

# A Framework for Composing Personalized Web Resources

Bipin Upadhyaya<sup>1</sup>, Hua Xiao<sup>2</sup>, Ying Zou<sup>1</sup>, Joanna Ng<sup>3</sup>,  
Alex Lau<sup>3</sup>

<sup>1</sup> Dept. Of Electrical and Computer Engineering, Queen's University  
Kingston, Ontario, Canada  
{9bu, ying.zou}@queensu.ca

<sup>2</sup> School of Computing, Queen's University  
Kingston, Ontario, Canada  
huaxiao@cs.queensu.ca

<sup>3</sup> IBM Canada Laboratory, Markham, Ontario, Canada  
{jwng, alexlau}@ca.ibm.com

**Abstract.** There are a large number of Web resources available on the Internet. However, only small subsets of Web resources are used to fulfill a user's needs. Due to the heterogeneity and decentralization of Web resources, it is a time-consuming and tedious process for users to sift through the sheer volume of Web resources to fulfill their needs. The previous accessed Web resources are not automatically tracked. Therefore, if the activities (*e.g.*, planning a trip) need to be repeated, a user has to perform the same process over and over again. To support users' recurring online activities, we propose a framework for creating a personalized Web space which helps a user to manage and orchestrate services for performing recurring online activities. Our framework provides techniques for a user to: 1) discover available Web resources distributed in different websites despite the format of Web resources; 2) provide mechanisms to allow users to share their information and resources across different websites and services; and 3) compose Web resources distributed in different websites to fulfill their activities. As a proof of concept, we designed and developed a prototype to demonstrate the use of our proposed framework for creating a personalized Web space.

**Keywords:** Web resource composition, Web resource identification, RESTful service; personalized Web space

## 1 Introduction

Internet has become one of the major sources for people to obtain information and perform tasks (*e.g.*, online shopping) for their daily activities. Various types of Web resources, such as Web pages, Simple Object Access Protocol (SOAP)-based Web Services and Representational State Transfer (REST) services [17], exist on the Internet to provide various functionalities. Although there are a large amount of Web resources available on the Internet, a very small subset of Web resources is used to fulfill the needs of users' daily activities, such as online shopping and planning a trip. It is a time-consuming and tedious process for users to sift through the sheer volume

of Web resources to perform their daily activities due to the following limitations of existing technologies to access Web resources.

1) Heterogeneous Web resources hinder search engines and users to discover suitable Web resources for fulfilling users' goal of daily activities. Existing Web resources are described in heterogeneous formats. For example, Web Service Description Language (WSDL) [7] is used to describe SOAP-based Web services that make remote procedure calls. WSDL is designed for programmers. It is difficult to be understood by non-IT professional users. HTTP-based APIs are increasingly used by companies such as Twitter [36] and Flickr [18]. They are chosen over SOAP-based services. However, HTTP-based APIs reveal little information about the functionality of the APIs. It is challenging for existing search engines such as Google [21] to discover the HTTP-based APIs due to the lack of functional descriptions.

2) Replicated information and accounts are across different online services and websites. More and more people use social networking services, such as Facebook [16], MySpace [29] and Twitter [36] to communicate. However, the messages and friends in one social networking service are not easily shared with another social networking service. For example, messages posted on Facebook cannot be viewed by the friends on MySpace although a user may have the same friends in both services. To share a message and communicate with friends in each service, users have to log into each service to post the replicated information across multiple online services. It requires effective tools or platforms help users manage the accounts and replicated information across the boundaries of services.

3) Web resources for fulfilling a user's activity are often distributed in several websites. In today's online experience, users frequently re-visit Web resources distributed across different websites to perform repeated tasks. For example, a person planning a conference trip needs to locate various Web resources to search for transportation, reserve accommodation, and look for local attractions. In current practices, users have to visit multiple websites to find the desired Web resources. The visited Web resources are not recorded. Therefore, users cannot automatically reuse the already performed process for a recurring activity. It is a time-consuming process to manually compose Web resources. The result of service composition may not provide the optimal outcome. For instance, a user may not be able to discover a Web service that provides the most economical air ticket.

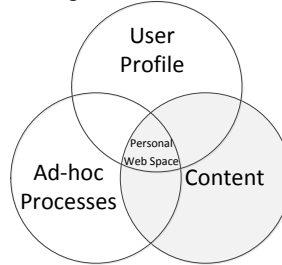
To support the user's social, recreational, professional, and other activities, it is essential to create a personalized Web space that provides the following functionalities:

- 1) Discover available Web resources from websites to fulfill a user's goal (*i.e.*, activities), despite the formats of Web resources. For example, when a user needs to buy a flight ticket, we can return appropriate Web resources related to the flight ticket purchasing. The Web resources could be a SOAP-based service, a HTTP-based API, a website or even a message with a coupon code for buying flight tickets.
- 2) Provide mechanisms to allow users to share their information and Web resources over different websites and online services. For example, a user only needs to post a message one time and the personalized Web space can automatically share the message across multiple social networking services that a user registered.

- 3) Help users compose Web resources distributed over different websites to fulfill their activities. For instance, when a user plans a trip, we can automatically provide and organize the related Web resources from heterogeneous websites to the user and help her/him plan the trip.

To achieve these functionalities, we propose a framework for building a personalized Web space that assists a user in managing various Web resources and composing Web resources for automatic reuse. We create a unified description schema to describe heterogeneous Web resources. A unified description schema can facilitate our framework to discover functionally similar Web resources regardless of the formats. We provide an approach to describe relations among different Web resources forming a resource graph. The relations specified in such a resource graph enable users to manage various Web resources easily. Moreover, we provide techniques to compose coarse granular Web resources using a resource graph. The composite Web resources are represented as an ad-hoc processes which capture user's dynamic needs. Similar to traditional business processes, an ad-hoc process captures a set of activities (*i.e.*, work items) and the control and data flows among the activities. Different from the business processes, work items within an ad-hoc process do not follow strict order. For example, "planning a night out" can be described as an ad-hoc process. It involves several work items, such as going for a movie, and reserving a table in a restaurant in any orders based on a user.

The remainder of this paper is presented as follows. Section 2 gives an overview of the proposed personalized Web space. Section 3 describes the schema for representing various Web resources in a unified format. Section 4 proposes a resource graph for modeling the relations among Web resources. Section 5 presents the technique for constructing ad-hoc processes. Section 6 discusses our prototype for demonstrating the proposed framework. Section 7 discusses the related work. Finally Section 8 concludes the paper and explores the future work.

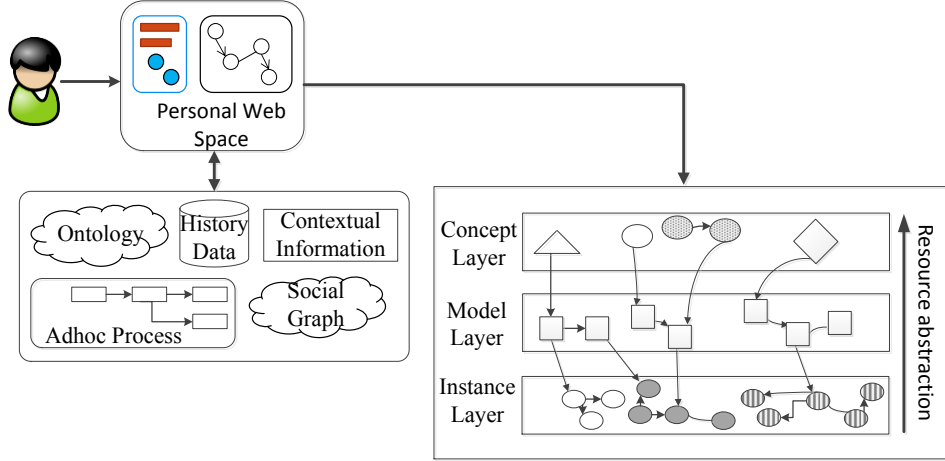


**Figure 1:** Overview of a personalized Web space

## 2. The Proposed Personalized Web Space

We envision that a personalized Web space is an interaction of three major components: user profiles, content and ad-hoc processes as shown in Figure 1. A user profile contains data related to a user including profession, interests, online behaviors, frequently visited Web resources and his/her social peers. Content refers to the information delivered by Web resources, such as subscribed RSS feed, bookmarks and Web services. Ad-hoc processes denote a set of recurring user's activities. An ad-

hoc process involves activities that are frequently performed in the past and might be interested to a user in the future. More specifically, each activity can be fulfilled by one or more Web resources.

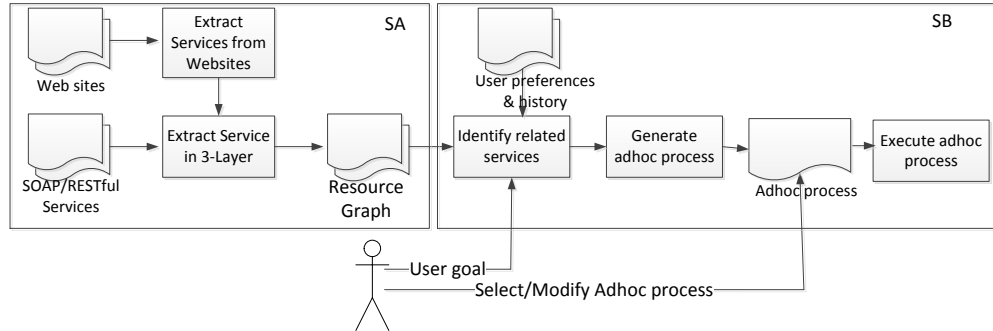


**Figure 2:** Architecture of a Personal Web Space

In Figure 2, we present three layer architecture of the proposed personalized Web space including instance layer, model layer and concept layer. The architecture treats Web resources as a key element to manage the data involved in a personalized Web space. The instance layer represents the Web resources distributed over the internet. Web resources with similar functionality can be implemented by various types of services (e.g., RESTful services or SOAP-based services) in different websites. The model layer represents the Web resources in our proposed unified schema. The concept layer contains the semantic concepts that are used to describe the functionality of Web resources available in the instance layer.

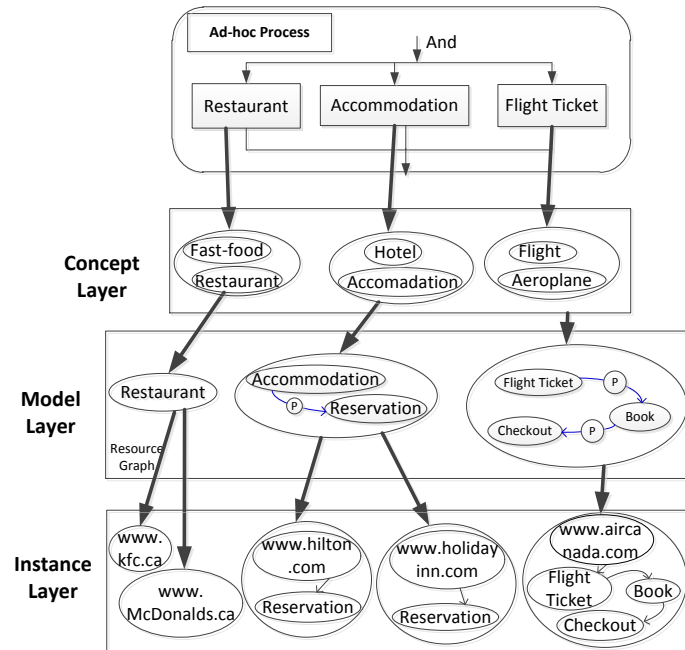
The semantic concepts of the Web resources are annotated by the users in the personal Web space. Instance layer represents the physical Web resources hosted by different service providers. Based on their usages by the users, the semantic relations can be established among Web resources. For example a “restaurant” resource and a “review” resource in different Websites can have the “hasReview” relationship. We identify relations among different Web resources and construct a resource graph in the model layer. Resource Description Framework (RDF) [5] is designed specifically for exchanging and integrating Web data. In the proposed architecture, we migrate various Web resources into RESTful services, and then adopt RDF to describe RESTful services and their relations. In concept layer, we allow a user to annotate semantic concepts to the Web resources using keywords. Concept layer provides a high level overview of the functionality of Web resources. For example a flight booking service has concepts: flight and city. The concepts are then linked to resources in the model layer and instance layer. Web resource providers can specify the categories of their Web resources using the concepts defined in the concept layer. Concept layer defines relations among different Web resources based on concept shared. We use open database such as WordNet [32] and ConceptNet [30] to correlate

concepts in the concept layer. Our previous work [6] provides more detail steps on extracting concepts from service description files.



**Figure 3:** Overall steps to generate an ad-hoc process

Figure 3 shows the overall steps to generate an ad-hoc process. The framework takes the various Web resources (such as forms defined Websites, SOAP based services and RESTful services) as inputs. We extract Web resources and abstract them in three layer architecture. The resource graph is generated based on the three - layer architecture. Given a user goal, our framework identifies Web resources required to fulfill the user's goal based on his/her preferences and historical data. A user can modify the generated ad-hoc process and then executes the process.



**Figure 4:** An example scenario of an application of the architecture for the personalized Web space

When accomplishing an online activity, a user can simply describe the desired goal using keywords. We map a goal into Web resources described by a resource graph and infer an ad-hoc process to help a user fulfill the goal. Based on the data flow between Web resources, we identify the control flows between Web resources. Figure 4 shows an example scenario of using our proposed framework. A user provides a keyword “travel” to describe his/her goal. Our framework generates an ad-hoc process which maps the activities into the Web resources in the model layer. The concrete Web resources, such as Air Canada web site (*i.e.*, [www.aircanada.com](http://www.aircanada.com)) to book flight tickets, are represented in the instance layer. Using the mapping between the model layer and the instance layer, we can find the concrete Web resources to fulfill different work items specified in the ad-hoc process. Meanwhile, a user can interact with the personalized Web space to connect different Web resources and customize the ad-hoc process.

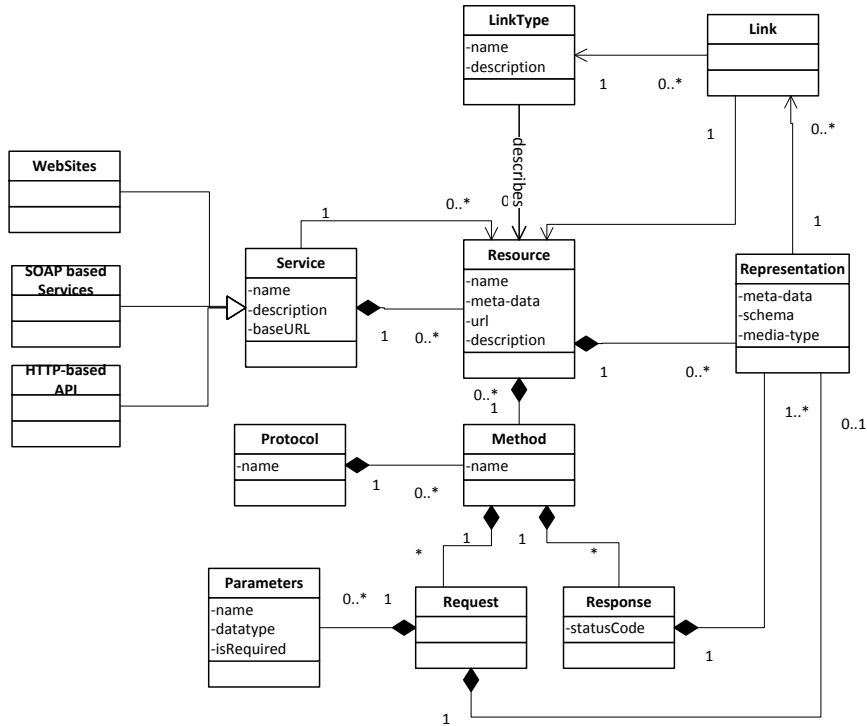


Figure 5: The unified description schema

### 3 Modeling Heterogenous Web Resources in a Unified Format

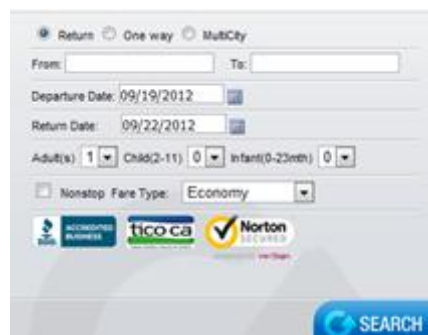
We devise a unified resource schema that represents heterogeneous Web resources (*e.g.*, Web Forms, SOAP-based services, and HTTP-based APIs) in a single format as shown in Figure 5. A service aggregates different Web resources identified using a

Universal Resource Identifier (URI). A Web resource represents an entity from the real world. Its state can be exposed and changed via accessing the URI. We represent all the Web resources in REST style.

```
<xs:element name="Restaurant">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xsd:string"/>
      <xs:element name="establishedDate" type="xsd:date"/>
      <xs:element name="lat" type="xsd:Integer"/>
      <xs:element name="lon" type="xsd:Integer"/>
      <xs:element name="address" type="xsd:String"/>
      <xs:element name="phone" type="xsd:String"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

**Figure 6:** Excerpt of resource schema of a restaurant resource

Web resources are accessed using methods defined in a protocol. In the HTTP protocol, a method can be GET, POST, PUT, and DELETE. A GET method notifies the service provider to retrieve the data. A PUT method updates the data with the ones send in the request. A DELETE method deletes the data of a service. A POST method creates a new resource. A method has at least one request and several optional response messages. A request includes a set of parameters to be sent to the Web resource. A response message is linked with a representation that returns the state of the Web resource after a method invocation. The response uses different status code to represent faulty responses (*e.g.*, incorrect parameters, and server errors). The protocol of the method specifies the different types of status code that defines how the response should be interpreted. For example, HTTP status code 200 represents that the request has succeeded. The information returned in the response is dependent from the method used in the request. For example, when a GET method is invoked, an entity corresponding to the requested resource is sent in the response. Figure 6 shows the resource schema for the representation of a restaurant.



**Figure 7:** A Web form for flight search

```

<Service>
<Name> Flight Search </Name>
<Description>Search and bok the flight</Description>
<Provider>Flight Network </Provider>
<BaseURL>http://http://www.flightnetwork.com</BaseURL>
<Resource>
  <Name> Flight Search </Name>
  <Meta-data>
    <name= "Search"/>
    <name= "Flight"/>
    <name= "tickets"/>
  <Meta-data>
    <URL> http://www.flightnetwork.com/flights/search</URL>
    <Protocol type="HTTP" ref="http://tools.ietf.org/html/rfc2626">
    <Method name="POST">
    <Request>
      <Parameter>
        <name="Category" datatype="String" />
        <name="From" datatype="String" />
        <name="To" datatype="String" />
        <name="From" datatype="String" />
        <name="Departure Date" datatype="Date" />
        <name="Return Date" datatype="Date" />
        <name="Adult(s)" datatype="Integer" />
        <name="Child(2-11)" datatype="Integer" />
        <name="infant(0-23mth)" datatype="Integer" />
        <name="Type" datatype="String" />
      </Parameter>
    </Request>
    <Response>
      <media-type> txt/html </media-type>
    </Response>
  </Method>
</Protocol>
</Resource>
</Service>

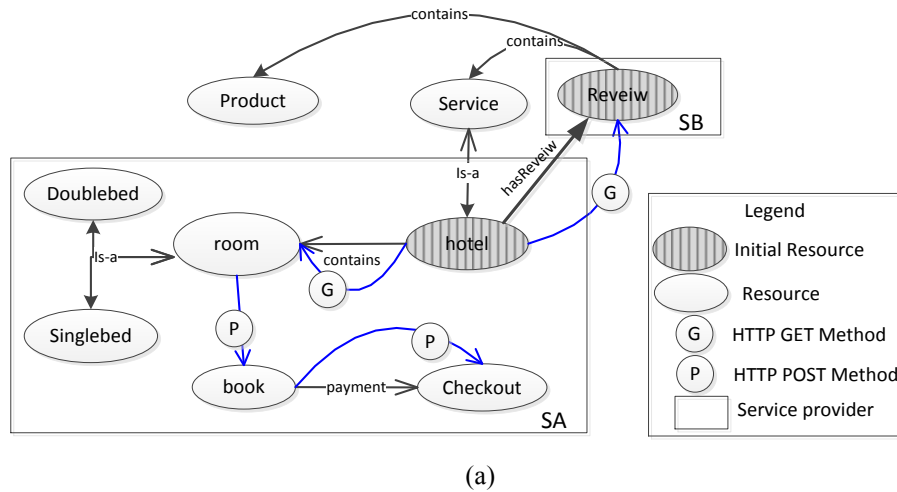
```

**Figure 8:** A unified schema to represent a Web form illustrated in Figure 7

We develop techniques to migrate non-RESTful services to RESTful services in order to represent such Web resources in the proposed unified format. For example in our earlier work [37], we provide an approach to migrate SOAP-based services to RESTful services. We identify Web resources from a SOAP-based Web service by analyzing its WSDL and map the contained operations to Web resources and HTTP methods. When a user uses such migrated Web resources, the corresponding WSDL operations are invoked. HTTP-based APIs are RESTful services. The functionality provided by Web sites can be abstracted as a RESTful service. For example, a Web form to search tickets illustrated in Figure 7 can be used as a ticket searching service; the input parameters are described in the input fields in a Web form (such as input text, text area and radio buttons) and the output parameters is the result returned in the html format. Figure 8 shows the unified schema extracted from a flight search functionality of a Web Form shown in Figure 7.



Figure 8 shows an example of a Web form represented in the unified schema. The Web resource shown in Figure 7 is provided by “Flight Network” and the base URL (<http://www.flightnetwork.com/>) of the service provider. The resource name is “Flight Search” with meta-data “Search”, “Flight” and “Tickets”. The resource uses POST method. The resource has request parameters including “Category”, “From”, “To”, “Departure Date”, “Return Date”, “Adult (s)”, “Child (2-11)”, “infant (0-23mth)” and “Type”. POST method is used by the form. The response has the media-type “txt/html”. Converting different types of Web resources to REST style provides a simplified view for users to use and manipulate Web resources. The Web resources are described in RDF to simplify the manipulation of Web resources specified in different languages, such as query languages (*e.g.*, SPARQL [35]), transformation languages (*e.g.*, Gleaning Resource Descriptions from Dialects of Languages [19]), and rule languages (*e.g.*, Rule Interchange Format [30]).



```
GET /hotel

Response

<hotel>
<name>Holiday Inn</name>
<establishedDate> 2010-01-20</establishedDate>
<lat>35.159401</lat>
<lon>-84.876701</lon>
<address>2 Princess Street Kingston, ON K7L 1a2, Canada </address>
<phone>(613) 549-3508 </phone>
<link rel="hasReview" href="http://../review/0439023483"/>
</hotel>
<link rel="next" href="/book"/>
```

(b)

**Figure 9:** An example of Web resources and their relations

## 4 Representing a Resource Graph

We create a resource graph to represent Web resources and the relations between Web resources. We consider the resource graph as a semantic network model which consists of entities and relationships. Entities in a resource graph denote Web resources identifiable using URIs. A Web resource may be linked to other Web resources by a set of relations. We have identified three types of relations:

**Data flow based relations:** Data flow relations define the flow of data between two or more resources. The data flow relation is determined by matching the schema between the input and output of methods in Web resources. We use link specification [31] to describe the data flow based relation between Web resources in a resource graph. A “rel” attribute defined in the link specification gives the information about the semantics of the link. Figure 8 b. shows two links with “rel”. We identified the following four kinds of data flow relations

- i. **See-also** recommends other Web resources. For example a book resource, b1, has *see-also* relation with book resources, b2 and b3, if all the three books are written by the same author a1.
- ii. **Same-as** provides the similar Web resources. For example, if flight resources, f1 and f2, fly between two cities at the same time, these Web resources are related by *same-as* relation.
- iii. **Is-a** defines is-a relation between the Web resources. For example, a single bedroom, and a double bedroom are the resources of room type. Hence both single bedroom and double bedroom have an *is-a* relationship with the resource room.
- iv. **Contains** define different Web resources encapsulated in a composite Web resource. For example, a search result to a book results multiple instance of the resource books.

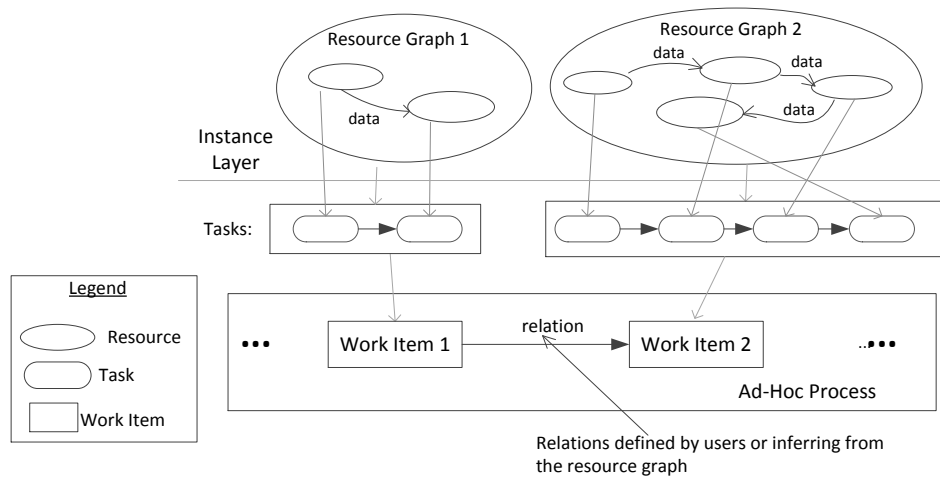
**Transitions based relations:** The response of a Web resource contains next state transition information. A user agent can decide next state based on the semantics of the relations defined in the links available in the response. The relations are used to recommend new Web resources, identify similar Web resources, and define the relationship between the Web resources. Similar to data flow relations, we use link specification to describe the transition based relations.

**Semantic relations:** A semantic relation among Web resources is imposed to Web resources based on the concept relation found in the conceptual layer of three layer architecture. Our approach relates Web resources based on concept shared between them. We identified concepts between resources and then identify the related between the concepts between resources. We propagate this relation down to the resource layer. Figure 9 shows an example resource graph. The concepts in two services are “hotel” and “review”. Since Hotel and review are semantically related, this relation can be propagated to the instance layer, connecting the review and the hotel resources. Moreover the analysis of input and output parameters if there is some data flow relation. In this particular case in Figure 9 the review resource requires the name of the hotel and its location as input parameter. The hotel resource is the initial resource. The review resource is not hosted by the same provider, but is linked with “hasReview” relation. Double bedroom and single bedroom resources have is-a

relationship with room, all semantic relation from the room resource is carried over to those two different categories of the rooms. The small circle represents the method that can be invoked on Web resources from the current state. In our resource graph, the resource from where a user can start consuming a service (a starting point) is defined as the initial node. In Figure 9, the hotel resource is represented in a darker color and it denotes the initial node. When a user requests a service, this node is returned; and from that node, a user can start using the resource. The semantic information embedded in the nodes (i.e., rel attribute in link) can substantially increase the user agents' capability to discover Web resources.

## 5 Generating Ad-hoc Processes

A resource graph describes the relations among multiple Web resources. However, when a user needs to fulfill the goal of a daily activity, the goal generally only involves a very small subset of Web resources defined in the resource graph. Ad-hoc processes are designed to describe and organize such a subset of Web resources. This section presents the definition of ad-hoc processes. Then we introduce our approach to generate ad-hoc processes based on the input of users, expert knowledge, and the resource graph.



**Figure 10:** Definition of an ad-hoc process

### 5.1 Definition of Ad-hoc Processes

An ad-hoc process records the work items that need to be performed for achieving a goal. Figure 10 illustrates our hierarchical definition of ad-hoc processes that can be represented in different levels of abstraction. A task, the lowest level of abstraction, is defined as an operation (i.e., method in the uniform resource schema) on a resource. For example, a task “search for flight ticket” could be implemented as the resource “flight ticket” with the associated operation “GET”. There exists a mapping between

the methods of Web resources captured in a resource graph and the tasks in the ad-hoc processes.

A transaction often consists of one or more tasks. Therefore, we define the notion of a work item to specify a set of tasks along with their data flow for fulfilling a transaction. For example, to fulfill the transaction of “buy flight tickets”, a user needs to perform tasks “searching for ticket”, “choosing ticket” then “paying the bill”. The data “searching results of tickets” is passed from the first task to the second task, and the data “ticket price” is passed from the second task to the third task. In a higher level of abstraction, a work item can contain one or more tasks. An ad-hoc process aggregates work items and existing ad-hoc processes (*i.e.*, sub ad-hoc processes) performed by a user to fulfill a goal. We define the follow three control relations among work items.

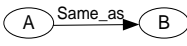
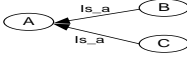
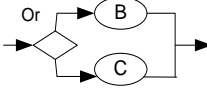

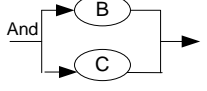
- **And** relation indicates that all the work items have to be executed.
- **Or** relation means that users only need to execute one task from a given set of work items.
- **Optional** is used to recommend related work items after one work item is performed. For example, after a user performs the work item “buying a flight ticket”, we may use optional relation to recommend a work item “map of the airport” to a user.

The methods of the Web resources fulfilling the tasks are connected by data links in the resource graph. The data links make tasks archiving a work item highly coupled. A user needs to execute all the tasks linked by the data in the work item. For example, a work item “buy flight tickets” includes tasks “search ticket”, “choose ticket” and “pay the bill”. To accomplish such a work item, the user has to perform all three tasks within the work item. The relations among work items are loosely coupled. Users can choose different work items to perform, add a new work item or remove an existing work item from the ad-hoc process. Therefore, various users can create personalized ad-hoc processes with different work items to achieve the same goal. For instance, when planning a trip, some users may prefer taking flight than driving car; but other users may prefer driving than flying. The ad-hoc processes of planning a trip for these user groups contain different work items since the ad-hoc process for the former group contains the work item “buy flight tickets” and the latter does not have.

## 5.2 Generating Ad-hoc Processes

Our framework allows a user to specify a goal using a collection of keywords, and helps a user compose an ad-hoc process to fulfill the goal. We use the goal description to find a matching ontology to extend the semantic meanings of the goal. An ontology represents knowledge as common entities (*e.g.*, people, travel and weather), and the relations among the entities. In ontology, the semantic of a high-level goal is expanded into one or more concrete entities. Figure 11 illustrates an example ontology that defines “travel”. The semantic of a high level entity (*e.g.*, “travel”) can be further expanded into four entities: “transportation”, “accommodation”, “tourist Attraction” and “car rental”.



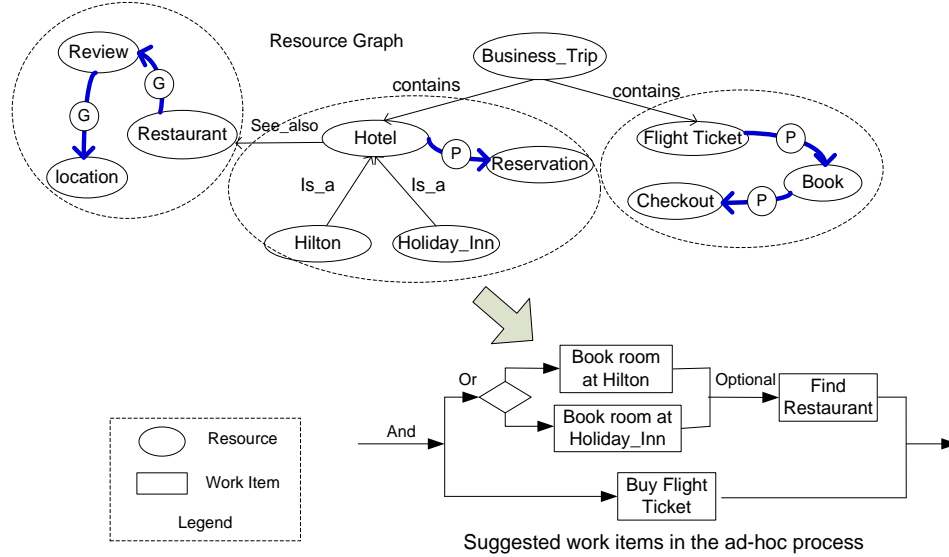
		needs to perform one of the tasks.
		Web Resources B and C are instance of resource A. Thus, resources B and C are similar. A user only needs to choose one from Web resources B and C
		Web Resource A contains Web resources B and C. Therefore, to handle A, users might need to perform the work items related to both B and C.

To group the Web resources with similar functionalities, we design an algorithm to identify tasks for the ad-hoc process. The details of the algorithm are described in described in our earlier paper [42]. In the algorithm, we take an ontology which matches with the goal description as the input. The algorithm uses a stepwise approach to discover and organize the Web resources according to the level of abstraction. The high level entities in an ontology graph convey more abstract meanings suitable for discovering Web resources offering general purpose services. Such services allow users to receive the desired Web resources (*e.g.*, expedia.com) in one place without having to go through multiple servers for visiting different Web resources. For example, expedia.com provides a general service for planning a trip by providing information, such as car rental, flight ticket purchasing and hotel reservation. The low level entities in an ontology graph provide more specific meanings of the goal and therefore indicate the possibility to find concrete Web resources which provide more specialized services or information. For example, to check into a flight operated by Air Canada, a user has to visit more specialized Web resources by going to Air Canada website to print their boarding passes and check flight status.

To satisfy a user with varying needs in different levels of specialization, we use the breadth-first search algorithm to scan the ontology graph. We identify the general purpose Web resources from the top of the graph and the specialized Web resources from the low level of entities in the graph. To identify the control relations between tasks, our framework analyzes the relations of Web resources captured in a Web resources graph to infer the control relations among tasks. Table 1 summarizes the mappings from Web resources in a resource graph to the control relations among tasks.

As listed in Table 1, the “See\_also” relation in the resource graph is mapped to an “optional” relation in the ad-hoc process which is used to recommend tasks to a user. The user has the option to perform it or ignore it. The “Same\_as” relation in the resource graph indicates that two resources are equal. Therefore, we convert the “Same\_as” relation into an “Or” relation of tasks. “Is\_a” relation shows that one resource is an instance of another and these instances have the same features. Therefore, the siblings of “Is\_a” relation in the resource graph are converted to “Or” relation of tasks. “Contains” relation indicates that one resource is the composite of other Web resources. Therefore, the elements in a “Contains” relation in the resource

graph is converted to “And” relation among tasks. We can further segment the tasks into a work item if a set of tasks are related to one transaction. The relation among the work items is inferred from the relations among the segment of tasks.



**Figure 12:** An example of relation inference

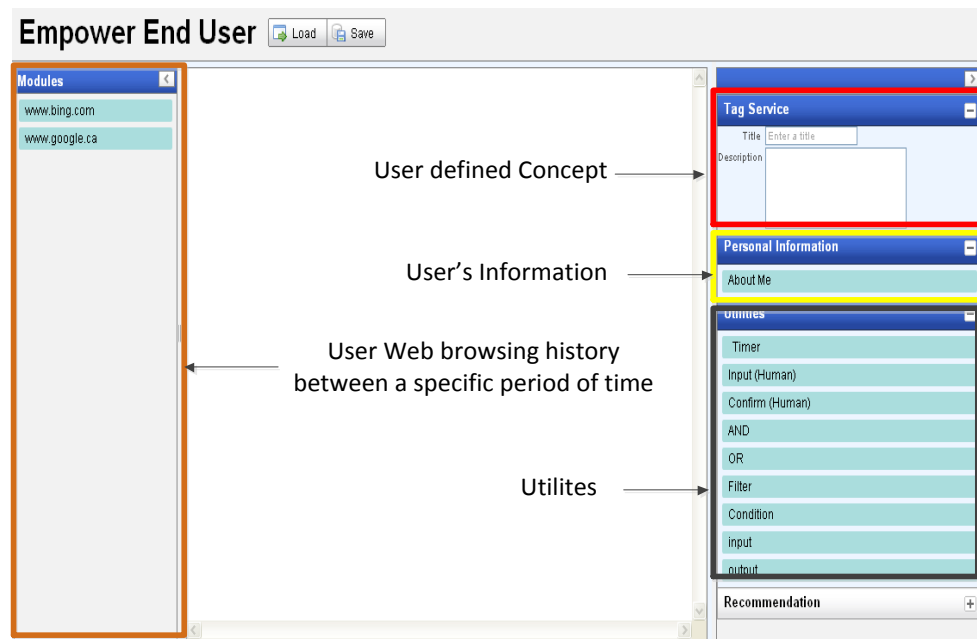
Figure 12 presents an example of mapping the relations defined in the resource graph to the relations of related work items in an ad-hoc process. In Figure 12, the Web resources “Hilton”, “Holiday\_Inn”, “Flight”, and “Restaurant” are converted to work items in the ad-hoc process. In this example, if we trace the entire resource graph, these work items can be implemented by several tasks which are associated with more specialized Web resources.

## 6 A Prototype Implementation

As a proof of concept of our proposed framework, we have designed and developed a prototype personal Web space. We use a Firefox plugin to record a user’s Web browsing history. Our prototype analyzes the browsing history and connects different Web resources visited by users to generate ad-hoc processes in order to automate the repetitive tasks. To collect the Web resources of interest, our prototype allows a user to annotate the Web resources of interest using tags. Such Web resources are recorded and converted to the unified schema.

Figure 13 shows the screenshot of the prototype that helps a user compose Web resources. We implement our prototype using WireIT [41] utility which allows the user to wire different Web resources. The left hand side of our prototype illustrated in Figure 13 shows the Web resource interacted with the user. There are three sections on the right hand side of the screen depicted in Figure 13. The top section (*i.e.*, Tag

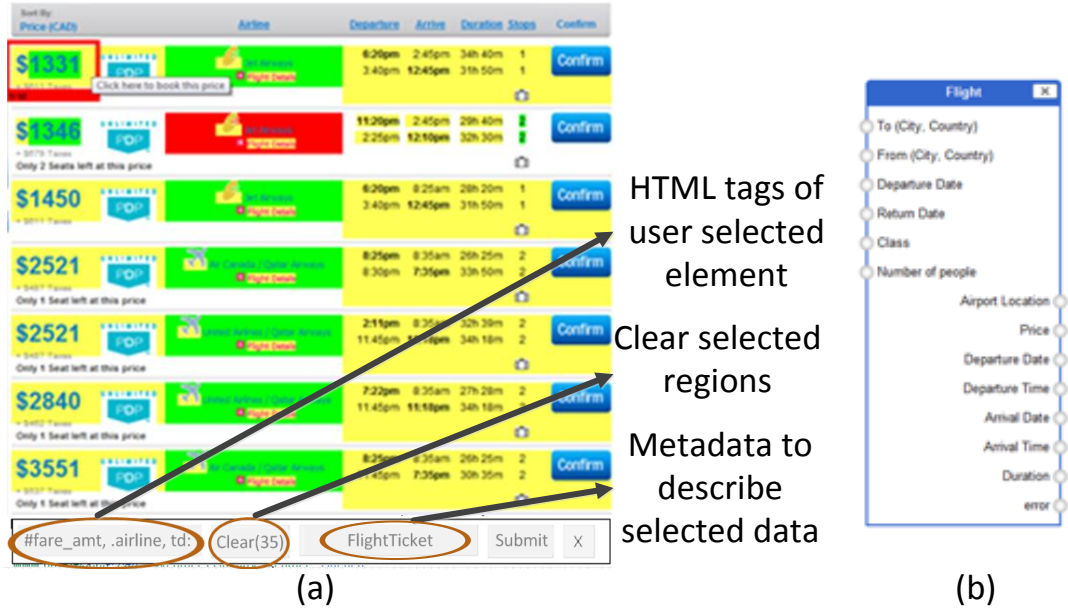
Service) allows a user to tag and describe the functionality of Web resources as concepts. The middle one (*i.e.*, Personal Information) in the right hand side of the screen shown in Figure 13 describes the personal data including address, credit card details, and online accounts. The bottom section (*i.e.*, utilities) contains the utilities that help the user invoke the defined ad-hoc processes following certain criteria, such as timing constraints. The center region enables a user to connect to Web resources.



**Figure 13:** An annotated screenshot of our prototype

As an example usage scenario, a user uses Flight Network to search for the flights. The user goes through the returned Web resources and tags the Web resources of interest using the prototype. In the screenshot shown in Figure 14 (a), a user selects a rate, flight departure date and departure time. The bottom of the interface (Figure 14 (a)) shows the tags of selected elements, name of the component and a submit button. Figure 14 (b) shows the input/output interface for flight selection interface. When a user interacts with more than one Web resources to accomplish a certain task, he can connect the selected Web resources to define the flow between Web resources. We represent the connected the Web resources using the proposed resource graph.



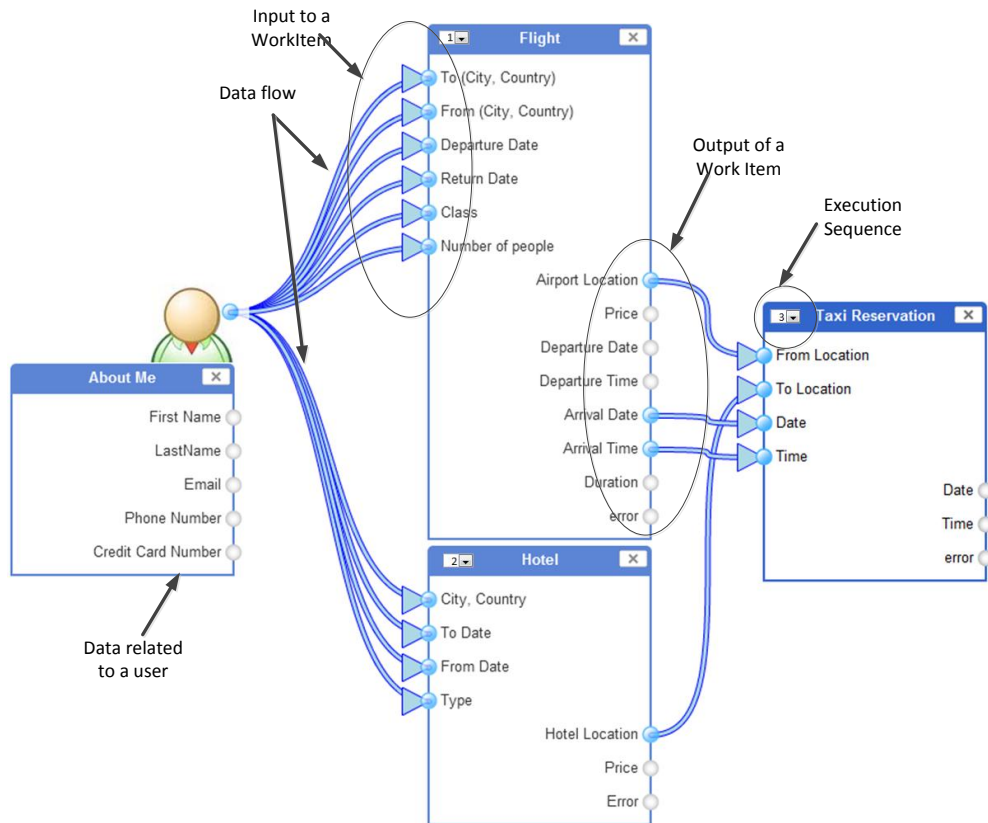


**Figure 14:** (a) Web resources annotated with the information of interest (b) Abstract Web form functionality in a form of input and output

Let us consider a use case scenario for our prototype. Assume that a user wants to attend a conference. The main activities involve booking a flight ticket and a hotel room. He may also need a taxi to transfer from the airport to the hotel. A user needs to enter flight information, like To (City, Country), From (City Country), Date and time Range. Similarly, a user needs the duration (From Date and To Date) and the type of room as input to reserve a hotel room. The taxi reservation can leverage the input from the Flight and Hotel services. The time to rent a taxi is dependent on the time when a flight arrives. A user manually performs initial the activities. Our prototype records all the events occurred in the browser during the given interval of time. The user can then wire different Web resources to produce an ad-hoc process. The user connects these two Web resources to define the data flow. Furthermore, we abstract the ad-hoc process to allow the process to link with different Web resources of the equivalent functionality. Figure 15 shows an ad-hoc process for managing traveling to attend a conference.

## 7 Related Work

Our work is related to three different areas: (a) service migration, (b) data extraction, and (c) service composition. The following subsections describe the related work in the corresponding area.



**Figure 15:** An annotated screenshot for creating an ad-hoc process by a user

## 7.1 Service Migration

A few approaches for migrating legacy systems to Service-Oriented Architectures (SOA) have been proposed in [2]. These approaches use static reverse engineering techniques to identify potential services. RESTful services are relatively new in the area of SOA. To the best of our knowledge, there is not much work done in migration. A few approaches [4, 38, 25] are proposed to model the REST based services. Kopecky et al. [27] present hRESTS as a solution for missing machine-readable Web APIs of RESTful services. They argue that a microformat is the easiest way to enrich existing human-readable HTML documentations. They introduce a model for RESTful services, but with a focus on documentation and discovery. Alarcon et al. [1] introduce a meta-model for descriptions of RESTful services. The meta-model is the basis for the Resource Linking Language. The authors identify links as first class citizens and focus on service documentation and composition. Engelke et al. [9] present an industrial case study to migrate a transportation Web service to a RESTful service and describe issues encountered in designing and implementing a set of

RESTful services to extend and replace traditional Web services. Laitkorpi et al. [25] describe an approach of abstracting Application interfaces to REST-like services. There are mainly three major steps: analyzing a legacy API, abstracting it to a canonical form with constraints in place, and generating the adapter code for the abstraction. Athanasopoulos et al. [4] provide a model-driven approach in identifying REST-like resources from legacy service descriptions. Using the information contained in the descriptions of the available functionality (in the form of WSDL or message schema specifications), the authors proposed a way to model service operations signatures into a MOF model called Signature Model. Different from the above approaches, we represent different Web resources in REST style using a unified description schema.

## **7.2 Data Extraction**

Data extraction is a key feature for many user programming tools. Yahoo Pipes [12] and Potluck [23] can extract data from structured data sources, such as Web services, RSS feeds. Sifter [22] and Solvent [24] are tools that work with unstructured data sources, such as the human-readable information on webpages. Mash Maker [11] supports extraction, aggregation, and visualization across multiple websites. The interface in Mash Maker allows users to directly manipulate URLs, but requires some technical expertise, such as understanding URL arguments and regular expressions. Karma [10] allows the users extract data from a website into a data table through demonstration. To extract data, users visit a webpage and click on the sections within the page they want to extract, such as a restaurant name and address. The tool uses an XPath generalization scheme to find similar data on the same page and other related pages, and then copies the data into a table. Our approach of data extraction is similar to Karma's approach as we use the selector to select the data [33]. In addition, we use HTTP events to abstract the resources.

## **7.3 Service Composition**

Service composition is a process to combine collaborating services to obtain more complex functionality. Mashup performs service composition manually in a user friendly way to compose services without following the formal definition of business processes. Yahoo! Pipes [12] is a visual drag and drop environment for fetching and merging data from different sources. Using Yahoo Pipes [12], a user can combine many feeds into one, then sort, filter and translate it and place them on a personal website. Pipes support a variety of output formats such as RSS, JSON, and KML. Our approach is different from Yahoo Pipes as we abstract ad-hoc processes based on the HTTP events and our approach does not depend on the data formats.

Carlson et al. [8] provide an approach to progressively compose Web services based on the interface matching. Given a Web service, Carlson et al. use the output description of the Web service to match the inputs of existing Web services in the repository. Liu et al. [43] propose a Mashup architecture which extends the SOA model with Mashups to facilitate service composition. Similar to the work of Carlson et al. [8], Liu et al. match the input with output to help end-users compose services.

They also use tags and Quality of Service (QoS) to select services. Our work enhances service Mashups by providing guidance to end-users as they create their Mashups through the automatic composition of services and abstracting services at concept level.

## 8 Conclusion

This paper presents a framework for composing Web resources in a personalized Web space. In the framework, Web resources are described by a unified description schema and are wrapped as RESTful services. The unified schema addresses the issues of integration resources and drives the innovation to the user side. We design a resource graph to represent the semantic relationship among Web resources. By analyzing the relations among Web resources and using ontologies, our framework can generate ad-hoc processes for composing Web resources. We have built a prototype to demonstrate that the repetitive tasks in the Web can be automatically tracked and the user can change simple Web resources into reusable services by annotating the data with them.

In future we will improve our framework to allow a user to share ad-hoc processes. In our current implementation, the resource graph is manually created from the user's browsing history, we plan to provide automatic approach to identify the relations among the Web resources and generate the resource graph for a given set of Web resources. We also want to design case studies to evaluate the performance of our framework for generating ad-hoc processes from a user's goal.

## References

1. R. Alarcón and E. Wilde. "RESTler: Crawling RESTful services." In Proceedings of the 19th international conference on World Wide Web, WWW '10, pages 1051-1052, New York, NY, USA, 2010. ACM.
2. A. Almonaies, J.R. Cordy, T.R. Dean, "Legacy System Evolution towards Service-Oriented Architecture", Proc. International Workshop on SOA Migration and Evolution (SOAME 2010), Madrid, Spain, pp. 53-62.
3. A. Alowisheq, D. E. Millard, and T. Tiropanis, "EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies." International Semantic Web Conference 2009: 941-948
4. M. Athanasopoulos and K. Kontogiannis, "Identification of REST-like Resources from Legacy Service Descriptions, WCRE 2010.
5. D. Beckett, B. McBride (editors), "RDF/XML Syntax Specification (Revised)," W3C Recommendation 10 February, 2004
6. B. Upadhyaya, F. Khomh, Y. Zou, A. Lau and J. Ng, A Concept Analysis Approach for Guiding Users in Service Discovery, IEEE International Conference on Service-Oriented Computing and Applications (SOCA'12), Taipei, Taiwan.
7. R. Chinnici, J. Moreau, A. Ryman, S. Weerawarana, "Web Service Description Language", W3C Recommendation (2007)
8. M. P. Carlson, A. H. H. Ngu, R. M. Podorozhny, L. Zeng, "Automatic Mash Up of Composite Applications," International Conference on Service Oriented Computing (ICSOC) 2008, Sydney, Australia, December 1-5, 2008, pages: 317-330

9. C. Engelke and C. Fitzgerald, "Replacing Legacy Web Services with RESTful Services," WS-REST 2010 First International Workshop on RESTful Design
10. R. Ennals and D. Gay. "Building Mashups by Example", Proceedings of IUI (2008)
11. R. J. Ennals, M. N. Garofalakis, "MashMaker: mashups for the masses," Proceedings of the 2007 ACM SIGMOD international conference on Management of data, ACM.
12. Yahoo Pipes: Rewire the web, <http://pipes.yahoo.com/pipes/>
13. Amazon Advance Search, [http://www.amazon.com/Advanced-Search-Books/b/ref=sv\\_b\\_0?ie=UTF8&node=241582011](http://www.amazon.com/Advanced-Search-Books/b/ref=sv_b_0?ie=UTF8&node=241582011)
14. I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration," Technical report, Global Grid Forum (2002)
15. Dublin Core Metadata Initiative, <http://dublincore.org/>, last accessed on September 9, 2010
16. Facebook, <http://www.facebook.com/>, last time accessed on August 22, 2011
17. R. Fielding. "Architectural Styles and The Design of Network-based Software Architectures". PhD thesis, University of California, Irvine (2000)
18. Flickr, <http://www.flickr.com/>, last time accessed on Aug. 29, 2011
19. Gleaning Resource Descriptions from Dialects of Languages (GRDDL), <http://www.w3.org/2004/01/rdxh/spec>, last accessed on October 12, 2010
20. B. Upadhyaya, R. Tang and Y. Zou. An approach for mining service composition patterns from execution logs. Journal of Software Evolution and Process; DOI: 10.1002/smr.1565, 2011.
21. Google, <http://www.google.com>, last time accessed on August 22, 2011
22. D. Huynh, R.C. Miller, and D. Karger, "Enabling Web Browsers to Augment Web Sites Filtering and Sorting Functionalities," In Proc. UIST 2006. ACM Press (2006), 125-134.
23. D. Huynh, R.C. Miller, and D. Karger. Potluck: Data Mash-Up Tool for Casual Users. In Proc. ISWC 2007. Springer (2007), 239-252.
24. Solvent. <http://simile.mit.edu/wiki/Solvent>
25. M. Laitkorpi, J. Koskinen, and T. Systa, "A UML-based Approach for Abstracting Application Interfaces to REST-like Services," 13th Working Conference on In Reverse Engineering, 2006, pp. 134-146.
26. Link Relations, <http://www.iana.org/assignments/link-relations/link-relations.xhtml>, last accessed on October 12, 2010
27. J. Kopeck'y, K. Gomadam, and T. Vitvar, "hRESTS: An HTML Microformat for Describing RESTful Web Services". In WI-IAT '08: Proc. Int. Conf. on Web Intelligence and Intelligent Agent Technology.
28. P. May, H.C. Ehrlich, T. Steinke, "ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services," In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148--1158. Springer, Heidelberg (2006)
29. MySpace, <http://www.myspace.com/>, last time accessed on August 22, 2011
30. [H. Liu, P. Singh, ConceptNet: A Practical Commonsense Reasoning Toolkit. BT Technology Journal, To Appear. Volume 22](#)
31. RIF In RDF, <http://www.w3.org/TR/rif-in-rdf/>, last accessed on October 12, 2010
32. G. A. Miller, "WordNet: A Lexical Database for English", Communications of the ACM Vol. 38, No. 11: 39-41.
33. SelectorGadget: point and click CSS selectors, <http://www.selectorgadget.com/>
34. H. M. Sneed and S. H. Sneed, "Creating Web services from legacy host programs," 5th International Workshop on Web Site Evolution (WSE), pp. 59-65, 2003.
35. SPARQL Query Language for RDF, <http://www.w3.org/TR/rdf-sparql-query/>, last accessed on October 12, 2010

36. Twitter, <http://twitter.com/>, last time accessed on August 22, 2011
37. B. Upadhyaya, Y. Zou, H. Xiao, J. Ng, and A. Lau, "Migration of SOAP-based Services to RESTful Services," in Proc. Of International IEEE Symposium on Web Systems Evolution (WSE), September 30, 2011, Williamsburg, VA, USA.
38. S. Vinoski, "RESTful Web Services Development Checklist," Internet Computing, IEEE 12, 96–95 (2008)
39. Web Linking, <http://tools.ietf.org/html/draft-nottingham-http-link-header-10#section-4>, last accessed on October 12, 2010
40. Windows Live Messenger, <http://explore.live.com/windows-live-messenger>, last time accessed on August 23, 2011
41. WireIt - a Javascript Wiring Library, <http://neyric.github.com/wireit/>
42. H. Xiao, Y. Zou, R. Tang, J. Ng, and L. Nigul, "Ontology-Driven Service Composition for End-Users", In journal Service Oriented Computing and Applications, 2011
43. X. Liu, G. Huang, H. Mei, A User-Oriented Approach to Automated Service Composition, IEEE International Conference on Web Services (ICWS), Short paper, Beijing, China, September 23-26, 2008, pages: 773-776