Ousia Weaver: A tool for creating and publishing mashups as impressive Web pages

Ikuya Yamada[1,2] Wataru Yamaki[2] Hirotaka Nakajima[2,3] Yoshiyasu Takefuji[3]

[1]Graduate School of Media and Governance
Keio University
Endo 5322
Fujisawa, Kanagawa, Japan
+81-466-47-5111
ikuya@sfc.keio.ac.jp

[2]Studio Ousia Inc.
3F, Zoshigaya 2-5-12
Toshima Ward, Tokyo, Japan
+81-3-6915-2023
{yamaki, nakajima}@ousia.jp

[3]Faculty of Environment and Information Studies
Keio University
Endo 5322
Fujisawa, Kanagawa, Japan
+81-466-47-5111
takefuji@sfc.keio.ac.jp

ABSTRACT

Mashup is a technique which combines data and functionalities from multiple Web sources into new Web contents. Recently, software tools called mashup editors have attracted attention. These tools are typically targeted at non-expert users and allow them to create mashups without writing code. However, these tools tend to focus only on creating mashups and are lacking in publishing mashups. Therefore, if users want to publish results of mashups on the Web, they eventually have to do complex programming and/or settings.

In this paper, we propose a novel mashup editor called Ousia Weaver. It provides a sophisticated visual editor which enables users to create mashups without writing code. It addresses not only on creating mashups, but also on publishing mashups. With the proposed tool users can visualize their mashup results by simply choosing desired visualization widgets from the list of the widgets. It also involves a simple Web server which automatically publicizes mashup results on the Web. Therefore, users can create impressive Web pages from mashup results without any complex programming and/or settings. This paper presents the programming model, the user interface, and the implementation of Ousia Weaver, and also provides two concrete examples in order to demonstrate its effectiveness.

Categories and Subject Descriptors

D.1.7 [**Software**]: Programming Techniques — *Visual Programming*
; H.4.m [**Information Systems**]: Miscellaneous

General Terms

Design, Languages

Keywords

Mashup, End-user programming, Web 2.0

1. INTRODUCTION

Mashup is a technique which combines data and functionalities from multiple Web sources into new Web contents. One of the earliest mashup examples is *housingmaps.com*, which allows users to see real estate information visually on a map. It obtains a list of real estate information from Craigslist (www.craigslist.org), extracts locations from each one, and puts them onto a map using Google Maps API (code.google.com/apis/maps/).

Recently, software tools called mashup editors have attracted attention, such as Yahoo! Pipes [10], Microsoft Popfly [4], CMU Marmite [9], and Intel MashMaker [3,2]. These tools are typically targeted at non-expert users and allow them to create mashups without writing code.

However, these tools tend to focus on creating mashups and are lacking in publishing mashups. Therefore, if users want to publish results of mashups on the Web, they eventually have to do complex programming and/or settings.

In this paper, we propose a novel mashup editor called Ousia Weaver (see Figure 1). It provides a sophisticated visual editor which enables users to create mashups without writing code.

Ousia Weaver addresses not only on creating mashups, but also on publishing mashups. It provides the following three main components in order to publish mashup results as impressive Web pages: (1) *visualization widgets* which richly visualize mashups, (2) *data transformation operators* which add attributes to data and transform data to visualizable form, and (3) *mashup server* which provides a simple Web server functionality, and automatically publicizes mashup results on the Web.

With Ousia Weaver, users can visualize their mashups by simply choosing desired visualization widgets from the list of the widgets. When the user finishes defining how the mashup collects, combines, and processes data, a list of possible visualization widgets is automatically provided by Ousia Weaver. After the user selects desired widgets from the list, the mashup server automatically publicizes the visualized result of the mashup as a Web page.

In addition, Ousia Weaver is even powerful for experienced programmers. It has the following three distinguished characteristics which increase the convenience for programmers: (1) *parallel mashup execution* which enables programmers to execute mashup concurrently by multiple processes, (2) *extensible architecture* which allows programmers to add and update any desired operations into Ousia Weaver, and (3) *greater expressive power* which enables programmers to create complex mashups.

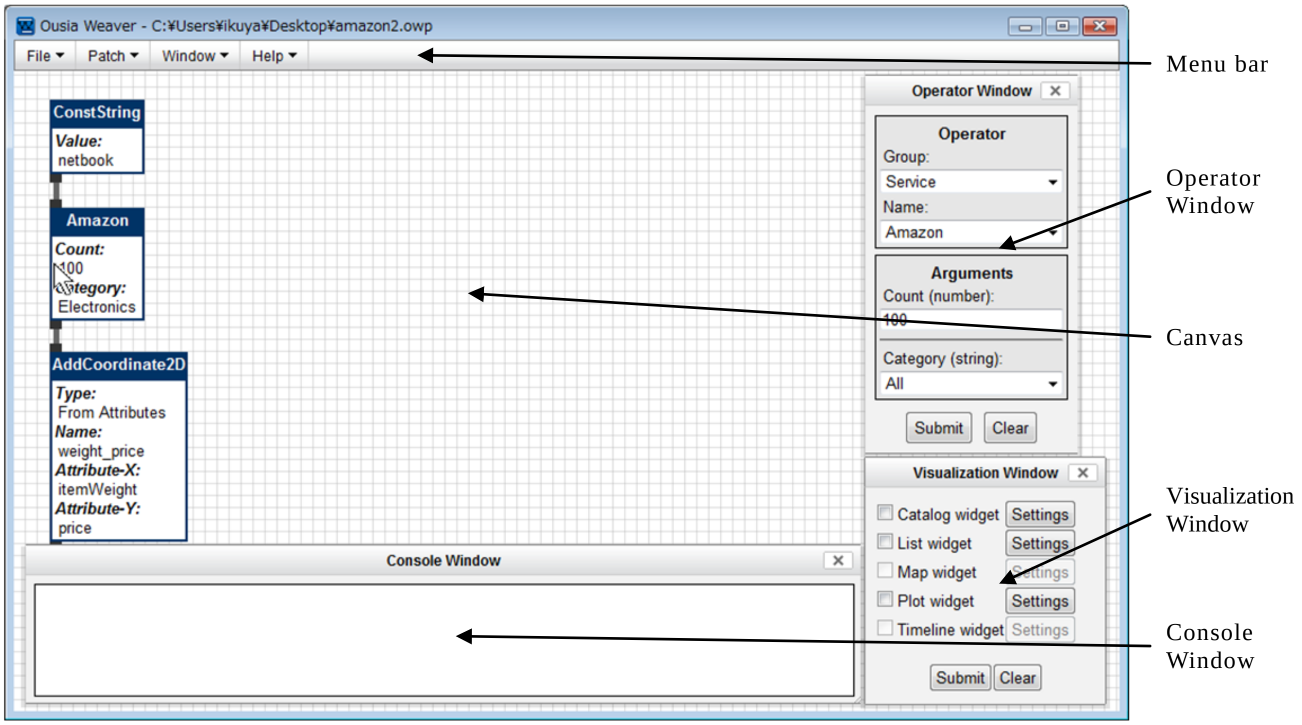


Figure 1. The user interface of Ousia Weaver.

2. THE OUSIA WEAVER SYSTEM

Ousia Weaver consists of the five fundamental components: *operators*, *data type system*, *mashup server*, *visualization widgets*, and *visual editor*. First we will overview this system, and then describe each of these components respectively.

2.1 What is Ousia Weaver?

Ousia Weaver is a mashup editor for help create and publish mashups. It provides a sophisticated *visual editor* and adopts a novel programming model which makes creating and publishing mashups easier.

The mashup programming in Ousia Weaver is composed of *data phase* and *visualization phase*. In the data phase, users create *mashup data-flows* which represent rules of collecting, combining and processing data, and in the visualization phase, users define how to visualize the result obtained by the mashup data-flows.

First, we describe the data phase. With Ousia Weaver, users can create mashup data-flows by stringing together components called *operators* in the visual editor. In particular, a mashup data-flow is represented as a directed graph structure (see Figure 2). Nodes, which we call operators, represent data operations such as collecting, combining, and processing data, and edges, which we call *connections*, connect two operators and define data path between these.

The typical way of creating a mashup data-flow is an iterative process that comprises of adding desired operators and stringing operators together using connections. The basic action in this process is *drag-and-drop*; users can create, move, and string together operators by drag-and-drop actions.

Mashup data-flows are executed in a data-flow manner. For example, the mashup data-flow shown in Figure 2 is executed as follows: (1) *Input operator* receives keywords from a user, (2)

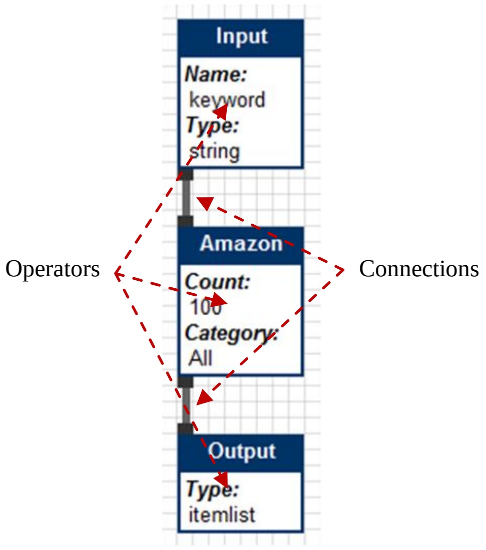


Figure 2. An example of mashup data-flow.

Amazon operator performs a search on *Amazon.com* with the keywords, and (3) *Output operator* returns the search result.

Also, the user can visualize the result of the mashup data-flow by simply selecting desired *visualization widgets* from the list of the widgets. After the user defines a mashup data-flow, Ousia Weaver automatically transits to the visualization phase and provides a list of visualization widgets which can be applied to the mashup data-flow.

Ousia Weaver also includes a simple Web server called *mashup server*. It automatically assigns a URL to each mashup and makes the mashup accessible to external users.

Therefore, with Ousia Weaver, users can create Web pages from mashups results without any complex programming and/or settings.

2.2 Operators

Operator is the most fundamental component which represents all data operations in Ousia Weaver. Figure 3 shows *Append operator* which appends a string to the input. As you can see in Figure 3, an operator consists of *input/output terminals*, *operator name*, and *argument name/value* pairs.

All operators can be classified into the following three groups:

1. **Base operator**

   These operators provide basic programming operations, including arithmetic operations, data type conversions, regular expressions, and so forth. They also contain standard selection operations, including *If* and *Switch*, and Loop operations, including *While*, and *Foreach*.

2. **Data source operator**

   These operators represent Web data sources. Typical examples of these operators are data extraction operators including *RSS* and *CSV*, and Web API operators including *Amazon*, *Google*, *Youtube*, and so forth.

3. **Data transformation operator**

   These operators perform data transformation operations. Typically, they are used to transform data into visualizable form. They include *AddImage* which adds images to data, *AddLocation* which appends geographical locations to data, *AddDate* which attaches dates to data, *AddDateRange* which adds ranges of dates to data, and *AddCoordinate2D*, which appends two dimensional coordinates to data.

Moreover, operators are extensible; users can add and update arbitrary operators whenever they need. Figure 4 shows source files of Append operator. Internally, an operator is built from two files; (a) a simple metadata file and (b) a program code file. The metadata is implemented using JavaScript and the program code Python.

Ousia Weaver also provides code templates in order to help users create these files. With these templates, users can create new operators by writing just a little amount of code fragments. In the case of Append operator, for instance, users have to write code fragments which are marked in bold in Figure 4.

Additionally, in Ousia Weaver, loop operations are represented using *sub mashup data-flows*; an operator involving loop has a separate sub mashup data-flow as its argument, and the data-flow is executed on each iteration of the loop.

2.3 Data type system

Ousia Weaver offers users a simple data typing system. Table 1 shows a list of all data types. As you can see, most of them are intuitive, but there are several unique data types including *any*, *item*, *itemlist*, and *data-flow*.

any is a data type which can represent any data including other data types.

itemlist represents mashup results. Internally, it is an array which contains multiple *item*. In addition, *item* and *itemlist* are represented using JSON (JavaScript Object Notation) format.

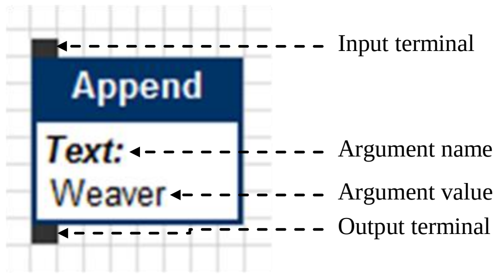


Figure 3. Append operator.

```
 ousiaweaver.Operators.Append = function(){
   this.addInputTerminal("string", "Input");
   this.addOutputTerminal("string", "Output");
}
ousiaweaver.Operators.Append.packageName =
   "base.string";
ousiaweaver.Operators.Append.arguments = {
   "text": {
      label: "Text",
      type: "string",
      required: true,
   }
}
```

(a) Append.js

```
 class AppendTerminal(OperatorBaseTerminal):
   def __init__(self,args):
      OperatorBaseTerminal.__init__(self, args)
      Input_terminal = {}
      input_terminal["Input"] = None
      self.set_input_terminals(input_terminal)
class AppendBody(OperatorBaseBody):
   def __init__(self, args, manager_args, worker):
      OperatorBaseBody.__init__(self, args,
         manager_args, worker)
   def run(self, input_terminals):
      output_terminals = {}
      output_terminals["Output"] =
         str(input_terminals["Input"]) +
         str(self.get_args()["text"])
      return output_terminals
```

(b) Append.py

Figure 4. The source files of Append operator.

data-flow represents a mashup data-flow. This data type is typically used in operators which involve loop operations. As we described above, these operators have sub mashup data-flows as their argument.

In addition, a terminal has a data type property, and two terminals, which have different data type properties, cannot be connected unless either or both properties are *any*. This concept is also implemented in the visual editor. We will describe this further later.

2.4 Visualization widgets

Visualization widgets are components which visualize mashup results. Figure 5 shows samples of visualization widgets. Each widget is shown as tab at the top of screen and users can see desired widget by clicking on the corresponding tab.

Currently, Ousia Weaver offers users the following five visualization widgets (see Figure 5).

1. **Catalog widget**

   This widget visualizes the mashup result like a catalog. It arranges items of the mashup result on the screen using their thumbnails, titles, and descriptions.

2. **Plot widget**

   This widget plots items of the mashup result on a two dimensional space using corresponding coordinates.

3. **Table widget**

   This widget presents all titles and descriptions of items in the mashup result in a table.

4. **Map widget**

   This widget depicts each item of the mashup result visually on a map using a corresponding geographical location.

5. **Timeline widget**

   This widget puts items of the mashup result into a graphical timeline using corresponding dates.

Note that, some visualization widgets are closely related to some data transformation operators. For example, the catalog widget requires thumbnails, and users can add them to the data using the AddImage operator. Similarly, the plot widget, the map widget, and the timeline widget require coordinates, geographical locations, and ranges of date, and these data can be obtained using the AddCoordinate2D operator, the AddLocation operator, and the AddDateRange operator respectively.

In addition, each visualization widget has a function that returns whether or not the widget can visualize the mashup. These functions scan results of the mashups and detect if the results have all data which are required to be visualized.

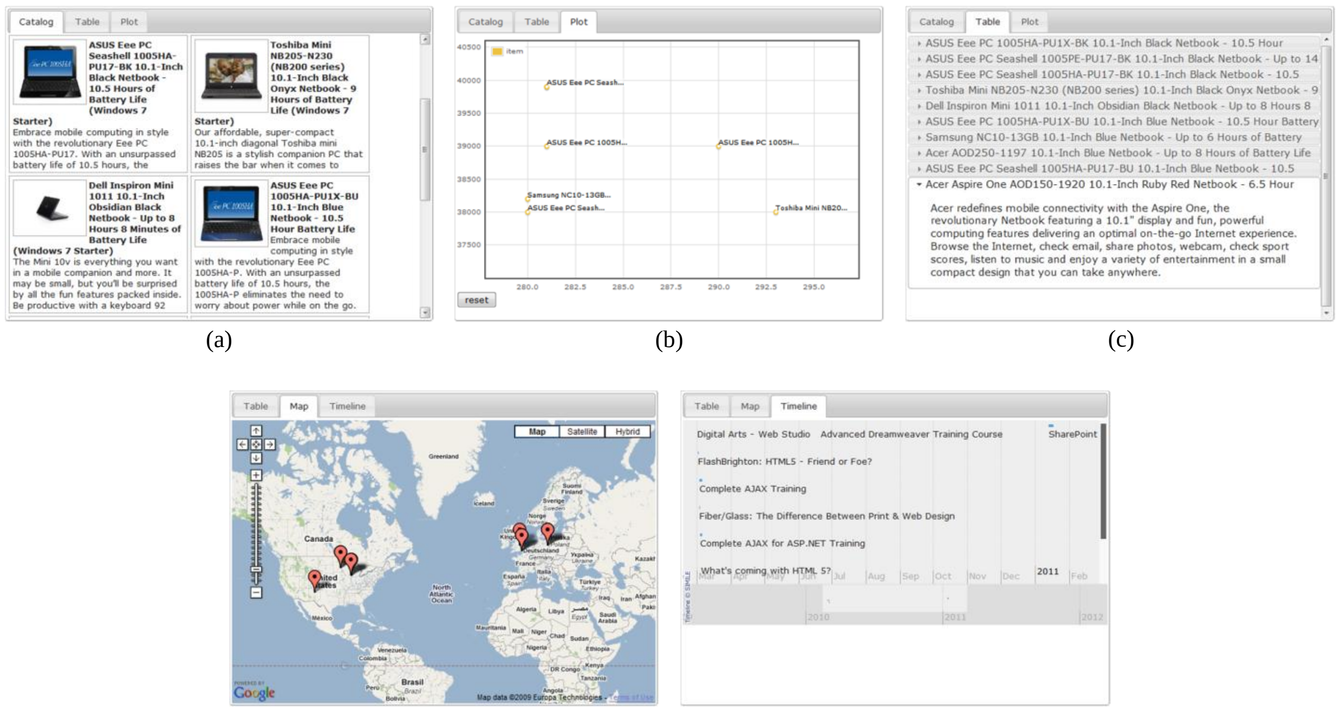
2.5 Mashup server

Mashup server is a lightweight server which executes mashups and makes the mashup results accessible to external users. It automatically assigns a URL to each mashup and dynamically executes the mashup when an external user accesses it. For example, the URL of a mashup named "search_videos" in the "foo" server would be "http://foo/search_videos/" and the external users can see the mashup result by accessing the URL.

The mashup server also caches mashup results. It assigns a unique identifier to each mashup result. Users can obtain previous mashup results by specifying their identifiers. (e.g., "http://foo/search_videos/result_identifier"). In addition, the

Table 1. A list of data types of Ousia Weaver.

Name	Description	Example
any	Can represent any data.	"text" / TRUE / 10 / [100, 200]
string	Represents a string.	"text"
boolean	Represents a boolean.	TRUE /FALSE
number	Represents a number.	10 / 0.11
coordinate	Represents a coordinate.	[100, 200]
datetime	Represents a date and a time.	Tue Jan 19 2010 00:00:34 GMT+0900
daterange	Represents a range of dates.	{ start: Tue Jan 19 2010, end: Sat Jan 23 2010 }
location	Represents a geographical location.	{ latitude: …, longitude: … }
image	Represents an image.	{ uri: http://www.keio.ac..., width: 200, height: 50, format: "png" }
uri	Represents a URI.	http://www.keio.ac.jp
array	Represents an array.	[0, 1, 2]
object	Represents an object (hash-table).	{ key1: "value1", key2: "value2" }
item	Represents a constituent of mashup result.	{ title: "Keio Univ.", uri: http://www.keio.ac..., description: "…" }
itemlist	Represents a mashup result.	[{ title: "Keio Univ.", uri: http://www.keio.ac... description: "…" }, …]
data-flow	Represents a mashup data-flow.	{ operators: […], connections: […] }



ASUS Eee PC Seashell 1005HA-PU17-BK 10.1-Inch Black Netbook - 10.5 Hours of Battery Life (Windows 7 Starter)
Embrace mobile computing in style with the revolutionary Eee PC 1005HA-PU17. With an unsurpassed battery life of 10.5 hours, the

Toshiba Mini NB205-N230 (NB200 series) 10.1-Inch Black Onyx Netbook - 9 Hours of Battery Life (Windows 7 Starter)
Our affordable, super-compact 10.1-inch diagonal Toshiba mini NB205 is a stylish companion PC that raises the bar when it comes to

Dell Inspiron Mini 1011 10.1-Inch Obsidian Black Netbook - Up to 8 Hours 8 Minutes of Battery Life (Windows 7 Starter)
The Mini 10v is everything you want in a mobile companion and more. It may be small, but you'll be surprised by all the fun features packed inside. Be productive with a keyboard 92

ASUS Eee PC 1005HA-PU1X-BU 10.1-Inch Blue Netbook - 10.5 Hour Battery Life
Embrace mobile computing in style with the revolutionary Eee PC 1005HA-P. With an unsurpassed battery life of 10.5 hours, the 1005HA-P eliminates the need to worry about power while on the go.

40500	40000	39500	39000	38500	38000	37500

280.0	282.5	285.0	287.5	290.0	292.5	295.0

- ASUS Eee PC 1005HA-PU1X-BK 10.1-Inch Black Netbook - 10.5 Hour
- ASUS Eee PC Seashell 1005PE-PU17-BK 10.1-Inch Black Netbook - Up to 14
- ASUS Eee PC Seashell 1005HA-PU17-BK 10.1-Inch Black Netbook - 10.5
- Toshiba Mini NB205-N230 (NB200 series) 10.1-Inch Black Onyx Netbook - 9
- Dell Inspiron Mini 1011 10.1-Inch Obsidian Black Netbook - Up to 8 Hours 8
- ASUS Eee PC 1005HA-PU1X-BU 10.1-Inch Blue Netbook - 10.5 Hour Battery
- Samsung NC10-13GB 10.1-Inch Blue Netbook - Up to 6 Hours of Battery
- Acer AOD250-1197 10.1-Inch Blue Netbook - Up to 8 Hours of Battery Life
- ASUS Eee PC Seashell 1005HA-PU17-BU 10.1-Inch Blue Netbook - 10.5
- Acer Aspire One AOD150-1920 10.1-Inch Ruby Red Netbook - 6.5 Hour

  Acer redefines mobile connectivity with the Aspire One, the revolutionary Netbook featuring a 10.1" display and fun, powerful computing features delivering an optimal on-the-go Internet experience. Browse the Internet, check email, share photos, webcam, check sport scores, listen to music and enjoy a variety of entertainment in a small compact design that you can take anywhere.

Digital Arts - Web Studio
Advanced Dreamweaver Training Course
SharePoint
FlashBrighton: HTML5 - Friend or Foe?
Complete AJAX Training
Fiber/Glass: The Difference Between Print & Web Design
Complete AJAX for ASP.NET Training
What's coming with HTML 5?

Figure 5. Samples of visualization widgets. (a), (b), (c), (d), and (e) corresponds catalog widget, plot widget, table widget, map widget, and timeline widget respectively.

mashup server can be configured to return the latest cache result of the mashup, instead of execute the mashup dynamically.

2.6 Visual editor

Visual editor (see Figure 1) enables users to create mashups visually and interactively. It mainly consists of the following three components: *canvas*, *operator window*, and *visualization window*.

2.6.1 Canvas

Canvas (see Figure 1) is the main part of the visual editor where all operators and connections are displayed.

The basic action on the canvas is drag-and-drop; users can move operators and create connections by drag-and-drop actions. Figure 6 shows how to create a connection on the canvas. As you can see, users can create a connection by dragging one terminal and dropping it onto the other terminal.

Also, Ousia Weaver has a useful feature which prevents users from bugs caused by data type mismatch. When users start dragging one terminal, other terminals will automatically be disabled and turn gray if they cannot be connected with the dragged terminal.

2.6.2 Operator window

Operator window allows users to create operators and change arguments of existing operators. Figure 7 shows the user interface of the operator window. When users click on the *submit button*, operator window adds a new operator to the canvas or changes arguments of an existing operator on the canvas.

It also contains the *operator field* and *arguments field*. The operator field enables users to easily select a desired operator from the *name selection box* shown at the top of the operator field. Also, it is possible to narrow down operators shown in the name

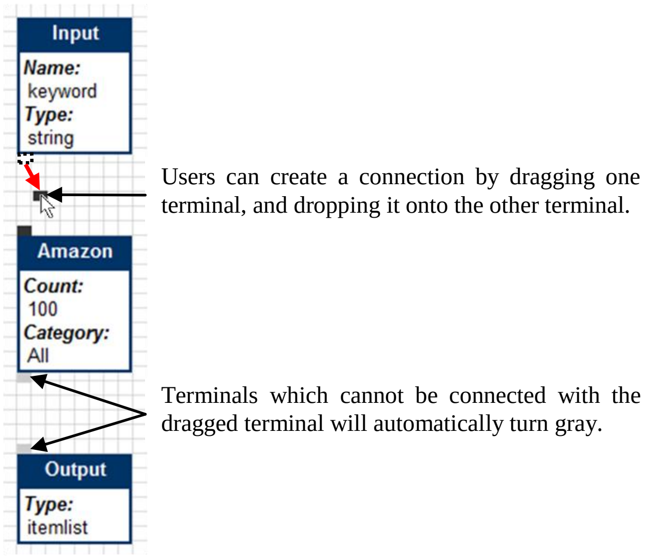

Input
Name:
keyword
Type:
string
Amazon
Count:
100
Category:
All
Output
Type:
itemlist
Users can create a connection by dragging one terminal, and dropping it onto the other terminal.
Terminals which cannot be connected with the dragged terminal will automatically turn gray.

Figure 6. Creating a connection on the canvas.

selection box by specifying the group of operators using the *group selection box*.

The arguments field allows users to specify arguments to operators. Each argument is shown separately on the field and users can easily enter any desired values.

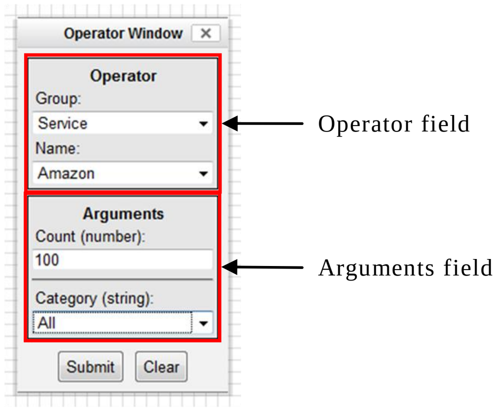


Figure 7. The user interface of operator window.

2.6.3 *Visualization window*

Visualization window allows users to select visualization widgets which will be used to visualize the mashup. Figure 8 shows the user interface of the visualization window. With this interface, users can choose desired visualization widgets by simply clicking on the checkboxes.

Note that, visualization widgets which cannot be applied to the mashup data-flow are automatically disabled and cannot be selected. The visualization window asks each visualization widget whether the result of the current mashup data-flow can be visualized or not.

There are also *Setting buttons* to the right of the checkboxes. It enables users to customize corresponding visualization widgets.

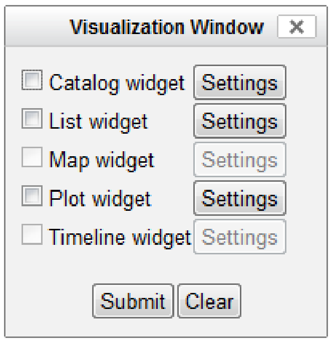


Figure 8. The user interface of visualization window.

3. IMPLEMENTATION

Figure 9 shows the architecture of Ousia Weaver. It consists of five main components; *operator manager*, *run-time engine*, *visualizer*, *visual editor*, and *mashup server*.

3.1 Operator manager

Operator manager is a simple component implemented in Python. It manages the list of all operators and delivers their program codes and metadata to other components.

Source files of all operators are stored in a special directory called *operator directory*. The operator manager periodically scans modifications in the directory and updates the list of operators if new modifications are detected.

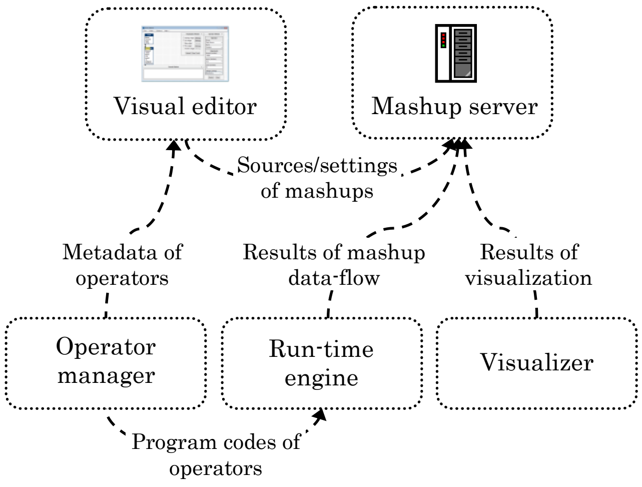


Figure 9. The architecture of Ousia Weaver.

3.2 Run-time engine

Run-time engine is a component responsible for executing mashup data-flows. It executes mashup data-flows in parallel; it divides a mashup data-flow into independent tasks and executes them using *worker processes*.

Figure 10 illustrates how the run-time engine executes mashup data-flows. It first creates a manager process, which controls the execution and watches the execution progress continually. Next the manager process extracts all operators from the data-flow and adds them to a queue called *task queue* as independent tasks. Then, the worker processes incrementally (1) take tasks from the queue, (2) obtain intermediate results required to perform the tasks from a buffer named *result buffer*, (3) obtain corresponding program codes from the operator manager, (4) execute these tasks, and (5) insert the results into the result buffer. Finally, the manager process detects the completion of the execution and returns the result.

As we described above, operators which involve loop operations have separate sub data-flows. Therefore, these operators are treated as independent data-flows and assigned to separate manager processes.

In addition, since creating a new process costs computational resources, the manager processes and the worker processes are recycled across mashup executions.

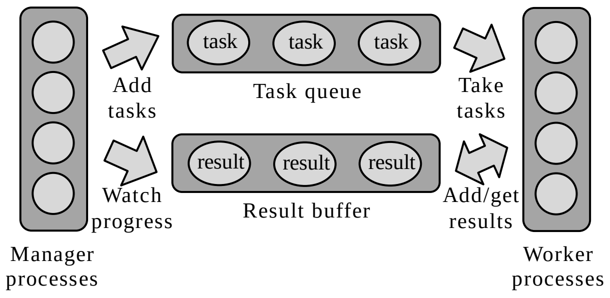


Figure 10. The execution model. The mashup is divided as multiple, independent tasks and executed in parallel.

3.3 Visualizer

Visualizer is a component which visualizes mashups. It contains visualization widgets inside. It passes the mashup result to each of selected visualization widgets, and simply adds the widgets to the screen as tabs.

The visualizer is implemented in JavaScript and uses the following four programming libraries:

- jQuery (jquery.com)
- jQuery Templates (plugins.jquery.com/project/jquerytemplate/)
- Simile Widgets (simile-widgets.org)
- Google Maps API (code.google.com/apis/maps/)

3.4 Visual editor

Visual editor provides rich visual editing interface for creating and publishing mashups. It is implemented using JavaScript and HTML, and can be run on most of commonly used Web browsers. It can also be executed as a stand-alone application using Mozilla's XUL technology (developer.mozilla.org/en/XUL/). Currently, it can be run on Windows, Mac OS X, and Linux.

In addition, operators, connections and terminals are drawn using HTML DOM nodes and styled using CSS. Also, we use Yahoo! User Interface Library (developer.yahoo.com/yui/) for implementing basic user interfaces including menus and windows.

3.5 Mashup server

Mashup server makes mashups accessible to external users. It provides a simple Web server which returns mashup results to external users, and a simple database which stores sources of mashups and caches of previous mashup results.

Figure 11 illustrates the basic sequence of operations of the mashup server. When (1) an external user accesses a mashup, (2) the mashup server executes the mashup using the runtime engine, (3) visualizes the result using the visualizer, and (4) returns the visualized result to the user.

The mashup server is written in Python and uses CouchDB (couchdb.apache.org) which is a simple database management system that can be used through Web APIs.

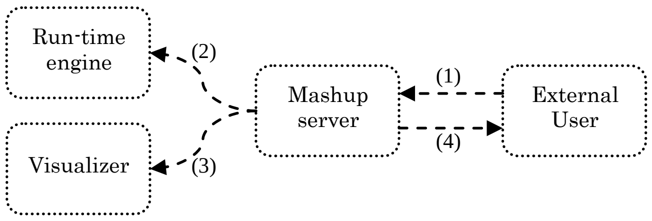


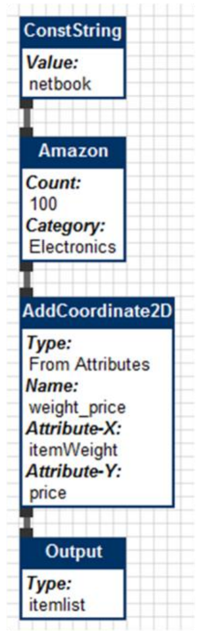
Figure 11. The basic sequence of operations of the mashup server.

4. MASHUP EXAMPLES

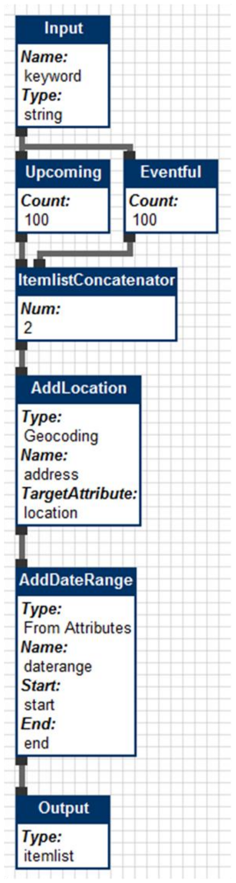
In this section, we will provide two concrete mashup examples: *netbook mashup* and *event mashup*, and explain how users can create mashups using Ousia Weaver.

4.1 Netbook mashup

First, we provide an example named "netbook_mashup". It is a simple mashup which performs a search on *Amazon.com* with the keyword "netbook", and visualizes the result using three visualization widgets.



(a)



(b)

Figure 12. (a) netbook mashup example and (b) event mashup example.

Figure 12a shows the mashup data-flow of this mashup. It is executed as follows: (1) *ConstString operator* outputs the keyword "netbook", (2) *Amazon operator* performs a search on *Amazon.com* with the keyword, (3) *AddCoordinate2D operator* adds coordinates to the result, and finally, (4) *Output operator* returns the result.

Note that, this mashup adds a coordinate to each item of the search result using the AddCoordinate2D operator. This operator extracts an item from the search result in sequence, create a two dimensional coordinate from the "itemWeight" attribute (specified as the third argument) and the "price" attribute (specified as the forth argument), and adds the coordinate as a new attribute named "weight_price" (specified as the second argument) to the item.

Also, this mashup is associated with the following three visualization widgets: c*atalog widget*, *plot widget*, and *table widget* (see Figure 5a, 5b, 5c) and users can obtain these visualizations by accessing "http://server_name/netbook_mashup". Note that, the plot widget uses the coordinates added by the AddCoordinate2D operator.

4.2 Event mashup

Our second mashup example is "event_mashup". This mashup is a little more complex than previous one. It searches events on *upcoming.yahoo.com* and *eventful.com* with a keyword specified by the user, gathers and processes these results, and visualizes the result using three visualization widgets.

Figure 12b shows the mashup data-flow of this mashup. It is executed as follows: (1) *Input operator* takes an input from the user, (2) *Upcoming operator* and *Eventful operator* search events on upcoming.yahoo.com and eventful.com, (3) *ItemlistConcatenator operator* concatenates these search results, (4) *AddLocation operator* and *AddDateRange operator* add locations and ranges of dates of the events to the concatenated result, and finally (5) *Output operator* outputs the result.

Note that, in order to add locations, the AddLocation operator uses a technique called *geocoding*; it extracts the address from each event and converts it into the geographical location using Google Maps API.

This mashup is tied to the following three visualization widgets: *table widget*, *map widget* and *timeline widget*. Figure 5d and 5e show part of the result of this mashup executed with the keyword "JavaScript" (Figure 5d corresponds to the map widget and Figure 5e to the timeline widget). Additionally, the map widget and the timeline widget use the locations and the ranges of dates added by the AddLocation operator and the AddDateRange operator.

5. RELATED WORKS

One of the earliest mashup editors is Yahoo! Pipes [10]. It adopts similar programming model to Ousia Weaver; user can create mashups by adding and stringing together its operators. It offers users a rich visual editor which allows users to create mashups by drag-and-drop actions. It also provides a "badge" feature which allows users to integrate its mashup results into existing Web sites. However, it is impossible to publish mashup results as independent Web pages. Moreover, its operators cannot be extended by users. This greatly limits the variety of mashups that can be created using Pipes.

CMU Marmite [9] is a similar tool to Pipes. In addition to Pipes, it suggests possible operators which can be connected to existing ones. It also supports mashup visualization using operators called "sinks". However, the visualization is poor and it doesn't support to publish mashup results on the Web.

Microsoft Popfly [4] is also a similar tool to Pipes and Marmite. It has a lot of significant advances over above tools. For example, it provides not only a mashup editor, but also a HTML editor which allows users to publish mashup results on the Web. Nevertheless, its editing interface is complex, and it eventually requires expertise including HTML, CSS, and JavaScript when users want to publish the mashup results on the Web. Moreover, a mashup of Popfly can be associated to only one type of visualization.

Intel MashMaker [3,2] is a unique mashup editor which is implemented as a Web browser add-on. The basic model of MashMaker is that expert users extract data from Web pages and create mashups using these data, and other users use these mashups. Due to this model, MashMaker requires expertise to create mashups. Moreover, its expressive power is limited since it doesn't support RSS and Web APIs.

There are also numerous number of mashup editors [7,8,6,1,5]. However, they tend to focus on extracting and combining data, and tend to lack in visualizing and publishing mashups.

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed a mashup editor called Ousia Weaver. It adopts a novel mashup programming model which enables users not only to easily create mashups, but also to easily publish mashups. It provides a sophisticated visual editor which enables users to create and publish mashups without writing code, and offers a variety of rich visualization widgets which visualize mashups impressively. With Ousia Weaver, users can easily create and publish mashups as impressive Web pages.

In terms of future work, we plan to do an extensive user evaluation which compares Ousia Weaver to existing mashup editors. We will also continue to improve Ousia Weaver especially for the visual editor, operators, and visualization widgets in order to increase its usability and expressive power.

7. REFERENCES

1 Albinola, Matteo, Baresi, Luciano, Carcano, Matteo, and Guinea, Sam. Mashlight: a Lightweight Mashup Framework for Everyone. In *2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009)* (Madrid, Spain 2009).

2 Ennals, Robert J and Garofalakis, Minos N. MashMaker: mashups for the masses. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (Beijing, China 2007), ACM, 1116-1118.

3 Ennals, Rob and Gay, David. User-friendly functional programming for web mashups. In *ICFP '07: Proceedings of the 12th ACM SIGPLAN international conference on Functional programming* (Freiburg, Germany 2007), ACM, 223-234.

4 Griffin, Eric. *Foundations of Popfly: Rapid Mashup Development (Foundations)*. Apress, 2008.

5 Huynh, David F, Miller, Robert C, and Karger, David R. Potluck: Data mash-up tool for casual users. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6, 4 (Nov. 2008), 274-282.

6 Lin, James, Wong, Jeffrey, Nichols, Jeffrey, Cypher, Allen, and Lau, Tessa A. End-user programming of mashups with vegemite. In *IUI '09: Proceedings of the 13th international conference on Intelligent user interfaces* (Sanibel Island, Florida, USA 2009), ACM, 97-106.

7 Simmen, David E, Altinel, Mehmet, Markl, Volker, Padmanabhan, Sriram, and Singh, Ashutosh. Damia: data mashups for intranet applications. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (Vancouver, Canada 2008), ACM, 1171-1182.

8 Tuchinda, Rattapoom, Szekely, Pedro, and Knoblock, Craig A. Building Mashups by example. In *IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces* (Gran Canaria, Spain 2008), ACM, 139-148.

9 Wong, Jeffrey and Hong, Jason I. Making mashups with marmite: towards end-user programming for the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (San Jose, California, USA 2007), ACM, 1435-1444.

10 YAHOO! INC. *Yahoo! Pipes*. http://pipes.yahoo.com.