Links and Cycles of Web Databases

Masao Mori[1], Tetsuya Nakatoh[2], and Sachio Hirokawa[2]

[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 public[3].

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,

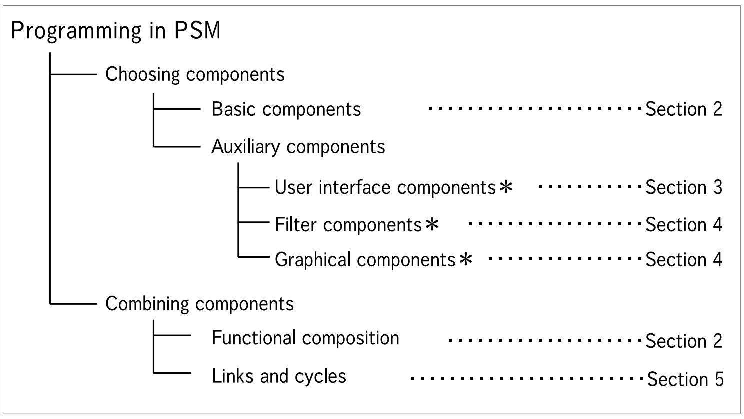


Fig. 1. The programming paradigm of PSM

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

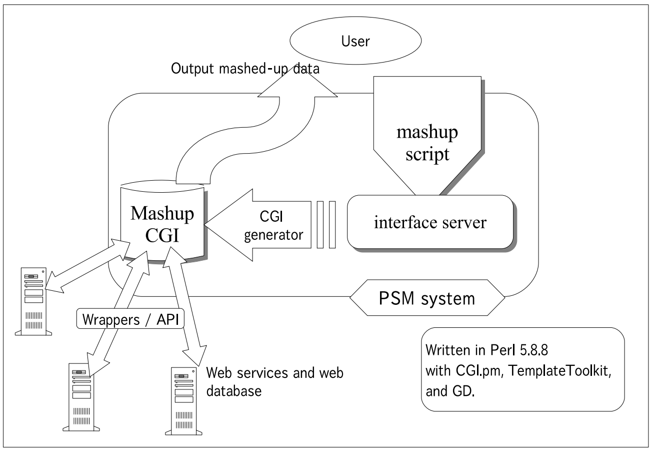


Fig. 2. An overview of PSM

	Rhapsody(`www.rhapsody.com`)		*Amazon*(`www.amazon.com`)	
	attribute	description	attribute	description
input	`artist`	name of artists	`ItemSearch`	keyword search
	`album`	names of CD titles	`ProductSearch`	product id search
output	`artist`	names of artists	`artist`	names of artists
	`album`	names of CD titles	`album`	names of CD titles
	`track`	url of the web page	`URL`	url of the web page

Fig. 3. API description of Rhapsody and Amazon

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:

```
Rhapsody.artist -> 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.

```
Start.x -> Amazon.ItemSearch,
```

The initial query might be complex, like

```
Start.k1:k2 -> 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

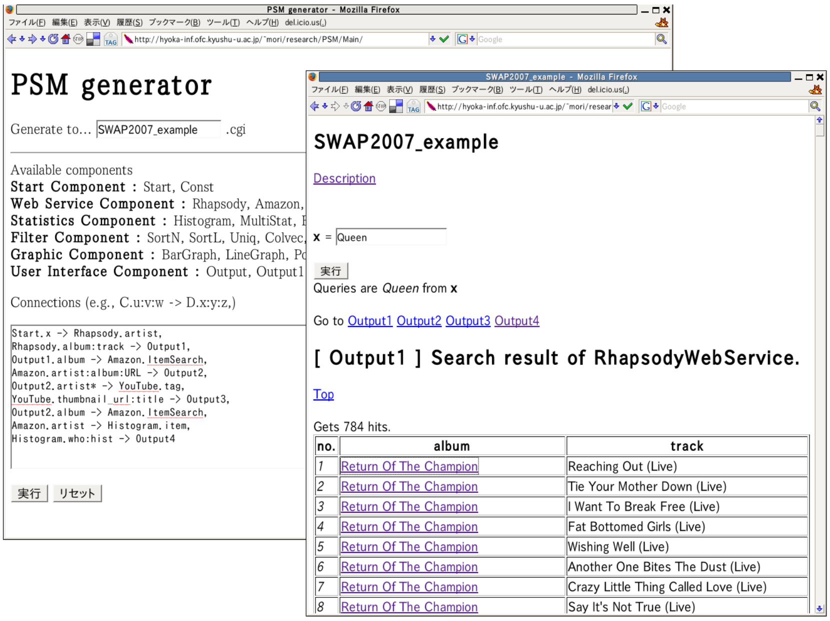
Now we define the mashup script with BNF. Note that $\langle fce \rangle$ denotes fc-expressions.

$$
\begin{aligned}
\langle MashupScript \rangle &::= \langle wslist \rangle \text{ "|" } \langle exps \rangle \\
\langle wslist \rangle &::= \langle wsname \rangle \{\text{","} \langle wsname \rangle\} * \\
\langle exps \rangle &::= \langle fce \rangle \{\text{","} \langle fce \rangle\} * \\
\langle fce \rangle &::= \langle ws \rangle \text{"->"} \langle ws \rangle \\
\langle ws \rangle &::= \langle wsname \rangle \text{"."} \langle chan \rangle \\
\langle chan \rangle &::= \langle attr \rangle \{\text{" : "} \langle attr \rangle\} * \\
\langle attr \rangle &::= \langle attrname \rangle \mid \langle attrname \rangle \text{" * "} \\
\langle wsname \rangle &::= \text{"names of web services"} \\
\langle attrname \rangle &::= \text{"names of attributes"}
\end{aligned}
$$

Asterisks "$*$" added to $\langle attr \rangle$ 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 - Mozilla Firefox
ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H) del.icio.us(.)
http://hyoka-inf.ofc.kyushu-u.ac.jp/~mori/research/PSM/Main/
Google
SWAP2007_example - Mozilla Firefox
ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H) del.icio.us(.)
http://hyoka-inf.ofc.kyushu-u.ac.jp/~mori/resear
Google

PSM generator

Generate to... SWAP2007_example .cgi

Available components
Start Component : Start, Const
Web Service Component : Rhapsody, Amazon,
Statistics Component : Histogram, MultiStat, I
Filter Component : SortN, SortL, Uniq, Colvec,
Graphic Component : BarGraph, LineGraph, Po
User Interface Component : Output, Output1

Connections (e.g., C.u:v:w -> D.x:y:z,)

```
Start.x -> Rhapsody.artist,
Rhapsody.album:track -> Output1,
Output1.album -> Amazon.ItemSearch,
Amazon.artist:album:URL -> Output2,
Output2.artist* -> YouTube.tag,
YouTube.thumbnail_url:title -> Output3,
Output2.album -> Amazon.ItemSearch,
Amazon.artist -> Histogram.item,
Histogram.who:hist -> Output4
```

実行 リセット

SWAP2007_example

Description

x = Queen

実行

Queries are *Queen* from **x**

Go to Output1 Output2 Output3 Output4

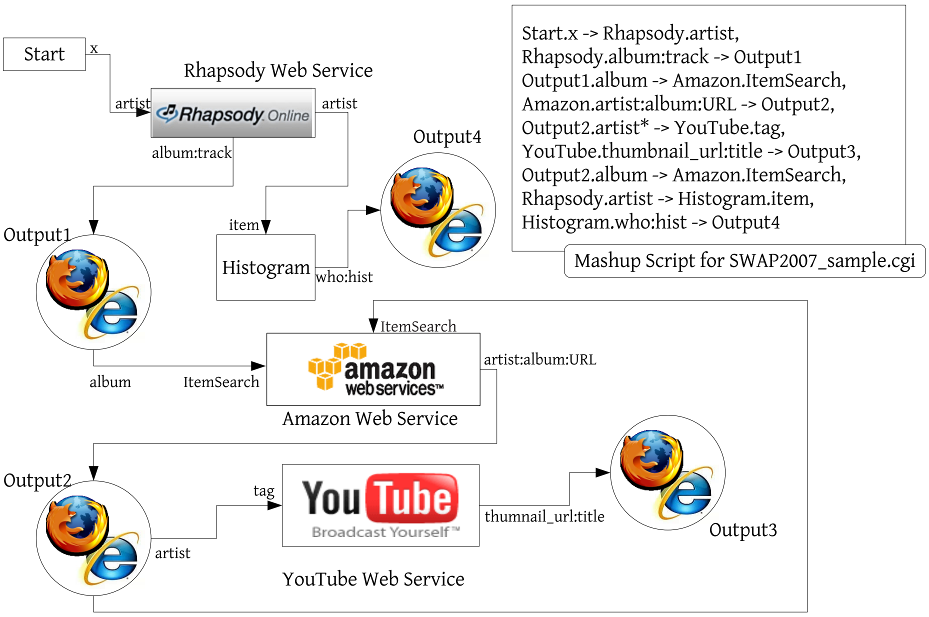
[Output1] Search result of RhapsodyWebService.

Top

Gets 784 hits.

no.	album	track
1	Return Of The Champion	Reaching Out (Live)
2	Return Of The Champion	Tie Your Mother Down (Live)
3	Return Of The Champion	I Want To Break Free (Live)
4	Return Of The Champion	Fat Bottomed Girls (Live)
5	Return Of The Champion	Wishing Well (Live)
6	Return Of The Champion	Another One Bites The Dust (Live)
7	Return Of The Champion	Crazy Little Thing Called Love (Live)
8	Return Of The Champion	Say It's Not True (Live)

Fig. 4. Interface server(left) and a generated CGI "`SWAP2007_example.cgi`"(right)



```
Start.x -> Rhapsody.artist,
Rhapsody.album:track -> Output1
Output1.album -> Amazon.ItemSearch,
Amazon.artist:album:URL -> Output2,
Output2.artist* -> YouTube.tag,
YouTube.thumbnail_url:title -> Output3,
Output2.album -> Amazon.ItemSearch,
Rhapsody.artist -> Histogram.item,
Histogram.who:hist -> Output4
```

Fig. 5. The mashup script and its graph for `SWAP2007_example.cgi`

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;

```
Rhapsody.album:track -> Output.album:track,
```

but we can write

```
Rhapsody.album:track -> 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`[4]. 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 `Rhaspody`, 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

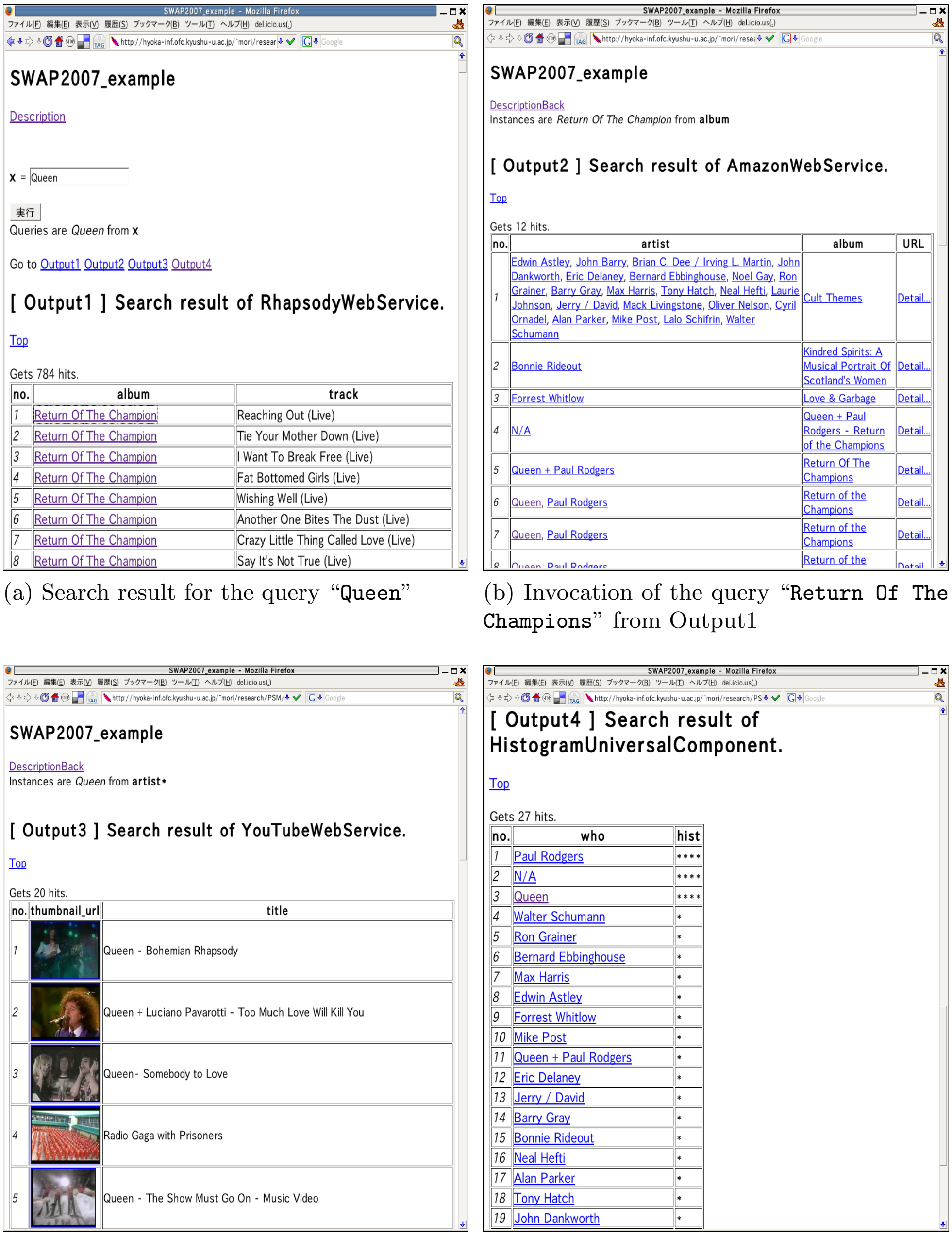
As we have seen, mashup scripts are essentially graphs over components. A path on the graphs can be regarded as a pipeline for collections of tuples. So that we have implemented *filter components*, like UNIX pipeline processing.

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`.


SWAP2007_example - Mozilla Firefox
ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H) del.icio.us(.)
http://hyoka-inf.ofc.kyushu-u.ac.jp/˜mori/resear
Google
実行

SWAP2007_example

Description

x = Queen

Queries are *Queen* from **x**

Go to Output1 Output2 Output3 Output4

[Output1] Search result of RhapsodyWebService.

Top

Gets 784 hits.

no.	album	track
1	Return Of The Champion	Reaching Out (Live)
2	Return Of The Champion	Tie Your Mother Down (Live)
3	Return Of The Champion	I Want To Break Free (Live)
4	Return Of The Champion	Fat Bottomed Girls (Live)
5	Return Of The Champion	Wishing Well (Live)
6	Return Of The Champion	Another One Bites The Dust (Live)
7	Return Of The Champion	Crazy Little Thing Called Love (Live)
8	Return Of The Champion	Say It's Not True (Live)

(a) Search result for the query “`Queen`”

SWAP2007_example - Mozilla Firefox
ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H) del.icio.us(.)
http://hyoka-inf.ofc.kyushu-u.ac.jp/˜mori/rese
Google

SWAP2007_example

DescriptionBack
Instances are *Return Of The Champion* from **album**

[Output2] Search result of AmazonWebService.

Top

Gets 12 hits.

no.	artist	album	URL
1	Edwin Astley, John Barry, Brian C. Dee / Irving L. Martin, John Dankworth, Eric Delaney, Bernard Ebbinghouse, Noel Gay, Ron Grainer, Barry Gray, Max Harris, Tony Hatch, Neal Hefti, Laurie Johnson, Jerry / David, Mack Livingstone, Oliver Nelson, Cyril Ornadel, Alan Parker, Mike Post, Lalo Schifrin, Walter Schumann	Cult Themes	Detail...
2	Bonnie Rideout	Kindred Spirits: A Musical Portrait Of Scotland's Women	Detail...
3	Forrest Whitlow	Love & Garbage	Detail...
4	N/A	Queen + Paul Rodgers - Return of the Champions	Detail...
5	Queen + Paul Rodgers	Return Of The Champions	Detail...
6	Queen, Paul Rodgers	Return of the Champions	Detail...
7	Queen, Paul Rodgers	Return of the Champions	Detail...
8	Queen, Paul Rodgers	Return of the	Detail

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

SWAP2007_example - Mozilla Firefox
ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H) del.icio.us(.)
http://hyoka-inf.ofc.kyushu-u.ac.jp/˜mori/research/PSM/
Google

SWAP2007_example

DescriptionBack
Instances are *Queen* from **artist***

[Output3] Search result of YouTubeWebService.

Top

Gets 20 hits.

no.	thumbnail_url	title
1		Queen - Bohemian Rhapsody
2		Queen + Luciano Pavarotti - Too Much Love Will Kill You
3		Queen- Somebody to Love
4		Radio Gaga with Prisoners
5		Queen - The Show Must Go On - Music Video

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

SWAP2007_example - Mozilla Firefox
ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H) del.icio.us(.)
http://hyoka-inf.ofc.kyushu-u.ac.jp/˜mori/research/PS
Google

[Output4] Search result of HistogramUniversalComponent.

Top

Gets 27 hits.

no.	who	hist
1	Paul Rodgers	****
2	N/A	****
3	Queen	****
4	Walter Schumann	*
5	Ron Grainer	*
6	Bernard Ebbinghouse	*
7	Max Harris	*
8	Edwin Astley	*
9	Forrest Whitlow	*
10	Mike Post	*
11	Queen + Paul Rodgers	*
12	Eric Delaney	*
13	Jerry / David	*
14	Barry Gray	*
15	Bonnie Rideout	*
16	Neal Hefti	*
17	Alan Parker	*
18	Tony Hatch	*
19	John Dankworth	*

(d) Invocation of Histogram component from Output1

Fig. 6. Executions of `SWAP2007_example.cgi`

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

[1] BrightPlanet. Deep web. White Paper, 2000.

[2] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. Technical report, World Wide Web Consortium, March 2001. `http://www.w3.org/TR/wsdl`.

[3] K. Hemenway and T. Calishain. *Spidering Hacks*. O'Reilly & Associates Inc., Mar. 2003. ISBN-13 978-0596005771.

[4] R. Milner. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.

[5] M. Mori, T. Nakatoh, and S. Hirokawa. Functional composition of web databases. In *Proceedings of International Conference Asian Digital Libraries 2006*, Lecture note in Computer Science 4312. Springer Verlag, 2006.

[6] M. Mori, T. Nakatoh, and S. Hirokawa. A light-weight implementation of mashups (in japanese). In *Proceedings of Data Engineering Workshop 2007*, C7-152. IEICE, 2007.

[7] T. Nakatoh, K. Ohmori, and S. Hirokawa. A report on metadata for web databases. In *IPSJ SIG Technical Reports*, 2004-ICS-138(17), pages 95–98, 2004.

[8] T. Nakatoh, K. Ohmori, Y. Yamada, and S. Hirokawa. Complex query and metadata. In *Proceedings of ISEE2003*, pages 291–294, 2003.

[9] T. Nakatoh, Y. Yamada, and S. Hirokawa. Automatic generation of deep web wrappers based on discovery of repetition. In *Proceedings of the First Asia Information Retrieval Symposium (AIRS 2004)*, pages 269–272, 2004.

[10] OASIS. *Web Services Business Process Execution Language Version 2.0*, April 2007. OASIS Standard.

[11] M. Sabbouh, J. Higginson, S. Semy, and D. Gagne. Web mashup scripting language. In *Proceedings of the 16th international conference on World Wide Web 2007*, pages 1305 – 1306. ACM Press, May 2007.

[12] J. Tatemura, A. Sawires, O. Po, S. Chen, K. S. Candan, D. Agrawal, and M. Goveas. Mashup feeds: Continuous queries over web services. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1128 – 1130. ACM Press, June 2007.

[13] J. Yang. Web service componentization. *Communications of the ACM*, 46(10):35–40, 2003.

[14] J. Yang and M. P. Papazoglou. Web component: A substrate for web service reuse and composition. In *Advanced Information Systems Engineering: 14th International Conference, CAiSE 2002 Toronto, Canada, May 27-31, 2002. Proceedings*, pages 21–36. Springer Verlag, May 2002.

[15] S. Yokoyama, A. Matono, S. M. Pahlevi, and I. Kojima. A framework for modularization and mashup of javascript codes on web2.0 (in japanese). *DBSJ Letters*, 5(3), December 2006.