Links and Cycles in Web Databases

Mori, Masao
Office for Information of University Evaluation, Kyushu University

Tetsuya, Nakatoh
Research Institute for Information Technology, Kyushu University

Sachio, Hirokawa
Research Institute for Information Technology, Kyushu University

http://hdl.handle.net/2324/8713
Links and Cycles of Web Databases

Masao Mori¹, Tetsuya Nakatoh², and Sachio Hirokawa²

¹ Office for Information of University Evaluation, Kyushu Univ., Fukuoka, Japan. 
mori.uoc@mbox.nc.kyushu-u.ac.jp
² Research Institute for Information Technology, Kyushu Univ., Fukuoka Japan. 
{nakatoh, hirokawa}@cc.kyushu-u.ac.jp

Abstract. This paper proposes a novel framework for composing web databases. Web databases are assumed to have explicit descriptions of I/O attributes and are considered as components of functional compositions. A user writes a script to connect output channels and input channels of components. A script determines a directed graph that may contain cycles which formalizes interactive and iterative behavior of a user through a browser. The interaction and iteration are realised by the notion of CGI-link. Auxiliary filters are introduced as components for universal manipulating tools. (Keywords: web service composition, mashups)

1 Introduction

This paper proposes a novel framework for composing web databases. Under the framework we implemented a system which is open to public³.

Web databases, sometimes called deep webs[1], hidden webs or invisible webs, have been paid attention since around 1996 because of their huge amount of information. Recently many web databases have been newly reconstructed into web services, like Amazon.com, Google, and so on. Web services provide access methods (API) for their hidden databases. On the other hand, for the purpose of accessing web databases there are many researches of web wrappers. A web wrapper collects information by analyzing HTML codes output from the human interface of a web database, e.g. [7],[8],and [9]. By virtue of web wrappers and APIs web developers are motivated to create a web service composition and the new style of web contents — mashup. While BPEL[10] is one of outcome from research of web service composition, mashup is a new style of combination of web services. Many mashup sites are implemented using visualization of AJAX techniques and communications of the REST style. Sabbouh et al.[11] proposed the Web Mashup Scripting Language which provides a set of procedures of JavaScript in order to integrate web services. Yokoyama et al.[15] studied a framework of AJAX for lightweight implementations. Importance of componetization of web services and web databases has been pointed out in [14] and [13], before mashups obtained much attention as we see now.

³ Available at http://hyoka-inf.ofc.kyushu-u.ac.jp/%7Emori/research/PSM/
Mashups have two types of processing; server side processing and client side processing. As for client side processing AJAX become popular to realize mashups because mashups with AJAX are supposed to process light-weight data. In this paper we focus on server side processing because of heavy-weight data processing. Currently our system adopt REST style communications as for web services, and web crawling as for web databases.

It seems that most of mashups provide integration of data rather than integration of process flows. In fact most of mashup web sites use only two or three web services. They do not need complex descriptions of process flows. Focusing on integrating web service feeds, Tatemura et al.[12] proposed “Mashup Feeds” which retrieves multiple feeds from many sites and provides users with a set of tools to manipulate the collection.

Mori et al.[5] proposed a novel approach and its system that generates mashup CGIs by giving a simple description of web databases compositions and stores the mashup CGIs in order to reuse them. The problem left in the researches [5] and is the actual interface using web browsers. In this paper we propose graphical primitives for mashup and give solutions for the following questions:

1. What is an easier script style to combine web services and web databases?
2. How does the system manage to layout and display data from multiple web services?
3. What is a better way to carry out next mashup execution and search?

We will introduce the notion of “user interface component” which is a key primitive to layout and display data, and carrying out the next execution step of mashups.

The structure of the paper is organized as Fig.1. New proposals are marked with asterisks(*). Section 2 explains a standard architecture for implementing mashup which requires basic components and their composition. In section 3,
we analyze how users use web databases with browsers. As a result, we introduce “user interface components” as new auxiliary components. In section 4, “filter components” and graphical components are introduced. In section 5, we introduce the notion of links and cycles as new methods of composition. These methods capture the repeated interaction of between a user and web databases.

2 PSM Architecture

Our system consists of three parts: interface server, CGI generator and mashup server. When a user accesses the interface server, the server provides a web interface for the user to describe mashups. A description of mashup is called a mashup script. Once the interface server passes a mashup script to the CGI generator, the generator forms a mashup CGI which is stored in the mashup server. The mashup CGI is executed in the mashup server and performs administration of communication and data processing so that the user can reuse the mashup CGI. The architecture of our system is named as the Personally Scripting Meta-CGI architecture, PSM for short. The overview of the architecture is shown in Fig 2.

2.1 I/O Attributes and I/O Composition

We call the subjects that input and output in PSM, as component. A mashup script is essentially a graph over components: paths of the graph shows data flow amongst components and each edge shows correspondence of attributes in components. The syntax of mashup scripts will be introduced in the rest of this section.

Most of web services provide complex queries in their search functions. A complex query is composed of a tuple of keywords for which web services return

![Fig. 2. An overview of PSM](image)
collections of tuples as search result. Search functions of web services are provided with a URL of API and variables of API. In this paper we call names of variables attributes. We introduce two web services for example in Fig.2.1. The first one is Rhapsody which is an online music web service. The second example is Amazon Web Service whose API is for database of music products in Amazon.com. Note that these examples are excerpts from original web service API.

We define attributes of complex queries as input channels and attributes of tuples in search results from web services as output channels. We call both of them I/O channels of web services. In PSM data on I/O channels are collections of tuples.

2.2 Functional Composition

Functional composition of web services is data passing from output channels on one web service to input channels on another. A mashup script consists of descriptions of functional compositions. For example, in order to pass data from the output channel artist of Rhapsody to the channel ItemSearch of Amazon, the mashup script should have:

\[
\text{Rhapsody.artist} \rightarrow \text{Amazon.ItemSearch},
\]

We call a pair of components as a functional composition expression, fc-expression for short.

The mashup CGI starts to work when the initial query is given, so that the mashup script must include at least one description about the initial query. Let us consider a special component Start to output the initial query to web components.

\[
\text{Start.x} \rightarrow \text{Amazon.ItemSearch},
\]

The initial query might be complex, like

\[
\text{Start.k1:k2} \rightarrow \text{Rhapsody.artist:album},
\]

Keywords from the output channels k1 and k2 of Start are passed to the input channels artist and album of Rhapsody, respectively. A fc-expression with complex data passing is written with tuples of channels separated by colon.
2.3 The Syntax of Scripts

Now we define the mashup script with BNF. Note that \( fce \) denotes fc-expressions.

\[
\text{MashupScript} ::= \text{wslist} "|" \text{exps}
\]

\[
\text{wslist} ::= \text{wsname} \{\text{wsname}\} *
\]

\[
\text{exps} ::= \{\text{fce}\} "|" \{\text{fce}\} *
\]

\[
\text{fce} ::= \text{ws} "->" \text{ws}
\]

\[
\text{ws} ::= \text{wsname} "." \text{chan}
\]

\[
\text{chan} ::= \{\text{attr}\} " : " \{\text{attr}\} *
\]

\[
\text{attr} ::= \{\text{attrname}\} \{\text{attrname}\} "|" *
\]

\[
\text{wsname} ::= \text{"names of web services"}
\]

\[
\text{attrname} ::= \text{"names of attributes"}
\]

Asterisks "*" added to \( \text{attr} \) is a word separator which will be introduced in the next section. Like Rhapsody and Amazon, web services and web databases with structured I/O channels are called by web components.

3 User Interface Component and CGI link

Now we consider roles of web browsers in PSM. Web browsers display data from web components on client PCs. Since we suppose that data in PSM are

![PSM generator](image)

![SWAP2007.example](image)

Fig. 4. Interface server(left) and a generated CGI “SWAP2007_example.cgi”(right)
sent through structured I/O channels from some web component, web browsers obtain not as text but collections of tuples. We define user interface components, UIC for short, that receive collections of tuples into input channels and display them in appropriate forms (e.g., HTML tables \(<table>...</table>\)) on client PCs. We denote it by Output. If distinct UICs are required, we can distinguish them by indexing, like Output1, Output2 and so on.

As input channels of a UIC can be known by output channels of the web component, channels of UIC can be omitted. For example, the functional compositions to a UIC Output like:

\[
\text{Rhapsody.album:track} \rightarrow \text{Output.album:track},
\]

but we can write

\[
\text{Rhapsody.album:track} \rightarrow \text{Output},
\]

We note three things about channels of UICs. Firstly we set that all of UICs must have the same names of output channels as names of input channels while names of input channels are determined by web components. Secondly we note variability of input channels of UIC. In the case that a UIC is on the right hand side of fc-expression, input channels of the UIC depend on the web component on the left hand side of fc-expression. Thus input channels of UIC are variable. Thirdly, output channels of a UIC can be regarded as output from users. This idea is very important. We will study this idea in the rest of this section.
How and what do we find keywords to continue web search? In many cases keywords might be chosen from the previous results. Let us consider the mashup script that generates SWAP2007_example.cgi. The UIC Output1 appears both in the right hand side of the second line and in the left hand side of the third line. While Output1 of the second line displays a collection of tuples (album,track) from the web component Rhapsody, functional composition of the third line means to set hyperlinks on all words which appears at the “album” column in the table Output1. Those hyperlinks call SWAP2007_example.cgi that send those words as queries to ItemSearch of Amazon web service API. Seeing Fig.4 search results of SWAP2007_example.cgi with hyperlinks on “Return Of The Champion” are shown in the front window. We call hyperlinks generated by output channels of UICs, CGI links. Note that a loop appears in the 4th and the 7th lines of the script, and a component named as Histogram appears in the 8th and 9th lines. These notions are introduced in section 4 and 5.

Sometimes data in one column of a UIC forms series of keywords. For example, let us observe the search result from Amazon web service arisen by a CGI link of Output2 in Fig.6. Series of names of artists can be seen at the artist column. They are marked off one phrase (or word) with comma. In order to make a CGI link for each keyword, the word separator, asterisk *, is put after the concerned output channel of UICs, like the forth line of the script in Fig.5.

The right window of Fig.6 (c) is the result by clicking the CGI link of “Queen” (in the 6th row, “artist” column) which arise the search API of YouTube with keyword “Queen”.

Now we discuss the three questions posed in the first section. By giving graphs of components we resolve the first question. We introduce user interface components for data layout management(question 2). Finally we prepare the CGI link machinery in order to set triggers for next search(question 3).

4 Filters and Graphical Components

As we have seen, mashup scripts are essentially graphs over components. A path on the graphs can be regarded as a pipeline for collections of tuples. So that we have implemented filter components, like UNIX pipeline processing.

SortL, SortN To sort data with respect to the first input channel of a web component. SortL for lexicographic order and SortN for numbers. Input channels of the sort component can be omitted.

Rhapsody.track:artist:album -> SortL,
SortL.album:track:artist -> Output,
The result of this example would be ordered data with respect to music title (track) from Rhapsody.

4 The sample script is available at
http://hyoka-inf.ofc.kyushu-u.ac.jp/%7Emori/research/PSM/Generated
SWAP2007_example.cgi
and the description of YouTube API can be found in www.youtube.com.
Fig. 6. Executions of SWAP2007_example.cgi

(a) Search result for the query “Queen”

(b) Invocation of the query “Return Of The Champions” from Output1

(c) Invocation of the query “Queen” in the 6th row, artist column from Output2

(d) Invocation of Histogram component from Output1
Uniq To remove duplication of records.
Rhapsody.track:artist:album -> Uniq,
Uniq.artist:album -> Output,

Colvec Extract a column from the collection of tuples.
Rhapsody.artist -> Colvec,
Colvec.id:value -> Output,
Colvec extract the column artist of data from Rhapsody component.

Transpose Regarding the collection of tuples as a matrix, this filter transpose data.

I/O channels of sort and uniq filter components are variable as well as UICs, and input channels of filters in the right hand side of fc-expression can be omitted. Now graphical components are introduced.

Histogram To count the appearance of a specified attribute at the input channel item of this component and make histograms. Output channels are who for appeared keywords and num for numbers of appearance. See the 8th and 9th line of SWAP2007_example.cgi and the result in Fig.6 (d).

BarGraph, LineGraph, PointGraph Those components receive vectors and plot graphs.

We can generalize about the component in terms of the standard output from user interface components and graphical components. Those components involve CGI links not only on text, but also multimedia objects.

5 Links and Cycles

Mashup feeds[12] is designed to make programs to collect feeds periodically. It iterates procedures by time-based scheduling. This method is suitable for feeds processing. On the other hand WMSL[11] utilized the control structure of JavaScript for iteration.

Since mashup scripts are written in simple descriptions of graphs over components, they might include cycles in the graphs of components. Note that cycles play a role of iteration in PSM. If the cycle consists of only web components, its execution would result in an infinite loop. If the cycle includes at least one UIC, it is possible to stop the iteration at the UIC. Thus links and cycles in PSM can control loops.

See the mashup script SWAP2007_example.cgi again and note the 4th and 7th lines where a loop can be found. It is easy to presume that the mashup script would stop at each loop step by the CGI links in Output2.

6 Conclusion and Future Works

We proposed the mashup scripting system PSM which resolve the three proper questions for mashups introduced in the first section. The idea of functional composition of components leads us to a simple format (graphs) of mashup
scripts. Moreover we proposed the new mashup programming style like UNIX pipeline processing. In this style loops can be realized by cycles of components, and can be controlled by CGI links.

PSM is implemented in Perl, independent of WSDL[2]. Data from web services and web databases are transformed into lists of hash in perl codes. All of data processing are done in single server, so that we need to improve the system to reduce overhead.

References