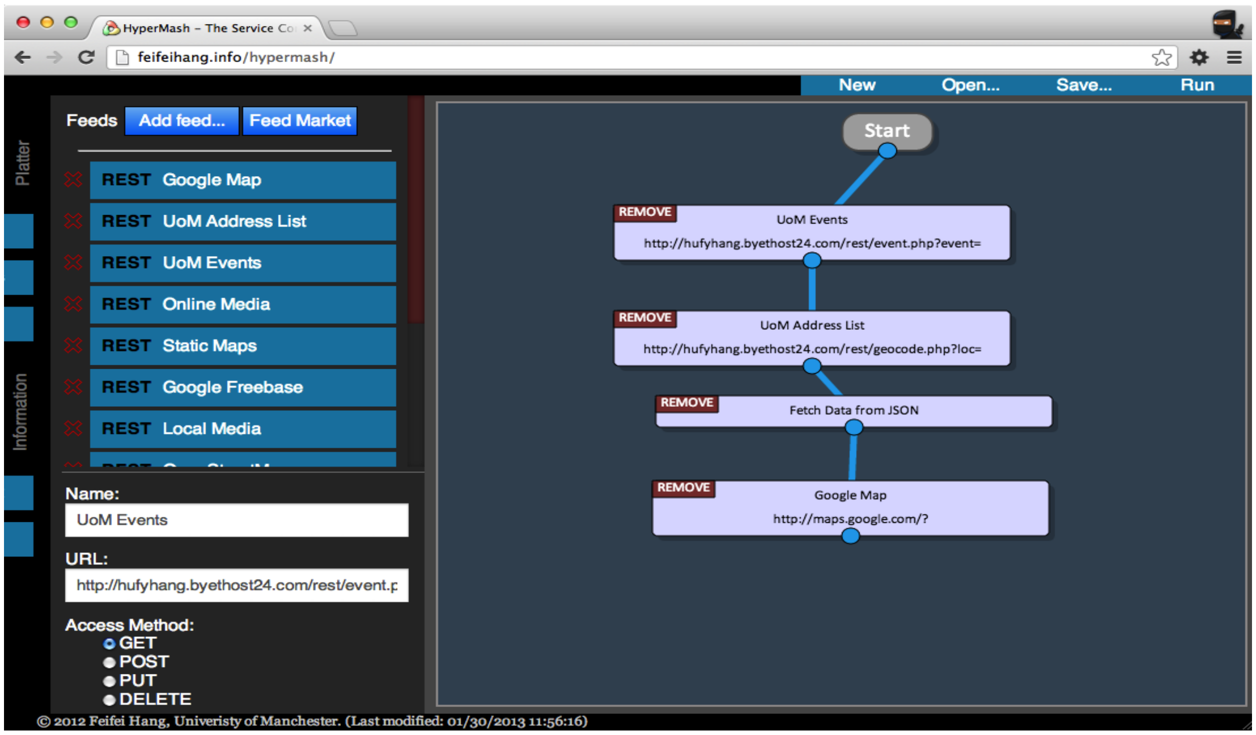
Figure 8. A Simple Semantic-UiSDL Querying Example

```
PREFIX su: <http://feifeihang.info/hypermash/semantic-usdl#>
SELECT ?name ?url
WHERE {
   ?o su:name ?name;
   su:url ?url;
   su:keyword <http://feifeihang.info/hypermash/projects/keyword.php?key=UoM>;
}
```

SPARQLer Query Results

name	url
"UoM Address List"	<http://feifeihang.info/service/rest/geocode.php?loc=>
"UoM Events"	<http://feifeihang.info/service/rest/event.php?event=>

Figure 9. A More Complex Semantic-UiSDL Querying Example



Name: UoM Events

URL: http://hufyhang.byethost24.com/rest/event.p

Access Method: [✓ GET] POST PUT DELETE

© 2012 Feifei Hang, Univeristy of Manchester. (Last modified: 01/30/2013 11:56:16)

Figure 10. Creating a Workflow Diagram in HyperMash

By leveraging the power of HTML5 technologies, we utilize HTML5 Web Worker (W3C, 2012) as the means of realizing asynchronous procedures while maintaining a reasonable number of threads, i.e., only three threads – one for the main thread, one for checking availability, and another one for consuming and manipulating Web services, at runtime.

3) *Monitoring the Availability of the Requested Web Service*

Shown in Figure 11, once the whole synchronization mechanism is started, the service composition engine initializes the HTML5 Web Worker containing *Availability Checker Connector* by posting a certain message through standard APIs to create the running thread for monitoring Web services.

The returned availability information is transferred to the *Service Monitor* for the availability analysis. If the *Service Monitor* finds the target component Web service is unavailable, it will inform the *Service Substitutor* to provide end-users a list of suggestions for replacing the unavailable Web service and highlight its corresponding part in the workflow diagram in the visual editor. Afterwards, the workflow will be terminated.

4) *Consuming the Component Web Service*

If the requested Web service is available at runtime, the other Web Worker will be invoked to initialize the thread for consuming Web services.

As shown in Figure 12, depending on the type of the requested component service, either M4REST Connector or M4SOAP Connector will communicate with the corresponding middleware system and component to consume the Web service.

5) *Finalizing the Runtime Synchronization Mechanism*

Once the returned information is received from the requested component service, the information will be buffered in the *Service Composer*. If there are no more component services involved in the workflow, the *Service Composer* will display the buffered information in the GUI to show the compositional result to end-users. Otherwise, the *Service Composer* will move on to the next primitive service involved and repeat the above steps.

6. ILLUSTRATING THE HYPERMASH APPROACH

To illustrate the HyperMash approach, we have developed a prototype system. The system was developed by using HTML5, JavaScript, RDF, and PHP, and is hosted on a standard Apache server[1] to prove its viability on standard Web configurations.

The way HyperMash is used depends on the purpose of the end-users. Below is a typical use example of HyperMash.

Tom is an office worker who wishes to integrate his daily work plan created by a SOAP-based Web service with a RESTful on-line map service to help him to know where

[1] http://feifeihang.info/hypermash

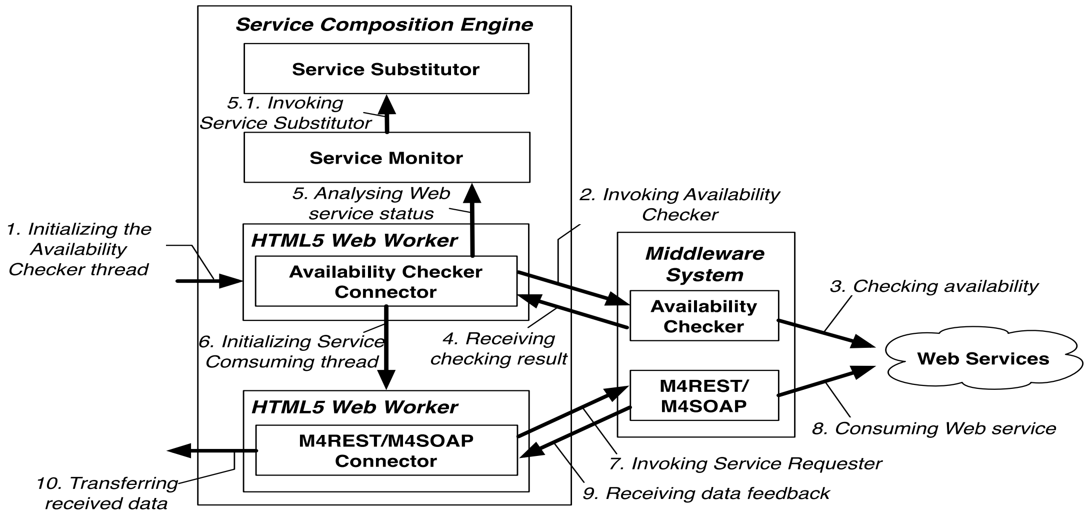

Service Composition Engine
Service Substitutor
5.1. Invoking Service Substitutor
Service Monitor
5. Analysing Web service status
HTML5 Web Worker
Availability Checker Connector
1. Initializing the Availability Checker thread
2. Invoking Availability Checker
Middleware System
Availability Checker
3. Checking availability
Web Services
6. Initializing Service Comsuming thread
4. Receiving checking result
HTML5 Web Worker
M4REST/M4SOAP Connector
M4REST/ M4SOAP
8. Consuming Web service
7. Invoking Service Requester
9. Receiving data feedback
10. Transferring received data

Figure 11. Runtime Synchronization Mechanism

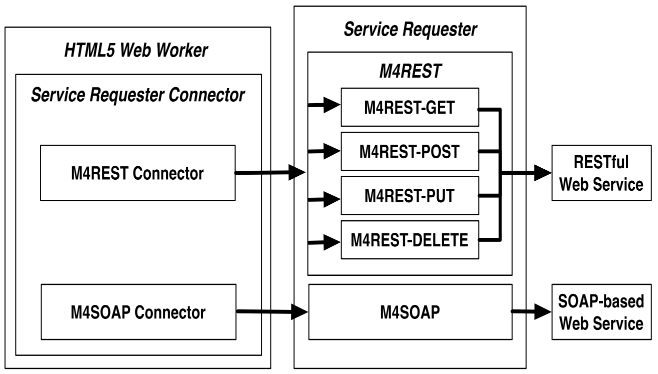


Figure 12. Middleware System Connectors

to go for his remaining works. He can perform this integration through the following steps:

Step 1: Importing both the RESTful and SOAP-based Web services into the service composition platform by providing the URL and WSDL addresses.

Step 2: Receiving semantic suggestions from the *Service Recommender* to facilitate service composition.

Step 3: Picking the desired Web services by clicking the corresponding graphical labels.

Step 4: Creating the desired workflow by simply connecting the graphical nodes in the visual editor.

Step 5: Selecting the desired function that he wants to use to retrieve his work plan from the SOAP-based service.

Step 6: Clicking “Save…” in the main GUI to save the ad-hoc composite service.

Step 7: Clicking “Run” in the main GUI to execute the ad-hoc composite service.

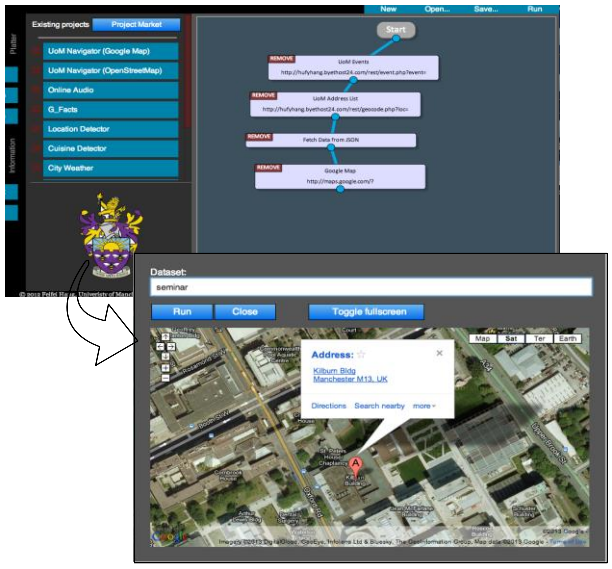


Figure 13. UoM Navigator

Additionally, since the HyperMash approach is based on standard Web technologies, he can now even use his ad-hoc composite service on his mobile devices.

Figure 13 shows the ad-hoc composite service created by Tom and its runtime execution result.

7. Evaluating the Accuracy of Service Recommendation in HyperMash

By adopting Semantic-UiSDL, HyperMash can better support end-users to compose services by reusing the existing service composition knowledge. In this section, we present our work on evaluating the accuracy of service recommendation in HyperMash.

Table 2. The Accuracy Comparison on Manually Service Retrieving and Semantic-UiSDL-Based Service Recommendation

Task ID	Activities	Sample Size	Manually Service Retrieving		With Semantic-UiSDL-Based Service Recommendation	
			# of hits	# of correct/relevant hits	# of hits	# of correct/relevant hits
1	Navigation	28	23	17	28	27
2	On-line Shopping	22	15	13	20	18
3	Langauge Translation	19	12	7	19	18
4	Weather Forecast	13	8	5	12	12
5	On-line Radio	8	5	5	7	6

Table 3. Precision and Recall of Two Approaches

Task ID	Precision		Recall	
	Manually Service Retrieving	With Service Recommendation Based on Semantic-UiSDL	Manually Service Retrieving	With Service Recommendation Based on Semantic-UiSDL
1	0.74	0.96	0.61	0.96
2	0.87	0.90	0.59	0.82
3	0.58	0.95	0.37	0.95
4	0.63	1.00	0.38	0.92
5	1.00	0.86	0.63	0.75

7.1 Evaluation Method

According to Chen et al. (Chen et al., 2011), Precision and Recall are two widely adopted metrics in the Information Retrieval domain. Thus, we use the below formulas to evaluate our work.

$$Precision = \frac{succ(c)}{succ(h)}, Recall = \frac{succ(c)}{sum(s)}$$

In the above formulas, *succ(c)* is the number of the relevant services retrieved, *succ(h)* is the total number of services retrieve, and *sum(s)* is the sample size.

7.2 Data Collection

We recruit a group of Master's students who have no professional knowledge of SOA to be our test users. We gave the students a 15-minutes tutorial on how to use HyperMash and asked them to create composite services to assist five different daily activities by using Semantic-UiSDL service recommendation and traditional and manual service retrieving, respectively. Each activity consists of a number of processes that needed to be achieved by primitive services. For example listed in Table 2, Task 1 is the activity "navigation", which contains the primitive services for detecting current location, locating destination, showing route map on a map service, etc.. To better evaluate our system, all the primitive services that needed to be composed in the new composite services are already be used in a set of pre-created composite services in HyperMash to fully generate the RDF graphs for providing semantic

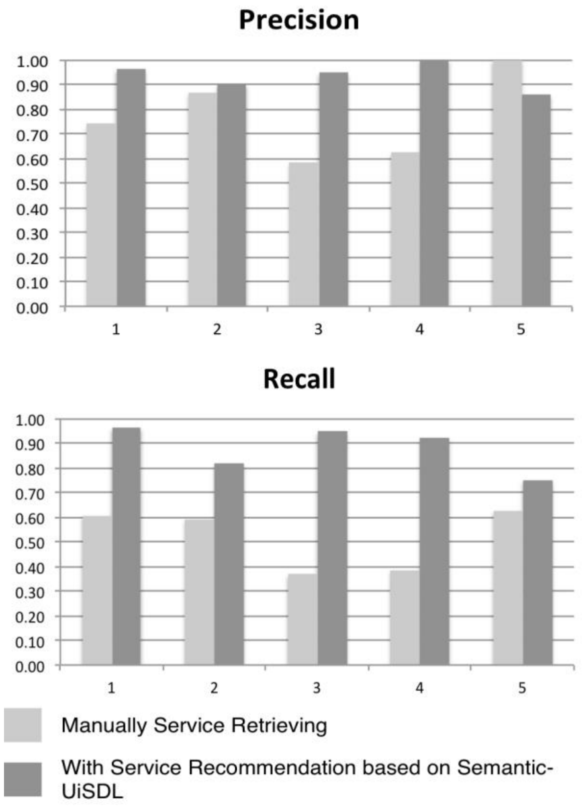


Figure 14. Performance Comparison of Two Service Retrieving and Recommendation Approaches

suggestions. The archived result of our experiment on average is listed in Table 2.

7.3 Results Analysis

Table 3 shows the calculation results of Precision and Recall against the data we collected. These results are also visualized in the form of bar chars in Figure 14.

As shown in Figure 14, it is clear that the service recommendation based on Semantic-UiSDL shows a higher accuracy than the manually service retrieving approach, which is commonly adopted in the existing end-user service composition systems such as Yahoo! Pipes. The exceptional case in Precision on Task 5 is due to the fact that the Semantic-UiSDL-based service recommendation retrieved more services (7 on average) from the repository, but contains an irrelevant candidate.

8. Conclusion

In this paper, we have identified four major limitations in current service composition approaches and service description languages, and proposed the HyperMash approach to address these limitations. Specifically, the first limitation has been addressed by providing a Web-based, end-user friendly composition platform. To overcome the second limitation, HyperMash enables the composition and combination of both RESTful and SOAP-based services. HyperMash provides the full set of RESTful interface features support to overcome the third limitation. Finally, Semantic-UiSDL enables automatically descriptions of user-defined composite services, and allows the sharing and reusing of the existing service composition knowledge. In so doing, we claim that this paper has made an important contribution to the area of end-user service composition.

In the paper, we have described HyperMash approach in detail and shown, through realistic examples and empirical study, that they have the potential to be used in practice.

Our work, however, still suffers from the following shortcomings:

- Although our prototype system can already help end-users easily compose and manipulate Web services, it still provides limited support for automatically mediating the transferred data at runtime.
- Semantic-UiSDL does not automatically include the user-level QoS information of primitive services and this information needs to be manually added to the description.

Our future work will address these shortcomings. In addition, we will investigate the best practice and design for supporting HATEOAS for RESTful Web services and enhance HyperMash with the appropriate security mechanism. We intend to evaluate our work more thoroughly, through user studies and real world experiments.

9. References

Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McDermott, D., Mcllraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K. (2002). DAML-S: Web Service Description for the Semantic Web. In *The Semantic Web — ISWC 2002* (Vol. 2342, pp. 348-363): Springer Berlin Heidelberg.

Bizer, C. (2009). The Emerging Web of Linked Data. Intelligent Systems, IEEE, 24(5), 87-92. doi: 10.1109/MIS.2009.102

Cappiello, C., Matera, M., Picozzi, M., Sprega, G., Barbagallo, D., Francalanci, C. (2011). DashMash: A Mashup Environment for End User Development. In *Web Engineering* (Vol. 6757, pp. 152-166): Springer Berlin Heidelberg.

Chen, L., Hu, L., Zheng, Z., Wu, J., Yin, J., Li, Y., Deng, S. (2011). WTCluster: Utilizing Tags for Web Services Clustering. In *Service-Oriented Computing* (Vol. 7084, pp. 204-218): Springer Berlin Heidelberg.

Coalition, The OWL Services. (2003). OWL-S: Semantic Markup for Web Services. Retrieved 27 December 2013, from: http://www.daml.org/services/owl-s/1.0/owl-s.html.

Danielsen, P. J., Jeffrey, A. (2013). Validation and Interactivity of Web API Documentation. Paper presented at the Proceedings of *2013 IEEE 20th International Conference on Web Services*, Santa Clara, USA.

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine.

Hang, F., Zhao, L. (2013). HyperMash: A Heterogeneous Service Composition Approach for Better Support of the End Users. In *Web Services (ICWS), 2013 IEEE 20th International Conference on* (pp. 435-442). IEEE.

Hoang, D. D., Paik, H. Y., Benatallah, B. (2010). An analysis of spreadsheet-based services mashup. In *Proceedings of the Twenty-First Australasian Conference on Database Technologies-Volume 104* (pp. 141-150). Australian Computer Society, Inc..

Initiative, Dublin Core Metadata. (2012). Dublin Core Metadata Element Set, Version 1.1. Retrieved 11 October 2013, from http://dublincore.org/documents/dces/

Jin, J., Nahrstedt, K. (2004). QoS specification languages for distributed multimedia applications: A survey and taxonomy. *Multimedia, IEEE, 11*(3), 74-87.

Krummenacher, R., Norton, B., Simperl, E., Pedrinaci, C. (2009). Soa4all: enabling web-scale service economies. In *Semantic Computing, 2009. ICSC'09. IEEE International Conference on* (pp. 535-542). IEEE.

Lécué, F., Gorronogoitia, Y., Gonzalez, R., Radzimski, M., Villa, M. (2010). SOA4All: an innovative integrated approach to services composition. In *Web Services (ICWS), 2010 IEEE International Conference on* (pp. 58-67). IEEE.

Liu, X., Hui, Y., Sun, W., Liang, H. (2007). Towards service composition based on mashup. In *Services, 2007 IEEE Congress on* (pp. 332-339). IEEE.

NetMarketShare (2013). Browser Market - Market Share for Browsers, Operating Systems and Search Engines. Retrieved 22 January 2010, from http://marketshare.hitslink.com/report.aspx?qprid=0

Obrenovic, Z., Gasevic, D. (2008). End-user service computing: spreadsheets as a service composition tool. *Services Computing, IEEE Transactions on*, *1*(4), 229-242.

DARPA's Information Exploitation Office (2006). The DARPA Agent Markup Language. Retrieved 18 January 2013, from http://www.daml.org/

Pautasso, C. (2009). Composing restful services with jopera. In *Software Composition* (pp. 142-159). Springer Berlin Heidelberg.

Richardson, L., Ruby, S. (2008). *RESTful web services*. O'Reilly.

Simmen, D. E., Altinel, M., Markl, V., Padmanabhan, S., Singh, A. (2008). Damia: data mashups for intranet applications. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 1171-1182). ACM.

Torma, S., Villstedt, J., Lehtinen, V., Oliver, I., Luukkala, V. (March 2008). Semantic web services–a survey. Retrieved 27 December 2013, from http://www.cs.hut.fi/u/sto/B158.pdf

W3C (2001). Web Services Description Language (WSDL). Retrieved 18 January 2013, from http://www.w3.org/TR/wsdl

W3C (2004). Resource Description Framework (RDF). Retrieved 18 January 2013, from http://www.w3.org/RDF/

W3C (2009). Web Application Description Language (WADL). Retrieved 18 January 2013, from http://www.w3.org/Submission/wadl/

W3C (2012). Web Worker. Retrieved 18 January 2013, from http://www.w3.org/TR/workers/

W3C (2013a). RDFa Core 1.1 - Second Edition. Retrieved 10 October 2013, from http://www.w3.org/TR/rdfa-syntax/

W3C (2013b). SPARQL 1.1 Overview. Retrieved 11 October 2013, from http://www.w3.org/TR/sparql11-overview/

W3C (2013c). XMLHttpRequest Level 2. Retrieved 18 January 2013, from http://www.w3.org/TR/XMLHttpRequest/

Wikipedia (2013). Mashup (web application hybrid). Retrieved 24 January 2013, from http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)

Wong, J., Hong, J. I. (2007). Making mashups with marmite: towards end-user programming for the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 1435-1444). ACM.

Yahoo! (2013). Yahoo! Pipes. Retrieved 18 January 2013, from http://pipes.yahoo.com/pipes

Authors

Feifei Hang is a PhD candidate in the School of Computer Science at the University of Manchester. His main research area is in investigating the best practice for and developing supporting approaches to support service composition for end-users in the context of Web 2.0. His research interests also include: services computing, cloud computing, end-user development, and user experience in emerging Web technologies.

Liping Zhao is an academic member of School of Computer Science at the University of Manchester. Her research focuses on the discovery and development of reusable software patterns, symmetries and models, and the application of these fundamentals to model-driven development and service-oriented computing. A pioneer in service sciences, she co-funded and led the first academic network on service sciences in the UK. Her contribution to software patterns and the service sciences agenda has earned her three IBM Faculty Awards.