Acta Cybernetica **21** (2014) 383–399.

Service Composition for End-Users

Otto Hylli*, Samuel Lahtinen*, Anna Ruokonen*, and Kari Systä*

Abstract

RESTful services are becoming a popular technology for providing and consuming cloud services. The idea of cloud computing is based on on-demand services and their agile usage. This implies that also personal service compositions and workflows should be supported. Some approaches for RESTful service compositions have been proposed. In practice, such compositions typically present mashup applications, which are composed in an ad-hoc manner. In addition, such approaches and tools are mainly targeted for programmers rather than end-users. In this paper, a user-driven approach for reusable RESTful service compositions is presented. Such compositions can be executed once or they can be configured to be executed repeatedly, for example, to get newest updates from a service once a week.

Keywords: service composition, REST, web, WADL

1 Introduction

Use of internet-based services is a routine activity for millions of users. However, the services are often silos and users do not have means to operate and manage their content across the services. Even average PC users can transfer content between applications, but nothing similar is possible for the Internet services they use. In this paper we propose an approach that allows end-users to create compositions for the purpose of combing several internet services or resources.

In service-oriented approaches dominant in the enterprise services, the focus is on the definition of service interfaces and service behavior. Service-oriented architecture (SOA) aims at loosely coupled, reusable, and composable services provided for a service consumer. SOA can be implemented by Web services, which is a technology enabling application integration. Web services can be used for composing high level composite services and business processes. Business processes are often realized as a service orchestration implemented, for example, as WS-BPEL based processes [3]. WS-BPEL is targeted for composing operation-centric Web services utilizing WSDL and SOAP [20,21]. WS-BPEL is close to a programming language defining the logic for a service orchestration. Thus, it is mostly used by IT developers.

*Department of Pervasive Computing, Tampere University of Technology, E-mail: {otto.hylli,samuel.lahtinen,anna.ruokonen,kari.systa}@tut.fi

DOI: 10.14232/actacyb.21.3.2014.7

In cloud-based systems, resources are provided to the user as services via the Internet. On the other hand, the services are accessible anywhere and through several devices. Compared to basic Internet-based service delivery, cloud adds elastic provisioning and release of computing capabilities. Cloud computing and SOA share similar interests on service reuse and service composition. Moreover cloud computing emphasizes on-demand services, which means that services should be ready for use at any time when needed. This also applies for service configurations. Thus, service configuration and composition should be enabled on-line.

Compared to business processes, typical on-demand processes for end-users are personal, simpler, and their lifetime is shorter than traditional business processes. Thus, on-demand processes are often characterized as instant service compositions and service configurations. Such processes are typically defined by the end-user instead of the developer of the cloud services. Due to instant nature of the on-demand processes, their usage and specification should be as simple as possible and require no installation of process development and management tools.

An end-user driven approach for WS-BPEL-based business process development has been proposed in [18]. The approach is targeted for providing a method for easy sketching of service orchestrations. In the proposed approach, a set of scenarios, given as UML sequence diagrams, are synthesized into a process description. However, in the context of cloud computing and on-demand processes, the use of UML modeling and standalone tools is not a proper solution.

Usually, software services in the cloud are targeted for multiple users and they provide a programmable interface, most often a Representational State Transfer (REST) API. REST is a resource-oriented architectural style developed for distributed environments such as for Web and HTTP based applications [5]. RESTful services provide an unified interface (GET, PUT, POST, DELETE) for data manipulation. Thus, composition of such services often includes combining resources and is characterized as mashup-type of development. Some guidelines for mashup development have been proposed (e.g. [14]). Thus the WB-BPEL-based approach is not applicable for cloud-based services and mashups. Composing and orchestration of RESTful services is still lacking tool vendor independent practices and description languages. Thus, the development is often done more in an ad-hoc manner.

SaaS applications are often targeted for end-users. They are self-contained and contain user-interfaces, business rules, and possibly some metadata.

A recent trend is cloud mashups, which combine resources from multiple services into a single service or application [19]. The provider of these service compositions can enhance the cloud's capabilities by offering new functionalities, which make use of existing cloud services, to clients.

In this paper, a novel approach for developing personal service compositions is presented. The approach is targeted for the end-user and allows composition of RESTful cloud services. The approach includes tackling the following issues: (1) easy sketching of service compositions using a simple visual language, (2) a mechanism to export/save composite descriptions for future usage i.e. reusable composite descriptions, and (3) an engine for executing the service compositions, once or repeatedly. The implementation of the approach called Aino service composer is currently under development. The Aino

service composer includes a web browser based editor, which can be used to create simple on-demand service compositions. An earlier version of the tool description has been published in [9].

The rest of the paper is organized as follows. In Section 2, we describe the overall approach and related components. In Section 3, two use cases for end-user driven service composition are presented. Aino service composer is described in Section 4. In Section 5, related work and topics are discussed. In Section 6, conclusions and plan for future work are presented.

2 User-driven approach for service composition

In this paper, an end-user driven approach for defining personal service compositions is presented. The main goal of the approach is on easy design of service compositions, which requires minimal technical knowledge. The service composition is created by using GUI widgets, which are generated based on an imported service description. Widgets present individual resources and they can be dragged and dropped on the canvas. The user can draw dataflow pipes to connect the widgets. Incoming and outgoing dataflows are mapped to REST methods (e.g. outgoing dataflow for GETting a resource presentation).

The implementation of the approach called Aino service composer consists of two components, designer Ilmarinen and engine Sampo. Ilmarinen is a client side application for creating and editing compositions and it is run in a web browser. Sampo is a server side application, which is an engine for running the service compositions. The composition description is given in XML-based format, called Aino description. As a service description format, the approach is based on WADL descriptions [22]. It defines the resources, i.e., URIs, methods, and parameters. That is, while the Aino description specifies the service logic, the WADL description describes the service interface.

Sampo also plays a role of a service registry. Once a service is registered in Sampo, it can be used as a constituent service for future applications. One reason for providing a centralized registry, instead of letting the user search from the web, is that for RESTful services there is no agreement on one service description format. In case a third-party service does not have a compatible WADL description, it can be created afterwards and registered to Sampo. Thus, the approach allows using services, which do not natively provide a WADL description, as reusable constituents.

The approach includes the following steps:

(1) query services from the service registry,

(2) select services to be used as a part of the composition,

(3) composition described as a data flow between services, and

(4) send the composition description to the server engine to be executed.

The main steps are shown in Fig. 1. It also shows the relations between the main components of the system and descriptions, Aino and WADL, which are used for importing and exporting data (i.e. service and composition descriptions).

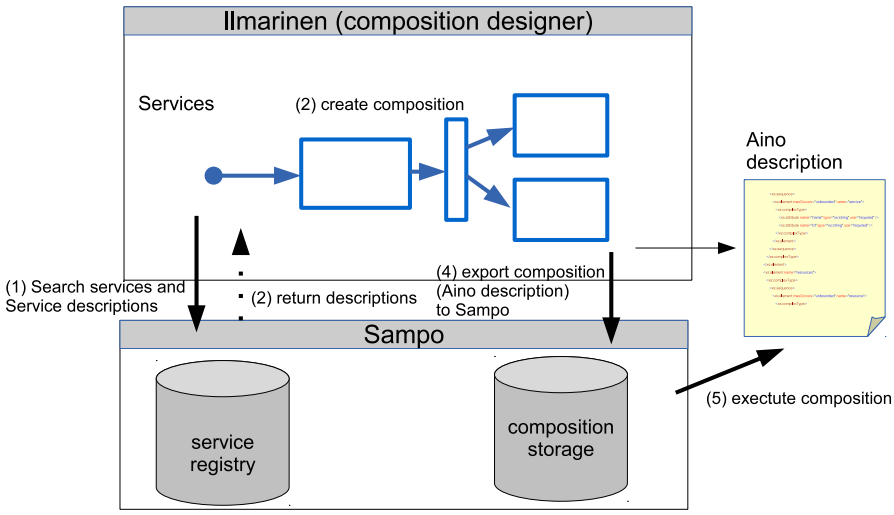

Figure 1: The main steps of the approach

3 Use cases

The following two use cases illustrate the possibilities offered by service compositions for regular internet users. They show how after encountering a normally labor intensive internet based task including multiple services, a user can pretty easily create a service composition that takes care of the task.

3.1 Use case 1: Photos from Twitter to Flickr selectively

An avid Twitter[1] user has been sending many photos taken with his smart phone directly to Twitter. The user wants a better way to organize and share his photos so he opens an account in Flickr[2] which enables him to save photos to different albums, associate keywords to them and decide which photos are public. Uploading all his photos manually to Flickr would be tedious for the user. He would have to go through his Twitter time line, download each photo to his computer and then upload it to Flickr.

To automate the upload process the user wants to create a service composition with Aino service composer. He opens the composition designer Ilmarinen and chooses that he wants to get photos. Ilmarinen shows him a list of services from where he can get photos and he chooses Twitter. From Twitter he chooses that he wants photos from one user which in this case is himself. He also indicates that all photos shouldn't be fetched, instead he will select the ones he wants. Then the user tells Ilmarinen that he wants to upload the photos selected in the previous step. From the services list shown by Ilmarinen he chooses Flickr as the upload target. Additionally he specifies that he wants to choose for each photo

[1] www.twitter.com
[2] www.flickr.com

whether it is private or public. Lastly, he tells Ilmarinen that he wants to delete photos and chooses Twitter. He specifies that from Twitter he wants to delete those photos he has marked as private for Flickr.

When he executes the composition the execution engine Sampo first asks him to authorize Sampo's use of his Twitter and Flickr accounts. Authorization will be done by using OAuth [10] which means that the user authenticates to both services which then give access tokens to Sampo. Sampo will store these access tokens for later use if the user wants it so that next time a service composition using these services is run the user doesn't need to authenticate to the services. He just has to log in to Sampo. When the actual execution has started Sampo will first show the user all his photos from Twitter and asks him to choose those he wants. After that Sampo shows the user his previously chosen photos and asks which of them he wants to be private in Flickr. After the execution has finished Sampo shows the user a execution results summary which tells that the execution was a success and shows how many photos were processed in each step.

3.2 Use case 2: Affordable reading

An enthusiastic book reader uses the Goodreads[3] service to support her hobby. Goodreads is an online community for readers where users can search for books, rate and review them. Users can also categorize books in their profile by adding them to different shelves. One of these shelves is to-read where the user has been adding interesting books, which she has found through Goodreads' recommendation system. She wants to buy some new reading from her to-read shelf but due to her current poor economic situation she wants it to be as cheap as possible. Searching for each book's price from her favorite online book retailer Amazon[4] and then comparing the prices manually would be time consuming so she decides to create a service composition to make the process quicker.

The user opens the service composition designer Ilmarinen and chooses that she wants information about books. Ilmarinen gives the user a list of services that deal with books. The user chooses Goodreads and indicates that she wants the content of a particular user's, in this case hers, particular shelf. Ilmarinen asks the user to input the name of the user and the name of the shelf which in this case are the user's Goodreads user name and to-read. Next the user tells Ilmarinen that she wants online shopping services. From the service list she chooses amazon.com. She specifies that she wants product information about the books from the previous step. Lastly she tells Ilmarinen that she wants the results in ascending order by price. When this composition is run the result is a table containing book information from Amazon including the price and a link to the Amazon product page where the book can be bought.

4 Implementation

The prototype implementation of the Aino service composer consists of two main components: Designer Ilmarinen and engine Sampo. Sampo executes the service compositions,

[3] www.goodreads.com
[4] www.amazon.com

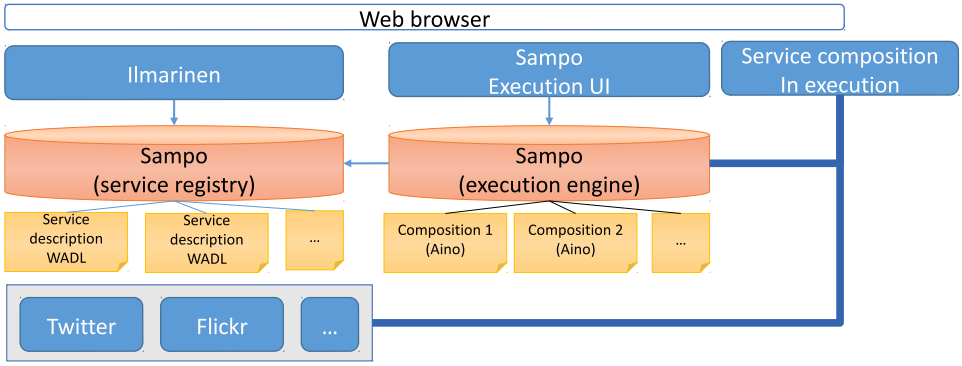

Figure 2: High level architecture of the Aino service composer

stores the service descriptions and offers Ilmarinen access to the information. The separation of the two main componets allows their more independent development. Figure 2 illustrates the high-level architecture of the Aino service composer. The user uses browser-based Ilmarinen to create service compositions. A service composition is a service. Its interface is defined as a WADL document and its execution instructions are defined with the Aino composition description language. Both XML documents are stored in Sampo. The user interacts with engine component Sampo which is used to execute the compositions. The execution and possible user interaction related to the execution is again done in a browser based UI.

4.1 Service description

All the constituent services, as well as the service composition, are described with WADL documents. WADL description defines the service, provided methods and their parameters, as well as data types. The data types can also be defined as separate XML schema files. An example of a simple service description is shown below. It has a partial definition of Twitter's get user timeline method which returns a specified number of tweets from the given user.

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
    <grammars></grammars>
    <resources base="https://api.twitter.com/1.1">
        <resource path="statuses/user_timeline.json">
            <method href="getTimeline"/>
        </resource>
    </resources>
    <method name="GET" id="getTimeline">
        <request>
            <param name="screen_name" style="query" type="xsd:string" />
            <param name="count" style="query" type="xsd:integer" />
        </request>
        <response>
            <representation mediaType="application/json" />
        </response>
```

```
    </method>
</application>
```

4.2 Engine Sampo

Engine Sampo is used in two ways, as a service registry and as an engine to execute the service compositions. Services can be added in the service registry as WADL descriptions. It provides the basic functionality for registration of the services, i.e. API for adding, removing, and searching the services. When a new WADL is added to Sampo the part of the categorization of the service and the resources can be done automatically based on the WADL and an expert user, who understands rest services and WADL, can complete the information and extend the suggested categorizations.

The given metadata is used to offer Ilmarinen lists of the services. For instance, the user can ask to get a list of services related to pictures. Thanks to the metadata Ilmarinen only needs to process WADLs of the services user adds to her composition instead of processing every WADL.

The other part of Sampo provides a REST interface for adding and executing Aino descriptions. The service composition execution uses Aino and the corresponding WADL descriptions for getting the required information on the services and their API. The engine uses this information to invoke correct API calls to the services and combine the tasks to create the complete composite service.

Sampo contains a user interface for handling the compositions. The user can parameterize the composition and define time intervals of execution. In case of a recurring task the service page can be used to start and stop the compositions and change their time intervals. For instance, one could define a service composition that is launched weekly.

Sampo implements simple basic services, for example, for displaying images and news feeds. These are available as components in Ilmarinen and can be added to a service composition in similar fashion as external services.

Sampo is implemented as a Java based web application with the Spring framework[5]. Sampo's implementation is ongoing work. Features that require work include making Sampo work with a creater number of data types and implementing metadata editing for services.

4.3 Designer Ilmarinen

Ilmarinen is a client side application, which provides a graphical interface for creating the service compositions. The user is provided a simple visual environment for defining the service composition. The composition is done partially in a guided manner. A screenshot of an early prototype version of the tool is shown in Figure 3. The user can choose the services e.g. Twitter, BBC Program guide, Weather) she wants based on the service category (e.g. Social media, file storage, picture, program guides). For the services the user can define the interaction and the resources related to the interaction.

In the service composition key elements are the services and data flow between them. After adding a service one can see the input and output possibilities offered by it. These

[5] http://projects.spring.io/spring-framework/

inputs and outputs are parameterized and services are connected to each other using them. When the user has finished, Ilmarinen generates the Aino description. This is exported to Sampo engine for execution. The composition is stored in Sampo and can be accessed directly using a corresponding link. That allows the users to access and execute the compositions directly without using Ilmarinen. This also enables sharing service compositions among different users.

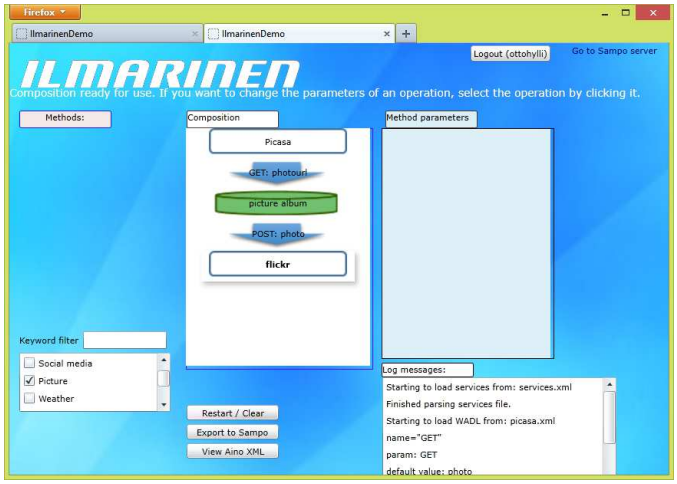

Figure 3: Screenshot of Prototype of Ilmarinen

4.4 Composite description language Aino

Descriptions written in Aino language define the services and resources involved in the composition and the dataflow. A dataflow from one service to another means by getting resource presentation from one service with GET methods and using it as an input to another service using PUT, POST, or GET methods. Services can provide three types of resources: resource out (for GETting a representation), resource in (for PUTting or POSTing), and resource in/out (for PUTting or POSTing and GETting). For data manipulation, control nodes, such as merge and select nodes, are used.

The dataflow can be modeled as an acyclic graph structure, which consists of resources, control nodes, and dataflow connections between them. Control nodes are used for manipulating resource representations, e.g. transforming or filtering data.

In addition to resource, control nodes and dataflow connections, the dataflow includes definition of method calls that are executed when the composition is run. These method calls to the services are presented as GET, PUT, POST, and DELETE elements in the XML description. In addition, the composite service can receive method calls from other compositions using this as a service or from user agent initiation. These are presented as onPUT, onGET, onPOST, and onDELETE elements. Corresponding request and response message types (including data types) are described in the services' WADL documents.

These activities corresponding to REST operations are the same, which are used in BPEL for REST [16] proposal.

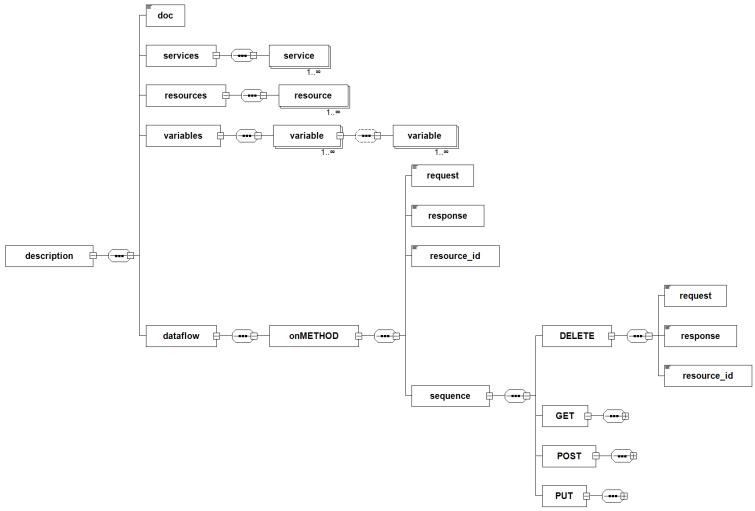

Figure 4: Aino language structure

To enable importing and exporting of compositions, Aino descriptions are transformed in XML format. The structure of Aino language is given in Figure 4. It is explained in detail using an example Aino description given below. The given description presents an example of sending links from Twitter tweets to Instapaper[6]. Instapaper is a service where users can add links to articles they found from the web that they want to read later. Resources part defines two resources, Twitter's user timeline and instapaper's add, which participate in the composition. User timeline returns the desired bumber of tweets from the specified user. Its WADL was an example in section 4.1. Instapaper's add resource adds the link in the url parameter to the account whose username and password are in the respective parameters.

The example composition consists of a receive message and two message invocations. Execution starts when the client invokes GET method on the composite resource (onGET element). Execution continues with a sequence of two invocations. First the composite service invokes GET method on Twitter and second it invokes POST method on Instapaper.

```
<?xml version="1.0" encoding="UTF-8"?>
<description name="tweetlinks2instapaper" >
<doc>Send links from the 10 most recent tweets from the specified user to Instapaper.</doc>
<services>
   <service name = "twitter" id="52d" />
   <service name = "instapaper" id="52f" />
</services>
```

[6] www.instapaper.com

```
<resources>
   <resource uri="https://api.twitter.com/1.1/statuses/user_timeline.json"
             resource_id ="r1" service_id = "52d" />
   <resource uri="https://www.instapaper.com/api/add" resource_id ="r2" service_id = "52f" />
</resources>

<variables>
   <variable name="twitterparams" type="variableset" >
       <variable name="screen_name" type="string" open = "true"  />
       <variable name="count" type="integer" value="10" />
   </variable>
   <variable name="links" type="linklist" />
   <variable name="instapaperparams" type="variableset" >
       <variable name="username" type="string" value = "john.smith@gmail.com" />
       <variable name="password" type="string" value="password123" />
       <variable name="urls" type="variablereference" value="links" />
   </variable>
</variables>

<dataflow>
   <onGET>
      <request></request>
      <response>links</response>
      <resource_id>r_comp</resource_id>
      <sequence>
         <GET>
            <request>twitterparams</request>
            <response>links</response>
            <resource_id>r1</resource_id>
         </GET>
         <POST>
            <request>instapaperparams</request>
            <response></response>
            <resource_id>r2</resource_id>
         </POST>
      </sequence>
   </onGET>
</dataflow>
</description>
```

Variables are used for storing and manipulating message values. For example, the given code listing defines three variables, which correspond to input and output message types of the used GET and POST methods. The variables *twitterparams* and *instapaperparams* are of the type *variableset* which means that they contain multiple variables. These variables contain the parameters for the requests to the services. This is indicated in the Aino description by putting them into the request elements of the service call. The member variables of these sets *screen_name*, *count*, *username* and *password* correspond directly to parameters defined in services' WADLs. So for example Twitter's user timeline method has a parameter named *screen_name*. The variable links contains a list of links. Links from the Twitter method call's response are saved to this variable. How this information is extracted from the response is explained in section 4.5. The links variable is also the response of the composition which means that it will be shown to the user. The variable is also one of the request parameters for Instapaper because of the variable reference in the instapaperparams. Because Instapaper's api doesn't support sending multiple links in one request, the execution engine has to make multiple post requests but this detail doesn't matter to the Aino description.

screen_name is initialized, when the user fills in the required input data, when she

decides to run the composition (see Figure 5). A control interface is used for specifying process instance specific information, such as initial value of process variables and repetition information, which is not part of Aino description.

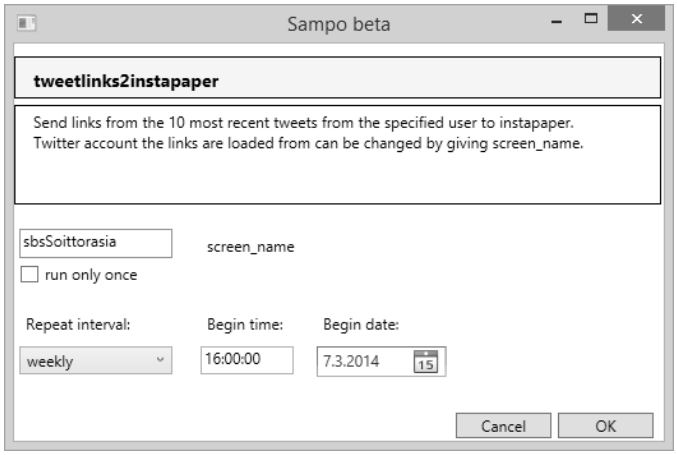

Figure 5: A Control User Interface for the service Compositions

4.5 Data processing

One challenge in combining different internet services into compositions are the different ways the services represent the same data. Many services deal with the same kind of data, e.g. photos or status updates. However, these services represent this data in different ways. One uses XML in representing its resources while another uses JSON. Even if both services in a composition use the same format the schema would very probably be different. Below is an example of how Twitter and Facebook represent a status update. Both service's status update contains the name of the poster, time of the posting and the actual content of the status. They have different names for these attributes and they also have a different time format for the posting time. In addition, each service has additional service specific information about the status update which is not shown here.

```
Facebook:
{
   "id":"201192066592832",
   "from":{
      "name":"Otto Hylli",
      "id":"10883825396030"
   },
   "message":"Hello, world.",
   "updated_time":"2012-05-15T20:35:25+0000",
}

Twitter:
{
   "text": "Hello, world",
   "id": 377326766385573888,
```

```
   "user": {
      "id": 918830997,
      "name": "Otto Hylli"
   },
   "created_at": "Mon Aug 16 17:45:23 +0000 2013"
}
```

Our solution for this problem is to define a set of generic data types that internet services provide and consume. These types include among others status update, photo, link, location and product. For each data type we define a group of attributes that this kind of data generally has. For instance, a status update has the name of the poster, the content of the status update, and the time of the posting. For a service that returns representations that correspond to a certain data type the representation needs to be mapped to the data type. For example, in Twitter's and Facebook's cases mapping information tells how to build a status object from the JSON. For instance, Where in JSON the posting time of the update is and what the format of the time information is. This means that we need mechanisms to locate the interesting data from a structured document.

For locating the desired information from the representation we use XPath [23] for XML representations and JsonPath [6] for JSON. XPath is a language for addressing specific parts of a XML document. It is based on XPath expressions which select the specified nodes from the XML. JsonPath is a similar system for JSON. XPath and JsonPath based data processing information can be added by an expert user directly to a service's WADL or to the service's metadatain the service registry. In both cases the metadata will contain the required formatting information such as the time format used. For instance, Twitter's time format can be represented with this pattern string: E MMM d H:m:s Z y. The pattern format used is from the standard Java class used in the implementation to parse dates.

In the WADL XPath or JsonPath information is located inside the representation element of a resource's method's response. The information itself is contained in the param elements. The parameter's name is a keyword that tells what kind of information it contains, e.g. the author of a status update. The path attribute of the parameter contains the XPath or JSONPath expression itself. The example below shows the representation elements for Twitter's and Facebook's methods that return a list of status updates. A more refined description of the generic data types and service metadata that uses them will be published in [8].

```
Twitter:

<representation mediaType="application/json">
   <param name="status_text" type="xsd:string" path="$[*].text" />
   <param name="status_creator" type="xsd:string" path="$[*].user.name" />
   <param name="status_posted" type="xsd:string" path="$[*].created_at" />
</representation>

Facebook:
<representation mediaType="application/json">
   <param name="status_text" type="xsd:string" path="$[*].message" />
   <param name="status_creator" type="xsd:string" path="$[*].from.name" />
   <param name="status_posted" type="xsd:string" path="$[*].updated_time" />
</representation>
```

5 Related work

The idea of cloud computing is based on on-demand services, which are provided as SaaS applications. In the cloud, traditional business process management tools are already available as SaaS. However, they are targeted for design and management of structured business processes. Requirements for on-demand processes differ from traditional BPM. The ideal solution is to provide an easy and instant mechanism to support execution of personal and dynamic processes, which utilize existing SaaS applications available on the cloud.

5.1 Tools for mashup development

Ad-hoc processes are often expected to live only for a short time. The lack of documentation and proper design might make them single-use only. Thus, they may not be reusable and flexible, but they always need to be recomposed.

JOpera [15] is an Eclipse-based tool build for composing SOAP/WSDL and RESTful Web services. For software developers it provides many useful features such as process modeling, debugging and execution. For composing RESTful services JOpera uses BPEL for REST [16]. BPEL for REST is an extension to WS-BPEL to support compositions of RESTful Web services. The approach does not rely on usage of WSDL or other service descriptions. Resources are defined in the BPEL for REST description as a resource construct, which defines the resource URI and supported operations.

In [13], Marino *et al.* present HTML5-based prototype tool support for mashup development. They present a visual language for service composition. However, the paper is missing details on the user interface and tool usage. Also, details on the composition description are not given.

In [1], Aghee *et al.* discuss different types of mashups enabled by HTML5. A case example includes a location sensitive mobile mashup. The mashup runs natively in a mobile device and uses the GPS sensor build-in the device. In addition, it uses external Web APIs. Location data is sent to a server, which executes API calls to external services. This enables sharing the application between several uses. Mobile mashups enable use of real-time data gathered from the sensors in a mobile phone, e.g. real-time navigation.

Bottaro *et al.* present a simple visual language for composing location-based services [4]. The user uses a repository of web widgets. Widgets are dragged and dropped to build UI for the application. The application logic is defined by drawing connections between data widgets.

In [7], Grönvall *et al.* present ongoing work on user-centric service composition. GUI elements are prototypes of service invocations, which can be chained to compose data flows among services. They present a lightweight tool support for composing simple dynamic workflows, such as for combining SMS, email, and calendar services. Instead of modeling complicated workflows, the emphasis is on the user experience.

In EzWeb project [11, 12], a service-oriented platform for end-user mashup development has been built. The idea is to provide gadgets (e.g. Twitter, Flickr) the user could add to her "application page" creating a set of different applications and web services. The user can also define dataflow between the gadgets by connecting "events" the gadgets could give, e.g., an image url could be connected to another image displayer gadget that is

able to show the picture. All these gadgets are implemented for EzWeb environment. That is, implementation of their user interface, the way of communicating with servers, their events and event slots, are specific for the EzWeb environment. In our approach, the aim is to provide means to compose existing services together and execute these compositions. Thus, our target is to support composition of any third party services by introducing their service descriptions to our system.

5.2 Describing service compositions

Some approaches for modeling and describing RESTful service compositions have been proposed. Guidelines for UML modeling of RESTful service compositions is presented in [17] by Rauf *et al.* The static resource structure is modeled using class diagrams. The behavioral specification of the composite service is given using state chart diagrams.

In [24, 25], Zhao *et al.* discuss formal describing of RESTful services and resources as well as RESTful composite services. Their main interests is on supporting automatic service compositions. For service compositions they present a logic-based synthesis approach utilizing linear-logic and pii-calculus.

In [2], Alarcon *et al.* state that many of the recent service composition approaches rely on operation-based models and neglect hypermedia characteristics of REST. As a solution for composing RESTful services, they present a hypermedia-driven approach realized by using resource linking language (ReLL) for service description. The approach aims to support machine-clients by enabling automatic retrieving of resources from a web site. For describing the composite resources PetriNets are used. As an example of a composite resource, a social network application was presented.

6 Conclusions

Cloud computing is based on on-demand services, which should be available as needed. Similarly, it should also enable on-demand service compositions. In this paper, an end-user driven approach for personal service composition has been presented. The proposed tool support i.e. Aino service composer includes a composition designer running in a web browser and a server-side engine for storing and executing service compositions. The designer is designed for the end-users and it is used for creating personal service compositions. It focuses on end-user concepts and aims to hide complicated and unnecessary information, e.g. service descriptions, which are handled by the engine. Instead of handling data types, the user is allowed to use concepts such as a picture or a photo gallery. The presented use cases concentrate on combining social media services into a composite service. Also, the user is allowed to define repeatable executions for checking updates from the services.

To characterize the approach, it is designed for cloud environment providing a browser-based tool for building service compositions. It is based on WADL descriptions, which are also used for generating GUI widgets for the end-user. In addition, it enables defining RESTful workflows as a composite service.

Our future work includes finalizing the implementation and conducting case studies on applying the approach utilizing the developed tool support. Our future plans also include experimenting the tool usage with novice users.

References

[1] Aghaee, S. and Pautasso, C. Mashup development with HTML5. In *Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups*, Mashups '09/'10, pages 10:1–10:8, New York, NY, USA, 2010. ACM.

[2] Alarcon, R., Wilde, E., and Bellido, J. Hypermedia-driven RESTful service composition. In *Proceedings of the 2010 international conference on Service-oriented computing*, ICSOC'10, pages 111–120, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F, Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S. Business Process Execution Language for Web Services Version 1.1, May 2003. http://www.ibm.com/developerworks/.

[4] Bottaro, A., Marino, E., Milicchio, F., Paoluzzi, A., Rosina, M., and Spini, F. Visual programming of location-based services. In *Proceedings of the 2011 international conference on Human interface and the management of information - Volume Part I*, HI'11, pages 3–12, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] Fielding, R.T. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[6] Goessner, S. Jsonpath - xpath for json. http://goessner.net/articles/JsonPath/.

[7] Grönvall, E., Ingstrup, M., Pløger, M., and Rasmussen, M. Rest based service composition: Exemplified in a care network scenario. In Costagliola, G., Ko, A.J., Cypher, A., Nichols, J., Scaffidi, C., Kelleher, C., and Myers, B.A., editors, *VL/HCC*, pages 251–252. IEEE, 2011.

[8] Hylli, O., Lahtinen, S., Ruokonen, A., and Systä, K. Resource description for end-user driven service compositions. Submitted to 2nd International Workshop on Personalized Web Tasking (PWT 2014), 2014.

[9] Hylli, O., Lahtinen, S., Ruokonen, A., and Systä, K. Service composition for end-users. In *13th Symposium on Programming Languages and Software Tools (SPLST'13)*, page pp.15, 2013.

[10] Internet Engineering Task Force (IETF), http://tools.ietf.org/html/rfc6749. *The OAuth 2.0 Authorization Framework*, 2012.

[11] Lizcano, D., Soriano, J., Reyes, M., and Hierro, J.J. EzWeb/FAST: Reporting on a successful mashup-based solution for developing and deploying composite applications in the "upcoming ubiquitous SOA". In *Mobile Ubiquitous Computing, Systems,*

Services and Technologies, 2008. UBICOMM '08. The Second International Conference on, pages 488–495, 2008.

[12] Lizcano, D., Soriano, J., Reyes, M., and Hierro, J.J. EzWeb/FAST: reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming web of services. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '08, pages 15–24, New York, NY, USA, 2008. ACM.

[13] Marino, E., Spini, F., Minuti, F., Rosina, M., Bottaro, A., and Paoluzzi, A. HTML5 visual composition of rest-like web services. In *4th IEEE International Conference on Software Engineering and Service Science (ICSESS 2013)*, 2013. To appear.

[14] Mikkonen, T. and Salminen, A. Towards a reference architecture for mashups. In *Proceedings of the 2011th Confederated international conference on On the move to meaningful internet systems*, OTM'11, pages 647–656, Berlin, Heidelberg, 2011. Springer-Verlag.

[15] Pautasso, C. Composing RESTful services with JOpera. In *International Conference on Software Composition 2009*, volume 5634, pages 142–159, Zurich, Switzerland, July 2009. Springer.

[16] Pautasso, C. RESTful web service composition with BPEL for REST. *Data Knowl. Eng.*, 68(9):851–866, September 2009.

[17] Rauf, I., Ruokonen, A., Systä, T., and Porres, I. Modeling a composite RESTful web service with UML. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, pages 253–260, New York, NY, USA, 2010. ACM.

[18] Ruokonen, A., Pajunen, L., and Systä, T. Scenario-driven approach for business process modeling. *Web Services, IEEE International Conference on*, 0:123–130, 2009.

[19] Singhal, M., Chandrasekhar, S., Ge, T., Sandhu, R., Krishnan, R., Ahn, G-J., and Bertino, E. Collaboration in multicloud computing environments: Framework and security issues. *Computer*, 46(2):76–84, 2013.

[20] W3C, http://www.w3.org/TR/wsdl. *Web Services Description Language (WSDL) 1.1*, 2001.

[21] W3C, http://www.w3.org/. *Simple Object Access Protocol (SOAP) 1.2*, 2007. Last visited December 2011.

[22] W3C, http://www.w3.org/Submission/wadl/. *Web Application Description Language (WADL)*, 2009.

[23] W3C, http://www.w3.org/. *XML Path Language (XPath) 2.0 (Second Edition)*, 2010.

[24] Zhao, H. and Doshi, P. Towards automated RESTful web service composition. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 189–196, July.

[25] Zhao, X., Liu, E., Clapworthy, G.J., Ye, N., and Lu, Y. RESTful web service composition: Extracting a process model from linear logic theorem proving. In *Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on*, pages 398–403, Oct.