Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/092L39

Configurable RESTful Service Mashup: A Process-Data-Widget Approach

Shang-Pin Ma[1,*], *Chun-Ying Huang*[1], *Yong-Yi Fanjiang*[2] *and Jong-Yih Kuo*[3]

[1] Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung 202, Taiwan
[2] Department of Computer Science and Information Engineering, Fu Jen Catholic University, New Taipei City 242, Taiwan
[3] Department of Computer Science and Information Engineering, National Taipei University of Technology, Taipei 106, Taiwan

Received: 7 Aug. 2014, Revised: 8 Nov. 2014, Accepted: 9 Nov. 2014
Published online: 1 Apr. 2015

Abstract: Techniques for the mashup of services have been attracting considerable attention; however, reusable and reconfigurable models for the construction of mashup applications are still lacking. The REST (Representational State Transfer) software architecture has been widely accepted due to its usability and simplicity. This makes REST an appropriate foundation for the development of components for mashup applications. This study proposes a software framework based on the REST architecture, called Process-Data-Widget, to assist designers in building mashup applications. The proposed approach is meant to convert time-consuming, manual mashup-construction tasks into systematic, semi-automatic and configurable tasks. Such an approach would enable designers to design service processes, compose service data, and configure widget-based user interfaces by developing a mashup document. The document could then be input into a mashup engine to produce a corresponding mashup application and a new composite RESTful service.

Keywords: REST, RESTful Service, Service Composition, Widget, Software Framework

1 Introduction

The term mashup [3] refers to a web page or website that combines information and services from multiple sources on the web. Although mashups often lead to innovative composite applications with value added services for users, the notion has yet to gain widespread acceptance due to the difficulties involved in their creation [15], which require familiarity with programming skills (such as HTML, JavaScript, and XML) to link a variety of web APIs and data. Several GUI tools and platforms have recently emerged for the construction of mashups, including Yahoo Pipes [16], IBM Mashup Center [7], and Intel Mashmaker [5]. These tools are meant to simplify programming through the incorporation of graphic models and templates; however, programming tasks cannot be avoided entirely [2].

The mainstream of Web 2.0 website services conforms to REST [6] style (also referred to as RESTful web services.) REST defines a set of architectural principles [6,12] by which one can design web services with a focus on system resources, including how resource states are addressed and transferred using HTTP using a wide range of clients written in different languages. From the viewpoint of software architecture, the REST paradigm represents a light-weight realization of a service-oriented architecture. In contrast, the SOAP/WSDL based Web service architecture is considered a heavy-weight, due to its complex regulations for ensuring security, reliability, and the success of transactions. RESTful services have been widely accepted by the public for the creation of user-centric, non-critical applications due to the usability and simplicity they provide. REST can also be used to provide standard components for mashup construction.

Web service composition is considered an important technology, capable of providing more benefits than any single service [9]. Web Services Business Process Execution Language (WS-BPEL) [15], which is used to interconnect multiple partner services based on process-based model and style, has been accepted as the *de facto* language of service composition both in industry and academia while there is no standards for mashups. Hard-coding mashup applications are difficult to maintain and reuse, even when using mashup tools. As a result, numerous mashup models and methodologies have been

* Corresponding author e-mail: albert@ntou.edu.tw

proposed [14,4,2,10], based on semantic web and rule-based reasoning techniques. However, these efforts have yet to provide a service composition model based explicitly on REST. On the other hand, efforts [3,11] have been made to either extend WS-BPEL to REST-based service composition, or develop new markup languages; however, these methods tend not to account for all of the aspects necessary for a complete mashup application, such as system processes, data manipulation, and user interfaces. In response, this paper proposes a software framework, called Process-Data-Widget, to assist designers in the development of REST-based mashup applications within a lifecycle of mashup development. A mashup engine based on the Process-Data-Widget framework is also devised to allow users to build and configure mashup applications semi-automatically. The proposed approach is meant to convert the construction of mashups from an implementation-level endeavor into a design-level task. Using this approach, designers could design the service process, compose service data, and configure widgets for UI presentation through the development of a mashup document, which would then be input into a mashup engine to produce the corresponding mashup services. The Process-Data-Widget framework involves two kinds of reuse forms: (1) treating mashup RESTful services as a basic service for the construction of larger-scale composite applications; (2) slightly revising mashup documents in accordance with similar requirements.

The remainder of this paper is organized as follows: The Process-Data-Widget framework is outlined in Section 2. The system architecture and two examples are illustrated in Section 3. Related work is presented in Section 4, and conclusions are drawn in Section 5.

2 Proposed Approach: Process-Data-Widget Framework

This section describes the framework of the Process-Data-Widget (PAW). The design concepts of the PAW framework and document specifications are introduced in the following sub-sections.

2.1 Design Concepts

Through the observation of current mashup applications and tools, we identified three fundamental themes crucial to the lifecycle of mashup development, including (1) service process design; (2) data composition; and (3) widget presentation design. A conceptual diagram is presented in Fig. 1.

Each of these components is outlined in the following:

1. Service Process Design: The mashup designer should first design the sequence in which the multiple RESTful services will be executed, and decide how to interconnect input/output (I/O) data among these services. This requires two kinds of documents: (a) the RESTful Service Description Document (SDD), to specify the service capability and I/O data for a RESTful service; and (b) the Mashup Document (MD), to specify the control flow and data assignment among multiple services (in the Service Process Mashup section of MD).
2. Data Composition: Determining how to manipulate service data to produce data sets capable of meeting the needs of users is also critical. During the execution of the mashup service, service-related data (sometimes called service results) are generated after invoking each RESTful service. For example, in a travel mashup application, flight data, hotel data, and weather data are produced during service execution. These raw data are unreadable to end users; therefore, we devised a mechanism called Data Composition, to address this issue. An additional component of the MD is Data Mashup, which is meant to define all required data composition rules. The core concept behind data composition was borrowed from the join operator in the domain of databases. Following data composition, manipulated service data (also called service data sets) can be sorted using Merge Scan and Nested Loop, whereupon any number of sorted/unsorted data sets can be prepared for display in Widgets.
3. Widget Presentation Design: The previous two steps focus mainly on service invocation and data manipulation. However, the resulting data sets should also be easy to read and accessible to the end user (cannot be original XML or JSON data). Thus, determining how to infuse user interaction or user interface techniques into the mashup development process must also be dealt with. In the proposed approach, widgets are used to render data. Another component of the MD, called Widget Mashup, selects

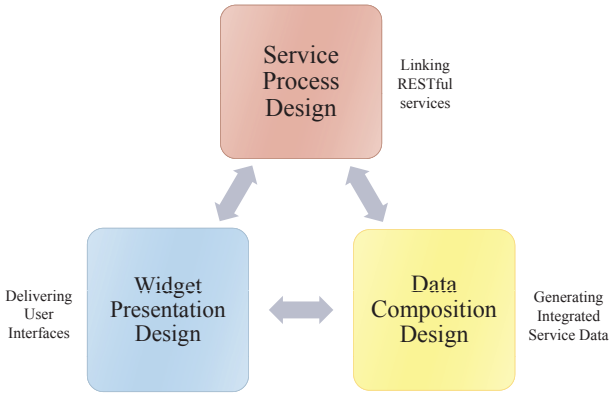


Fig. 1: Conceptual diagram

the data sets to be displayed and determines which widgets are required to render the selected data sets.

2.2 Document Specification Design

Based on the afore-mentioned concepts, we devised the two following specifications: the RESTful Service Description Document (SDD) and the Mashup document (MD). The rationale was to maintain readability for both humans and machine; therefore, Microformat was employed [8] as the foundation with which to design these specifications. Because it is based on HTML, Microformat documents can be processed by programs and viewed by human users.

2.2.1 RESTful Service Description Document (SDD)

Service Description Documents (SDD) describe the information required to invoke a RESTful service, including service URIs, applied HTTP methods, and input parameters. SDD includes three levels: services, service, and resources. The services level includes a range of service information described in the service block. The service level deals with a RESTful service and includes two properties: ID and multiple resources. The resources level specifies the properties of a resource, including resource URI, HTTP methods (GET, POST, PUT and DELETE), input parameters, and output specifications. In general, RESTful services use XML or JSON as the formats for service output. For example, the output data of a flight inquiry service might contain a list of flight information, including flight name, departure location, destination, departure time, and arrival time. The information would be transcribed in XML or JSON. Here we make two assumptions: (1) The input data is parameter-style because most RESTful services take name-value pairs as inputs; (2) The data elements must conform to the domain ontology because only in this way can we solve the issue of data interoperability. If the input/output elements and operations of a service are not named according to the domain ontology, a wrapping process is required to transform original names into ontology-compliant names.

In addition, determining how to search for component services is crucial to the development of a mashup application/service. Achieving search functionality requires the application of information retrieval mechanisms, such as indexing, removing stop words, and TF-IDF (term frequency and inverse document frequency), to retrieve service description documents that are strongly related to the mashup developers request. The developer can select services from a list of retrieved services and design the mashup document with which to build a mashup application/service.

2.2.2 Mashup Document (MD)

As mentioned in Section 2.1, the content of the MD can be divided into three parts: (1) Service Process, (2) Data Composition, and (3) Widget Presentation, corresponding respectively to the ServiceProcessMashup, DataMashup and WidgetMashup sections of the MD.

The service process section includes the service execution sequence and data assignment rules for using or adapting the output of a service in order to invoke another service. We define three common patterns for the service execution sequence: the sequence pattern, parallel pattern and selection pattern, as shown in Fig. 2.

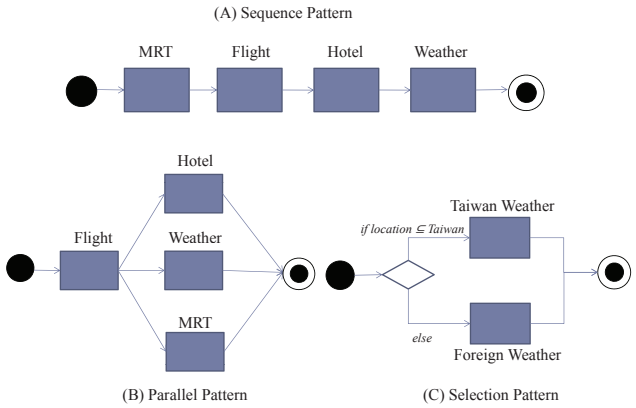


Fig. 2: Service process patterns

To illustrate these three patterns of service process, we consider a travel service, in which four RESTful services are integrated: MRT (Mass Transit Railway), Flight, Hotel, and Weather (Fig. 2). The sequence pattern can combine any number of services executed sequentially. As shown in Fig. 2(A), the four services are invoked one by one. The parallel pattern allows multiple services to be executed simultaneously. As shown in Fig. 2 (B), Hotel, Weather, and MRT services run in parallel. The selection pattern can be used to select one branch from a set of choices. As shown in Fig. 2 (C), whether Taiwan Weather or Foreign Weather is invoked depends on location information.

The ServiceProcessMashup section in MD contains a Process section and multiple ServiceBinding sections including information for binding real RESTful services and for the storage of service data. The Process section defines which services are to be combined as well as which process patterns to be applied. Each ServiceBinding section describes the reference to the RESTful service description, the input parameters, and the name of the service data set produced through the invocation of each participating service. Notably, multiple data sets can be produced because multiple RESTful

services can be invoked during the execution of a Mashup service. In addition, the name of the data set may be referenced in the input element for leveraging a generated data set from one service to invoke another service. An example of the ServiceProcessMashup section in MD is presented in Fig. 3.

```
- <div class="MashupDocument" interaction="true">
  - <div class="ServiceProcessMashup">
    - <div id="Traveling" class="ProcessService">
      - <div class="Sequence">
          <div id="S_Flight" class="ComponentService">Flight Service</div>
        - <div class="Flow">
            <div id="SF_Hotel" class="ComponentService">Hotel Service</div>
          - <div class="Choice">
              <div id="SFC_TaiwanWeather" class="ComponentService"
                Condition="location ==
                'Taipei'|'Taichung'|'kaoshiung'">Taiwan Weather
                Service</div>
              <div id="SFC_ForeignWeather" class="ComponentService"
                Condition="else">ForeignWeather</div>
            </div>
          </div>
        </div>
      </div>
    - <div id="S_Flight" class="ServiceBinding">
        <div class="ServiceID"
          serviceDescriptionName="FlightLocation">FlightLocation</div>
      - <div class="Input">
          <div class="parameter" name="departure"
            interaction="true">Taichung</div>
          <div class="parameter" name="destination"
            interaction="true">Taipei</div>
        </div>
        <div class="OutputDataSetName" datasetType="group">Flight_W-
          DataSet</div>
      </div>
```

Fig. 3: Example of ServiceProcessMashup document

In addition to services execution, the service mashup document is also responsible for the integration of service output data. We rely on two observations to design the data composition mechanism. First, the output data of a RESTful service is usually an XML document or a JSON array that contains a number of data items. (The presence of data items is the reason service output data are also called service data sets). Each data item contains a number of data fields (Figure 4 is an example of a service data set). Second, a SQL join clause can combine records from two or more tables in a database. Based on the two observations, we borrow the "join" concept to combine multiple service data sets that are produced by different RESTful services since a service data set is similar to a database table. A variety types of join, including inner join, left join, right join, and cross join, can be chosen.

Determining the ranking of data items in the joined data set is also required to enhance usability, as users hope to obtain the most useful data in the preceding data items. Here we apply two joining strategies: Nested-Loop (NL) and Merge Scan (MS) (also utilized in [4]). In the proposed approach, NL and MS strategies (Fig. 5) are used as ranking mechanisms and do not affect the invocation sequence of services. If the mashup developer chooses the NL style, a PAW Mashup Engine will begin

```
- <flights>
  - <flight>
      <flightname>CA</flightname>
      <departure>Taipei</departure>
      <destination>Taichung</destination>
      <date>2012-03-28</date>
    - <time>
        <departuretime>12:30</departuretime>
        <arrivaltime>13:30</arrivaltime>
      </time>
    </flight>
  - <flight>
      <flightname>CA</flightname>
      <departure>Taipei</departure>
      <destination>Taichung</destination>
      <date>2012-03-29</date>
    - <time>
        <departuretime>12:30</departuretime>
        <arrivaltime>13:30</arrivaltime>
      </time>
    </flight>
  </flights>
```

Fig. 4: Service data set example

by sorting the data items in the joined data set according to the horizontal data field and re-sort data items according to the vertical data field. If the mashup specifies the MS style, the PAW Mashup Engine will traverse the data items diagonally to ensure that the Top-N data items are not over-emphasized in the vertical data. For example, if we want to manipulate the data set for "hotel", we can set "hotel name" as the vertical data field and set "date" as the horizontal data field. When applying the NL style, the sorted data items may be (ignoring other data fields for simplicity): $\{Hotel_1, Day_1\}$, $\{Hotel_1, Day_2\}$, $\{Hotel_1, Day_3\}$, $\{Hotel_2, Day_1\}$, $\{Hotel_2, Day_2\}$, $\{Hotel_2, Day_3\}$, $\{Hotel_3, Day_1\}$, $\{Hotel_3, Day_2\}$, and so forth. When applying the MS style, the sorted data items may be $\{Hotel_1, Day_1\}$, $\{Hotel_1, Day_2\}$, $\{Hotel_2, Day_1\}$, $\{Hotel_3, Day_1\}$, $\{Hotel_2, Day_2\}$, $\{Hotel_1, Day_3\}$, $\{Hotel_1, Day_4\}$, $\{Hotel_2, Day_3\}$, and so forth. Using the MS style, the user can obtain information related to more hotels from the preceding data items. Sorting strategies may be selected according to the situation.

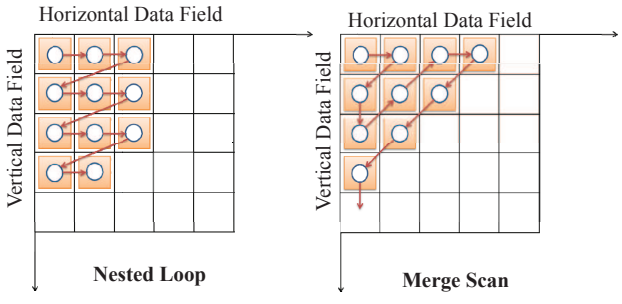


Fig. 5: Nested loop and merge scan styles

Figure 6 is an example of a partial MD illustrating a data mashup. The settings related to involved data sets, join keys, ranking style, and name of the output data set are designed in the MD.

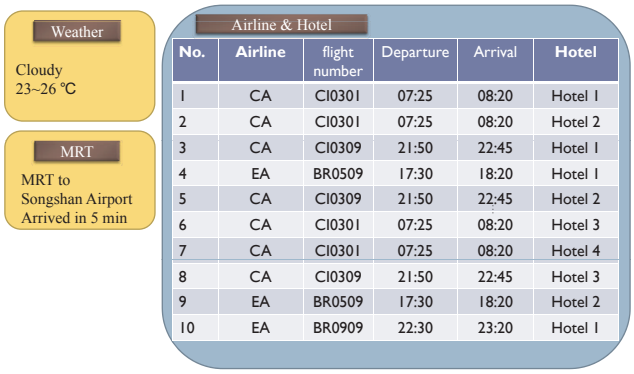
```
- <div class="DataMashup">
  - <div id="D-Flight_Weather" class="Data">
    - <div class="InnerJoin">
      - <div class="DataSetToJoin">
          <div class="PrimaryDataSet">Flight-DataSet</div>
          <div class="SecondaryDataSet">Weather-DataSet</div>
        </div>
      - <div class="Joinkeys">
          <div class="Key">Flight-DataSet.destination=Weather-
            DataSet.location</div>
          <div class="Key">Flight-DataSet.date=Weather-DataSet.date</div>
        </div>
        <div class="OutputJoinedDataSetName">Flight_Weather-
          JoinDataSet</div>
      </div>
    - <div class="Ranking">
        <div class="JoinDataSetName">Flight_Weather-JoinDataSet</div>
        <div class="SortingStyle">MS</div>
      - <div class="SortingKey">
          <div class="Vertical">date</div>
          <div class="Horizontal">weather</div>
        </div>
        <div class="OutputSortedDataSetName">Flight_Weather-
          SortedDataSet</div>
      </div>
    </div>
```

Fig. 6: Example of DataMashup document

The final step in designing a mashup application involves designing the widgets. Each widget can be treated as a container to be filled with sorted data. An example of a widget is presented in Fig. 7. Thus, two settings must be assigned: the data set to display and the style of the widget. In the proposed method, the data meant for display can be reduced by selecting appropriate data fields (and ignoring the others) or selecting only the preceding N data items. The proposed approach supports a number of widget styles, including Simple Panel, Simple Table, Menu Bar, Collapsible Panels, and Tabbed Panels. The PAW mashup engine can also provide two external interfaces: one for end users via widgets and the other for client programs via a RESTful service interface. Thus, we devised a "ServiceURL" property to allow designers to assign the preferred URL pattern for further access. Figure 8 presents an example of a partial MD used to generate the widget-based user interface for browsing sorted data sets.

3 Implementation

The system architecture is presented in Fig. 9. As mentioned previously, the PAW mashup engine is responsible for parsing the MD and generating a corresponding mashup application (and mashup RESTful service). Thus, the PAW mashup engine plays the role of a central bus capable of linking all participating RESTful



No.	Airline	flight number	Departure	Arrival	Hotel
1	CA	CI0301	07:25	08:20	Hotel 1
2	CA	CI0301	07:25	08:20	Hotel 2
3	CA	CI0309	21:50	22:45	Hotel 1
4	EA	BR0509	17:30	18:20	Hotel 1
5	CA	CI0309	21:50	22:45	Hotel 2
6	CA	CI0301	07:25	08:20	Hotel 3
7	CA	CI0301	07:25	08:20	Hotel 4
8	CA	CI0309	21:50	22:45	Hotel 3
9	EA	BR0509	17:30	18:20	Hotel 2
10	EA	BR0909	22:30	23:20	Hotel 1

Fig. 7: Example of Widget view

```
- <div class="WidgetMashup">
  - <div id="WC-Travel" class="WidgetContainer" type="TabbedPanel">
    - <div id="W-Flight_Hotel" name="Flight-Hotel Data Table" class="Widget"
        type="PagedTable" itemsPerPage="10">
        <div class="DataSetName">Flight_Hotel-RankingDataSet</div>
      - <div class="FieldSelector">
          <div class="Filter" selectable="true">flightname</div>
          <div class="Filter">departure</div>
          <div class="Filter">destination</div>
          <div class="Filter">date</div>
          <div class="Filter">departuretime</div>
          <div class="Filter">arrivetime</div>
          <div class="Filter">hotelname</div>
          <div class="Filter">address</div>
          <div class="Filter">phone</div>
        </div>
        <div class="ServiceURL">/Flight_Hotel</div>
      </div>
```

Fig. 8: Example of WidgetMashup document

services according to the MD and providing two external interfaces: widgets for embedding HTML and RESTful services to supply service data in XML, for end users and client programs, respectively.

To demonstrate the feasibility of the proposed approach, we developed two mashup applications. The first application was a travel service combining flight, hotel, and weather services, the user interface of which is presented in Fig. 10. The second application was a news retrieval service connecting the keyword extraction service to the news search service to allow users to browse a news webpage and then click on one of extracted keywords to browse other related pages. The final user interface for the news service is presented in Fig. 11. Both of these applications were generated by the PAW mashup engine, such that the mashup designer merely developed SDDs and MDs without the need to write any programs. The control of service processes, the combination of data, and generation of the GUI were performed automatically by the PAW mashup engine.

	flightname	departure	destination	date	departuretime	arrivetime	hotelname	address	phone	weather
no. 1	CA	Taichung	Taipei	03-28	12:30	13:30	EvergreenLaurel	No. 63, Songjiang Rd., Taipei City 10455, Taiwan (R.O.C.)	886-2-2501-9988	Rain, 19°C~28°C
no. 2	EA	Taichung	Taipei	03-28	12:30	13:30	EvergreenLaurel	No. 63, Songjiang Rd., Taipei City 10455, Taiwan (R.O.C.)	886-2-2501-9988	Rain, 19°C~28°C
no. 3	TA	Taichung	Taipei	03-28	12:30	13:30	EvergreenLaurel	No. 63, Songjiang Rd., Taipei City 10455, Taiwan (R.O.C.)	886-2-2501-9988	Rain, 19°C~28°C
no. 4	MA	Taichung	Taipei	03-28	12:30	13:30	EvergreenLaurel	No. 63, Songjiang Rd., Taipei City 10455, Taiwan (R.O.C.)	886-2-2501-9988	Rain, 19°C~28°C
no. 5	UA	Taichung	Taipei	03-28	12:30	13:30	EvergreenLaurel	No. 63, Songjiang Rd., Taipei City 10455, Taiwan (R.O.C.)	886-2-2501-9988	Rain, 19°C~28°C
no. 6	CA	Taichung	Taipei	03-28	12:30	13:30	Howard	No. 160, Ren Ai Rd. Sec.3, Taipei, Taiwan (R.O.C.)	886-2-2700-2323	Rain, 19°C~28°C
no. 7	EA	Taichung	Taipei	03-28	12:30	13:30	Howard	No. 160, Ren Ai Rd. Sec.3, Taipei, Taiwan (R.O.C.)	886-2-2700-2323	Rain, 19°C~28°C
no. 8	TA	Taichung	Taipei	03-28	12:30	13:30	Howard	No. 160, Ren Ai Rd. Sec.3, Taipei, Taiwan (R.O.C.)	886-2-2700-2323	Rain, 19°C~28°C
no. 9	MA	Taichung	Taipei	03-28	12:30	13:30	Howard	No. 160, Ren Ai Rd. Sec.3, Taipei, Taiwan (R.O.C.)	886-2-2700-2323	Rain, 19°C~28°C
no. 10	UA	Taichung	Taipei	03-28	12:30	13:30	Howard	No. 160, Ren Ai Rd. Sec.3, Taipei, Taiwan (R.O.C.)	886-2-2700-2323	Rain, 19°C~28°C

Fig. 10: User interface for travel service

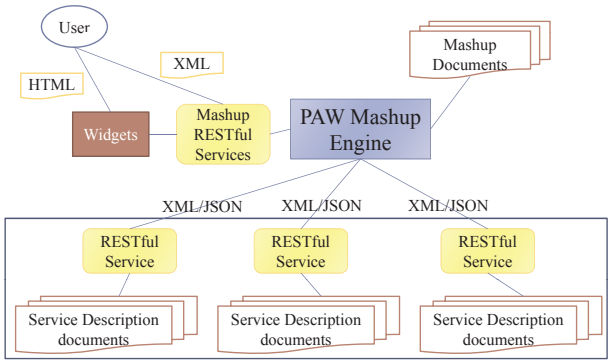


Fig. 9: Overall architecture

	Title	PubDate	Abstract	Link
no. 1	(ISC)2公佈第五屆亞太區資訊安全領導成就表彰計劃(ISLA)獲獎者	Thu, 07 Jul 2011 00:15:42 -0700	...三周年 香港2011年7月7日電 /美通社亞洲/ -- 全球最大的資訊安全專業組織及CISSP(R)首席者(ISC)²(R)(「ISC-squared...成果，獲提名的專業人員可根據以下三項類別作出分類：資訊安全計劃(一項或多項)管理級專業人員，高級資訊保安專業...	http://tw.news.yahoo.com/article/url/d/a/110707/54/2un0s.html
no. 2	種美機密駭客找承包商下手	Thu, 21 Jul 2011 23:53:57 -0700	（路透倫士頓21日電）根據兩家資訊安全公司，一群可能為外國政府工作的駭客，正積極透過侵入為美國...承包商，但他們表示，電子郵件內程式碼精準經過分析，發現與其他安全專家過去10天提供的惡意軟體有關，分析惡意郵件的兩家資安公司...	http://tw.news.yahoo.com/article/url/d/a/110722/16/2vj6z.html
no. 3	RIM的PlayBook獲美聯邦政府認證	Fri, 22 Jul 2011 05:53:52 -0700	...成為首款獲得這項許可的平板電腦。美國在911恐怖攻擊後不久，隨即通過聯邦資訊安全管理法案，規定聯邦政府使用的所有電腦工具，都必須符合聯邦認證標準，RIM的...	http://tw.news.yahoo.com/article/url/d/a/110722/135/2vjvw.html
no. 4	演光電腦兵推考量增 加雷防駭	Sun, 17 Jul 2011 13:21:35 -0700	...都會在今天展開的電腦兵推中，列為重點檢查項目，官員週週露，為了強化官兵對資訊安全的警覺性，國防部平時就會以「釣魚郵件」來測試官兵，國防部演光二十七號演習...	http://tw.news.yahoo.com/article/url/d/a/110718/78/2v8ts.html

Fig. 11: User interface for news retrieval service

4 Related Work

In the following, we introduce a number of important studies related to RESTful service delivery and the construction of Mashup applications.

Sheth et al. [14] proposed a method of connecting services by SA-REST to enable services to obtain necessary data from other services. This approach makes use of semantic annotation and XSLT data adaptation to achieve service mashup. Kopecky et al. [8] defined a model of RESTful Web services to create the hRESTS microformat, a machine-readable document format for existing Web APIs. The proposed hRESTS microformat describes the main aspects of services, such as operations, inputs, and outputs. Battle and Benson [1] proposed a framework to align the publication of semantic data with existing web architectures, based on their distributed query architecture called Semantic Query Decomposer (SQD) and a java tool called Semantic Bridge for web services. This framework combines REST-based design and RDF data access, and supports the search of internal services through SPARQL (SPARQL Protocol and RDF Query Language) and external services for external users. Braga et al. [4] proposed a service mashup language for the graphical composition and automatic execution of queries over online data-sharing services. This approach includes a variety of techniques, such as parallelism and pipelining methods to control the execution sequence as well as merge-scan and nested-loop methods to join different service data, and a chunking mechanism to return the results in chunks. Zhang et al. [17] proposed a two-layer mashup service model for a university website integrating weather forecast sites, news sites, and course-related information. To improve efficiency and usability, this method can query and update data based on the caching mechanism and deliver information through the social network. Peng et al. [13] presented a framework called REST2SOAP to wrap SOAP services into RESTful services semi-automatically. REST2SOAP allows developers to create a mashup service by developing a BPEL document. REST2SOAP is also a heavy-weight solution utilizing many WS-* specifications.

Most of above studies utilized REST as a foundation from which to establish methodologies for the construction of mashups and service delivery. However, these methods do not provide a compositional model based explicitly on

REST to connect services, manipulate service data, and supply a user-oriented view, as we have done in this paper.

5 Conclusions and Future Work

This paper proposes a software framework called Process-Data-Widget (PAW), to function as a composition model for the construction of mashup applications. This framework enables developers to design the service process, compose service data, and configure widgets for presentation on a UI simply by constructing a mashup document (MD). The PAW mashup engine also parses the MD and generates a corresponding mashup application and associated mashup RESTful services.

In the future, we will continue the development of a GUI-based designer tool working atop the PAW framework to further ease the development of mashup services. Our second goal is the integration of data mining technology with the power of social networks to recommend appropriate services for the design of mashup applications.

Acknowledgement

This research was sponsored by National Science Council in Taiwan under the grant 100-2221-E-019-037.

References

[1] Robert Battle and Edward Benson. Bridging the semantic web and Web 2.0 with representational state transfer (REST). *Web Semantics*, 6(1):61–69, February 2008.

[2] Athman Bouguettaya, Surya Nepal, Wanita Sherchan, Xuan Zhou, Jemma Wu, Shiping Chen, Dongxi Liu, Lily Li, Hongbing Wang, and Xumin Liu. End-to-end service support for mashups. *IEEE Transactions on Services Computing*, 3(3):250–263, July 2010.

[3] Alessandro Bozzon, Marco Brambilla, Federico Michele Facca, and Giovanni Toffetti Carughu. A conceptual modeling approach to business service mashup development. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 751–758, Washington, DC, USA, 2009. IEEE Computer Society.

[4] Daniele Braga, Stefano Ceri, Florian Daniel, and Davide Martinenghi. Mashing up search services. *IEEE Internet Computing*, 12(5):16–23, September 2008.

[5] Robert Ennals. Intel mash maker: Mashups for the masses, March 2012. Available at http://software.intel.com/en-us/articles/intel-mash-maker-mashups-for-the-masses.

[6] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, May 2002.

[7] IBM. Ibm mashup center. Availbale at http://www.ibm.com/developerworks/lotus/products/mashups/.

[8] Jacek Kopecký, Karthik Gomadam, and Tomas Vitvar. hRESTS: An HTML microformat for describing RESTful web services. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '08, pages 619–625, Washington, DC, USA, 2008. IEEE Computer Society.

[9] Jonathan Lee, Shang-Pin Ma, Ying-Yan Lin, Shin-Jie Lee, and Yao-Chiang Wang. Dynamic service composition: a discovery-based approach. *International Journal of Software Engineering and Knowledge Engineering*, 18(2):199–222, 2008.

[10] Anne H. H. Ngu, Michael P. Carlson, Quan Z. Sheng, and Hye-young Paik. Semantic-based mashup of composite applications. *IEEE Transactions on Services Computing*, 3(1):2–15, January 2010.

[11] Cesare Pautasso. RESTful web service composition with BPEL for REST. *Data and Knowledge Engineering*, 68(9):851–866, September 2009.

[12] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. RESTful web services vs. "big"' web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM.

[13] Yu-Yen Peng, Shang-Pin Ma, and J. Lee. REST2SOAP: A framework to integrate SOAP services and RESTful services. In *Service-Oriented Computing and Applications (SOCA), 2009 IEEE International Conference on*, pages 1–4, 2009.

[14] Amit P. Sheth, Karthik Gomadam, and Jon Lathem. SA-REST: Semantically interoperable and easier-to-use services and mashups. *IEEE Internet Computing*, 11(6):91–94, November 2007.

[15] OASIS Web Services Business Process Execution Language (WS-BPEL) TC. Web services business process execution language (WS-BPEL) version 2.0, 11 April 2007 2007.

[16] Yahoo. Yahoo pipes. Available at http://pipes.yahoo.com/pipes/.

[17] Jia Zhang, Momtazul Karim, Karthik Akula, and Raghu Kumar Reddy Ariga. Design and development of a university-oriented personalizable web 2.0 mashup portal. In *Proceedings of the 2008 IEEE International Conference on Web Services*, ICWS '08, pages 417–424, Washington, DC, USA, 2008. IEEE Computer Society.

Shang-Pin Ma received his Ph.D. and B.S. degrees in Computer Science and Information Engineering from National Central University in 2007 and 1999, respectively. Now he is an associate professor of Computer Science and Engineering Department at National Taiwan Ocean University in Taiwan. His current research interests include web-based software engineering, service-oriented computing, and semantic web.

Chun-Ying Huang received the B.S. degree in Computer Science from National Taiwan Ocean University in 2000 and the M.S. degree in Computer Information Science from National Chiao-Tung University in 2002. He received the Ph.D. degree in Electrical Engineering from National Taiwan University in 2007. He is now an associate professor in Computer Science and Engineering Department at National Taiwan Ocean University. His current research interests focus on various aspects of computer networks and network security, including key management, attack mitigation, intrusion detection, and traffic analysis. Dr. Huang is a member of IEEE and ACM.

Yong-Yi FanJiang received the BS degree in Computer Science and Information Engineering from Tamkang University, Taiwan, in 1994. He received his MS and PhD degree in Computer Science and Information Engineering from National Chiao Tung University, Taiwan, in 1998 and from National Central University, Taiwan, in 2004, respectively. Currently, he is an assistant professor in the Department of Computer Science and Information Engineering, Fu Jen Catholic University, Taiwan, from 2007. His research interests include mobile and pervasive computing, software engineering, semantic web, and human computer interaction.

Jong Yih Kuo received his BS degree from National Tsing Hua University, Taiwan, Republic of China, in 1991, and his PhD degree from the National Central University, Taiwan, in 1998. He is now an associate professor in the Intelligent System Laboratory of the Department of Computer Science and Information Engineering at the National Taipei University of Technology in Taiwan. His research interests include agent-based software engineering and fuzzy logic.