Mashup Creation Using a Mashup Rule Language

HYOSOOK JUNG AND SEONGBIN PARK*

Department of Computer Science Education
Korea University
Seoul, 136-701 Korea

A mashup is an application that combines functionalities of other applications to create a new service. In this paper, we propose an ontology-based system by which users without programming experiences can easily construct mashups of Web resources. The system allows users to define their own search parameters using a rule language and expands users’ queries using the concepts in ontologies. Then it generates a web page specific to that request. We show how the proposed system can be used to create personalized mashups in the context of adaptive hypermedia.

Keywords: web 2.0, semantic web, mashup, ontology, adaptive hypermedia

1. INTRODUCTION

Web 2.0 represents a paradigm shift in how people use the Web. While most users were limited to passively viewing Web sites created by providers, they can now actively contribute to creating and updating contents as well as accessing them from web sites [16]. It enables the creation of new applications by resurging and combining different applications on the Web or by combining data and information from different sources [18]. Web 2.0 attractiveness of participation is an important factor for the success of an online social network sites such as facebook, YouTube, flickr, and Google. It has created online collaboration, user participation, social mechanism and API (Application Programming Interface) mashups. The capability to plug-in services such as API mashups has improved the participation attractiveness of the hosting sites [20]. Mashup is generated using open technologies such as AJAX, REST, RSS or ATOM [28].

In this paper, we propose an ontology-based system by which users can create mashups using a mashup rule language. Our system aggregates Web resources using Google AJAX Search API [10]. Since it is difficult for users without programming experiences to use the API, we define a mashup rule language that enables average users to easily search resources through Google services and integrate search results. Fig. 1 shows how our system works when a user sends a query. In order to create a mashup, a user defines a mashup rule that specifies which services the system uses to search resources such as web page, blog, image, video, news, book, and location (specific service), where the search results are embedded on a web page for display such as center, left, right, bottom, or top (layout of the search results), which site the search is restricted to (site restriction), and
which terms the system uses for the search (optional terms). The mashup rule compiler implemented using SableCC [9] compiles the mashup rule. Then, the user defines an ontology that contains concepts and their relationships about a domain, or search relevant ontologies from Swoogle [7]. The ontology restricts the scope of the search. When the user types a query, the mashup server extracts the terms related to the query from the ontology using Protégé API [8] and expands the query by combining the original query and the extracted terms using OR operator. In addition, if the user has defined optional terms in the mashup rule, the mashup server adds them to the expanded query using OR operator. Then the system starts searching resources by using the extended query and embeds the search results on a web page according to the conditions imposed in the mashup rule. An adaptive mashup can be also generated by utilizing the information contained in the user model in an adaptive hypermedia system [6].

Fig. 2 shows the screenshot of the system. A user enters a query in a text box (A). The user can see a window where the user can define the mashup rule by clicking a but-
ton (B). The user can see a window where the user can search ontologies from Swoogle by clicking a button (C). The user can see a mashup result (D).

This paper is organized as follows. Section 2 describes related works. Section 3 describes the functionalities of the proposed system using examples. Section 4 explains how adaptive mashups can be constructed. Section 5 concludes the paper and describes future works.

2. RELATED WORKS

There are two related areas to our work. One is about various approaches to create mashups and the other is adaptive hypermedia.

There are approaches that allow users without coding experiences to create mashups. Marmite provides graphical objects that represent code and allows users to build mashups by constructing and manipulating them [24]. Yahoo Pipe or Microsoft Popfly let users create mashups easily through the use of designed modules [14, 17, 27]. Marmite, Popfly and Pipe are simple and easy to use because they use modules designed by developers and provide a drag and drop interface. Popfly and Pipe build mashup by integrating contents extracted according to user’s input in parameters of their modules. Active Tags [12] is a Firefox extension that lets users use tags on any web page. It is easy for average Web users to create mashups based on tags they find on the web pages by clicking on a part of the page. Potluck is a tool to mash up data by merging fields of RDF without programming skills and data modeling expertise. Potluck loads some datasets and presents all fields from the different sources as field tags. Users drag proper field tags to a column to mix up different fields from the datasets [15]. MashMaker [5] is a browser extension that enables users to make personalized mashup with content and visualizations from other sources on the web and brings it all together in one place. To create a mashup, widgets should be created by other users in advance because a mashup is the combination of multiple widgets. However, creating a widget requires the knowledge of XML and Javascript. Our system is also for users without programming experiences. The difference between our system and other systems is that adaptive mashups for different types of users can be easily constructed. This is helpful especially in a Web-based education where students with different backgrounds have to study learning materials provided by teachers.

There is also a mashup system that can be used for advanced users with programming experiences. For example, users who have the knowledge of XML tags, Javascript, CSS and HTML can use Google mashup editor to create mashup quickly and easily [11, 19].

Some systems such as SA-REST [21] and ONKI [22] utilize ontologies. SA-REST uses ontologies to embed semantic annotations to RESTful services. SA-REST embeds semantic annotations in RDFa into the HTML pages or annotates them with GRDDL in order to add semantics to RESTful services. Its proxy server hosts a smashup (semantic mashups) editor, ontology, and mediation rules. The ontology captures the semantics and supports semantic reasoning and data mediation. Users send URLs of the annotated HTML pages to the proxy server. The proxy server applies an XSLT to the annotated HTML pages, extracts the RDF triples and products the service descriptions. The users create the smashup by using these descriptions [21]. ONKI is a service for describing
contents by using explicitly identified concepts or terms, publishing the content on the semantic web, and using the content as services in applications. ONKI services can be integrated with mashup applications using AJAX and web widgets or by using Web Service interfaces. ONKI uses ontologies to find the search concepts semantically matching user’s query and the annotation concepts of the content so that it can remove the ambiguities of the queries or content [22]. Like SA-REST and ONKI, our system uses ontologies as well. The difference between our system and other systems is that we use ontologies to find semantic concepts related to user’s query and expand the query.

There are also systems by which users can create mashups which work with RDF data. Tabulator [4] is a generic browser for linked data on the Web. It uses an outliner mode which allows users to explore an RDF graph in a tree structure and to expand nodes in order to get more information about them. As exploring the outliner view, users can create a table, Google map, calendar and timeline or other view that contains pieces of information about each equivalently-related object that is an RDF node. Semantic Web Pipes [29] is an engine that transforms and mashups Semantic Web Data in a graphical environment. It implements a workflow defined in advance that processes a set of RDF sources by using specialized operators. Each operator enables both ordered inputs and unordered inputs in different formats, and only one output. A pipe is a set of operators and the overall output of the pipe are linked inputs of other operators. Its implementation consists of an execution engine and an AJAX based pipe editor. By using the pipe engine, users can make pipes to fetch, mix, and process RDF files published on the Web. The output of a pipe is an HTTP-retrievable RDF model or XML file.

Adaptive hypermedia applications consist of a user model, a domain model and an adaptation model. The user model represents a user’s preferences, knowledge, goals, browsing history and other relevant features. The domain model describes how to structure the application’s content. The adaptation model is defined as a set of adaptation rules to link the domain and user model to generate the presentation of content. The adaptation rules define how to update the user model and generate an adapted presentation [6, 25, 26]. AHA! [6] creates adaptation and updates the user model according to adaptation rules defined in an adaptation model. It also defines a domain model which describes the conceptual content of the application by using concepts and concept relationships.

3. FUNCTIONALITY OF THE SYSTEM

Internet users spend much time and efforts finding resources suited to their needs or interests because there are a lot of resources on the Internet. If they control the process of the search according to their needs, they can get good resources easily. In addition, if they integrate the related resources and arrange them into a web page according to their tastes, they can browse the resources easily and understand them at a glance. If they restrict the scope of the search to a domain which they are interested in, they can exclude unnecessary resources which are not related to the domain of interests.

To this end, our system allows users to integrate resources from different sources on the Web by defining mashup rules about how to create personalized mashup. In the mashup rule, they can specify which resources the system needs to search, where the search results are embedded on a web page for display, whether the search is restricted to a spe-
cial web site or not, or whether the system uses a special query term or not. In addition, they can use an ontology so that vocabularies in the ontology can be used.

Fig. 3 shows how our system can be viewed as a function. The argument of the function is a specific input in the domain $X$ which consists of mashup rule, ontology, and query. For the argument $x$, the corresponding $y$ is an output of the function in the codomain $Y$ that is a mashup result. For example, the mashup rule and the query of $x_1$ and $x_2$ are the same, but their ontologies are different; the ontology of $x_1$ is related to a place, but the ontology of $x_2$ is related to object-oriented programming. So, their mashup results are different; $y_1$ is the mashup result for $x_1$ and $y_2$ is the mashup result for $x_2$. In addition, the ontology and the query of $x_2$ and $x_3$ are the same, but their mashup rules are different; the mashup rule of $x_2$ requests the merge of web pages, blogs, and images, but the mashup rule of $x_3$ requests the merge of web pages in a certain site like java.sun.com and images. So, their mashup results are different; $y_2$ is the mashup result for $x_2$ and $y_3$ is the mashup result for $x_3$.

Assume that a user wants to create a mashup about extinct animals using the contents from the sites "www.sciencedaily.com" and "discovermagazine.com". The user first defines a mashup rule as follows,

```
begin
  web.align = left
  web.site = "www.sciencedaily.com"
  blog.align = bottom
  blog.site = "discovermagazine.com"
  image.align = right
  video.align = top
  local.align = false
  news.align = false
  book.align = false
end
```
Then the user defines an ontology that is related to the topics of the contents for a mashup in OWL [13] as follows,

```xml
<?xml version = "1.0"?>
<rdf:RDF xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl = "http://www.w3.org/2002/07/owl#"
    xmlns = "http://www.owl-ontologies.com/biology.owl#"
    xml:base = "http://www.owl-ontologies.com/biology.owl"> 
    <owl:Ontology rdf:about = ""/> 
    <owl:Class rdf:ID = "animal"/> 
    <owl:Class rdf:ID = "fish" rdf:resource = "#animal"/> 
    <owl:Class rdf:ID = "bird" rdf:resource = "#animal"/> 
    <owl:Class rdf:ID = "mammal" rdf:resource = "#animal"/> 
</rdf:RDF>
```

From this ontology, the system extracts four words, “animal”, “fish”, “bird”, and “mammal”. If the user sends a query, “extinction” to the system, it expands the query using OR operation and the expanded query is “extinction or animal or fish or bird or mammal”. Fig. 4 shows the result generated from the system.

Our system can create a mashup using contents from Wikipedia [23] that is a free encyclopedia project by using wiki that creates collaborative websites. Wikipedia has become one of the most popular online resources. Its articles are written by volunteers collaboratively. Its articles introduce the reader to key articles, images, and categories that further describe the subject.

![Fig. 4. A screenshot of the mashup.](image-url)
The user can obtain the article and a list of articles which include terms related to the query ("Europe" in this example). Left figure in Fig. 5 shows the mashup created with the query, "global warming". Our system can create mashups using contents from DBpedia [2] as well. DBpedia is a data corpus extracted from Wikipedia by converting Wikipedia content into structured knowledge. The DBpedia datasets can be imported into third party applications, and accessed online by using different DBpedia user interfaces. They can be accessed by three types of access mechanisms such as Linked Data, the SPARQL protocol and downloadable RDF dumps. DBpedia allows users to ask sophisticated queries against Wikipedia and it is an example of linked data [3] which uses URIs to identify things and links between arbitrary things described by RDF.

```plaintext
begin
  web.align = left
  web.label = "Wikipedia"
  web.site = "en.wikipedia.org"
  web.expand = open
  web.query = "Europe"
  video.align = false
  image.align = false
  book.align = false
  blog.align = false
  local.align = false
  news.align = false
end
```

```plaintext
begin
  web.align = left
  web.label = "DBpedia"
  web.site = "dbpedia.org/resource"
```
4. APPLICATIONS

In this section, we show how the proposed system can be used in an educational environment. In an educational system, a teacher might want to provide different mashups according to student’s knowledge or interests. For example, the teacher can be an author of a course in AHA! and his students can be the users who take the course. The teacher wants to show the students an adaptive mashup associated to the course. So, the teacher can provide different mashups according to the course knowledge of the students by defining the mashup rule with their knowledge values stored in the user model of AHA!. The teacher can also choose an ontology to restrict the scope of the search.

4.1 Scenario 1

Assume that a teacher wants to integrate data from different sources for students who take a course about global warming using AHA! system. The teacher thinks that if student’s knowledge about the overview page is more than zero, the student can access the page about effects. This is represented in the domain model as follows,

```xml
<concept>
  <name> globalwarming.effects.knowledge </name>
  <description> about global warming </description>
  <expr> ((globalwarming.overview.knowledge > 0)) </expr>
</concept>
```

The teacher defines a mashup rule that reflects user’s features. For example, the rule below specifies that if a student takes the course and student’s knowledge about the effects page is more than ‘80’, the mashup system creates a mashup page. It also increases user’s knowledge about the effects page which is stored in the user model by ten percent of the current knowledge.

```python
begin
  personal.course = "globalwarming"
  globalwarming.effects.knowledge ≥ 80
  update.globalwarming.effects.knowledge = globalwarming.effects.knowledge + (0.1 * _globalwarming.effects.knowledge)
end
```
At this stage, the teacher can either define an ontology or find an ontology using Swoogle. We assume that the teacher uses an ontology retrieved from Swoogle with a query, "global warming".

<owl:Class rdf:about="#GlobalWarming">
  <rdfs:subClassOf rdf:resource="#ClimateChange"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom rdf:resource="#GreenhouseEffect"/>
      <owl:onProperty rdf:resource="#hasAssociatedPhenomena"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

Assume that student’s knowledge about the effect page is ‘85’ which is defined in the user model is as follows,

<record>
  <key> personal.course </key>
  <type>2</type>
  <persistent>true</persistent>
  <value>globalwarming</value>
</record>
<record>
  <key>globalwarming.effects.knowledge</key>
  <type>1</type>
  <persistent>true</persistent>
  <value>85</value>
</record>

If the student sends a query "global warming" to the mashup system, the system first searches a special class which matches the query in the ontology. Then, it looks for classes that are related to the special class by the property type <rdfs:subClassOf>. Finally, it adds the classes as terms to the query like “global warming or ClimateChange or GreenhouseEffect”. Fig. 6 shows the result mashup page.
4.2 Scenario 2

Assume that a teacher would like to provide general contents for students who have little knowledge about a subject and specific contents for students who are familiar with the subject. For this example, we assume that two students will see the mashup pages. The user model of the student with little background knowledge about the subject (Student 1) is as follows,

<record>
  <key>personal.course</key>
  <type>2</type>
  <persistent>true</persistent>
  <value>culture</value>
</record>

<record>
  <key>culture.Australia.knowledge</key>
  <type>1</type>
  <persistent>true</persistent>
  <value>25</value>
</record>

There is a different student whose knowledge about the Australia page is ‘85’. (Student 2) The teacher specifies that if student’s knowledge about the readme page is more than zero, the student can access the Australia page of the course. This is represented in the domain model as follows,

<concept>
  <name>culture.Australia.knowledge</name>
  <description>about Australia</description>
  <expr>((culture.readme.knowledge>0))</expr>
</concept>
Then the teacher defines two different mashup rules as follows.

First, if a student takes the culture course and the student’s knowledge about the Australia page is less than ‘30’, the mashup system creates a mashup page about Australian culture. It also increases the user’s knowledge about the Australia page by ten percent of the current knowledge. This is defined in the mashup rule as follows,

begin
    personal.course = "culture"
    culture.Australia.knowledge ≤ 30
    update.culture.Australia.knowledge = culture.Australia.knowledge + (0.1 * _culture.Australia.knowledge)
    web.align = left
    local.align = bottom
    blog.align = center
    news.align = right
    book.align = right
    video.align = center
    image.align = left
end

Second, if a student takes the culture course and the student’s knowledge about the Australia page is more than ‘80’, the mashup system creates a mashup page about Australian culture by adding a query term, "Aborigines". It also increases the user’s knowledge about the Australia page by ten percent of the current knowledge. This is defined in the mashup rule as follows,

begin
    personal.course = "culture"
    culture.Australia.knowledge ≥ 80
    update.culture.Australia.knowledge = culture.Australia.knowledge + (0.1 * _culture.Australia.knowledge)
    web.align = left
    web.query = "Aborigines"
    image.align = bottom
    image.query = "Aborigines"
    image.expand = open
    blog.align = center
    blog.query = "Aborigines"
    local.align = false
    news.align = false
    book.align = false
    video.align = false
end

The teacher defines a simple ontology in OWL.
Left figure and right figure in Fig. 7 show the results when Students 1 and 2 send the same query "Australia" to the mashup system, respectively.

5. CONCLUSIONS AND FUTURE WORKS

In this paper, we present an ontology-based system by which users can create mash-ups using a rule language. We propose a mashup rule language that enables users without programming experiences to easily create mashups. The system generates a mashup based on the information contained in the mashup rule, ontology, and user’s query. It can also generate an adaptive mashup by utilizing users’ features in the user model of an adaptive hypermedia system. We show that how the proposed system can be used in an educational environment.

We plan to extend our system to support an adaptive recommendation service where Web resources are recommended according to a context rule defined by a user. A context rule is similar to adaptation rules and it can be utilized in constructing adaptive mashup as well. In addition, we plan to utilize the contextual search functionality [1] to construct
MASHUP CREATION USING A MASHUP RULE LANGUAGE

Adaptive mashups. It can search the documents suited to a certain searching situation by utilizing a context file specified by a user.

APPENDIX: A MASHUP RULE GRAMMAR

The grammar for mashup creation is as follows:

\[
\begin{align*}
\text{program} & \rightarrow \text{`begin'} \text{ statements} \text{`end'}; \\
\text{statements} & \rightarrow \text{ statement statements}; \\
\text{statement} & \rightarrow \text{ `web'} \text{ dot web_opt} | \text{`local'} \text{ dot local_opt} | \text{`blog'} \text{ dot blog_opt} | \text{`video'} \text{ dot video_opt} | \text{`news'} \text{ dot news_opt} | \text{`book'} \text{ dot book_opt} | \text{`image'} \text{ dot image_opt}; \\
\text{web_opt} & \rightarrow \text{ general_opt} | \text{site_opt}; \\
\text{local_opt} & \rightarrow \text{ general_opt} | \text{point_opt}; \\
\text{blog_opt} & \rightarrow \text{ general_opt} | \text{site_opt} | \text{order_opt}; \\
\text{video_opt} & \rightarrow \text{ general_opt} | \text{order_opt}; \\
\text{news_opt} & \rightarrow \text{ general_opt} | \text{site_opt} | \text{order_opt}; \\
\text{book_opt} & \rightarrow \text{ general_opt}; \\
\text{image_opt} & \rightarrow \text{ general_opt}; \\
\text{general_opt} & \rightarrow \text{ query_opt} | \text{align_opt} | \text{label_opt} | \text{expand_opt}; \\
\text{query_opt} & \rightarrow \text{ `query'} \text{ assign str}; \\
\text{align_opt} & \rightarrow \text{ `align'} \text{ assign align_val}; \\
\text{label_opt} & \rightarrow \text{ `label'} \text{ assign str}; \\
\text{expand_opt} & \rightarrow \text{ `expand'} \text{ assign expand_val}; \\
\text{site_opt} & \rightarrow \text{ `site'} \text{ assign str}; \\
\text{order_opt} & \rightarrow \text{ `order'} \text{ assign order_val}; \\
\text{point_opt} & \rightarrow \text{ `point'} \text{ assign str}; \\
\text{align_val} & \rightarrow \text{ `false'} | \text{`top'} | \text{`bottom'} | \text{`center'} | \text{`left'} | \text{`right'}; \\
\text{expand_val} & \rightarrow \text{ `open'} | \text{`closed'} | \text{`partial'}; \\
\text{order_val} & \rightarrow \text{ `date'} | \text{`relevance'}; \\
\text{assign} & \rightarrow \text{ `='}; \\
\text{dot} & \rightarrow \text{ `'.'}; \\
\text{let} & \rightarrow \text{ lower} | \text{ upper}; \\
\text{lower} & \rightarrow \text{ `a'} | \text{`b'} | \text{`c'} | \text{`d'} | \text{`e'} | \text{`f'} | \text{`g'} | \text{`h'} | \text{`i'} | \text{`j'} | \text{`k'} | \text{`l'} | \text{`m'} | \text{`n'} | \text{`o'} | \text{`p'} | \text{`q'} | \text{`r'} | \text{`s'} | \text{`t'} | \text{`u'} | \text{`v'} | \text{`w'} | \text{`x'} | \text{`y'} | \text{`z'}; \\
\text{upper} & \rightarrow \text{ `A'} | \text{`B'} | \text{`C'} | \text{`D'} | \text{`E'} | \text{`F'} | \text{`G'} | \text{`H'} | \text{`I'} | \text{`J'} | \text{`K'} | \text{`L'} | \text{`M'} | \text{`N'} | \text{`O'} | \text{`P'} | \text{`Q'} | \text{`R'} | \text{`S'} | \text{`T'} | \text{`U'} | \text{`V'} | \text{`W'} | \text{`X'} | \text{`Y'} | \text{`Z'}; \\
\text{digit} & \rightarrow \text{ `0'} | \text{`1'} | \text{`2'} | \text{`3'} | \text{`4'} | \text{`5'} | \text{`6'} | \text{`7'} | \text{`8'} | \text{`9'}; \\
\text{extra} & \rightarrow \text{ `' '}; \\
\text{str} & \rightarrow \text{ `"'} (\text{let} | \text{digit} | \text{extra}) \text{ `"'};
\end{align*}
\]

REFERENCES


---

1 Terminals are single quoted strings such as 'a', 'b', 'begin', 'end', 'web', etc. Non-terminals are unquoted strings such as assign, dot, let, web_opt, local_opt, etc.


Hyosook Jung received bachelor’s degree in the Department of Elementary Education from Seoul National University of Education in Seoul, Korea, and master’s degree in the Department of Elementary Computer Education from Seoul National University Graduate School of Education in Seoul, Korea and doctoral degree in the Department of Computer Science Education from Korea University in Seoul, Korea. Her research interests include semantic web, adaptive hypermedia and computer science education.

Seongbin Park received bachelor’s degree in the Department of Computer Science from Korea University in Seoul, Korea, and both master’s degree and doctoral degree in the Department of Computer Science from the University of Southern California. He is currently an Associate Professor at the Department of Computer Science Education of Korea University in Seoul, Korea. His research interests include semantic web, adaptive hypermedia and computer science education.