Service Composition for End-Users

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

Department of Pervasive Computing
Tampere University of Technology
P.O. Box 553, FIN-33101 Tampere, Finland
otto.hylli, samuel.lahtinen, anna.ruokonen, kari.systa@tut.fi

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.

1 Introduction

In service-oriented approaches, 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 orchestrations implemented, for example, as WS-BPEL based processes [1]. WS-BPEL is targeted for composing operation-centric Web services utilizing WSDL and SOAP [2, 3]. WS-BPEL is close to a programming language defining the logic for a service orchestration. Thus, it is mostly used by IT developers.

In cloud computing, resources are provided to the user as services via the Internet. Cloud computing and SOA share similar interests on service reuse and service composition. Moreover, cloud computing emphasis on-demand services, which impose more requirements on flexible service and workflow configurations.

Compared to business processes, typical on-demand processes are personal, simpler, and their lifetime is shorter. 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 [4]. 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.

Software services in the cloud, namely Software-as-a-Service (SaaS) applications, differ from fine-grained IT services, which are typically used to form business processes in SOA systems. SaaS applications are often targeted for end-users. They are self-contained and contain user-interfaces, business rules, and possible some metadata. In addition, such services often provide REST API instead of SOAP interface. Representational State Transfer (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. [6]). Composing RESTful services is still lacking tool vendor independent practices and description languages. Thus, the development is often done more in an ad-hoc manner.

A recent trend is cloud mashups, which combine resources from multiple services into a single service or application [7]. 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 semi-structured approach for developing personal service compositions is presented. The approach is targeted for 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 is currently under development. The proposed tool support include a web browser based editor, which can be used to create simple on-demand service compositions.

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 is presented. The proposed tool support 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 approach is supported by two components, designer Ilmarinen and engine Sampo. Ilmarinen is a client side application running 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 WADL descriptions [8]. It defines the resources, i.e., URIs, methods, and parameters. That is, while the Aino description specifies the service logic, the WADL description describe the service interface.

Sampo also plays a role of a service registry. Once a service is registered in Sampo engine, 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 do 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 a reusable constituents.

The main focus 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 approach includes the following steps:

(1) query services from the service registry,
(2) select services to be used as a part of the compositions,
(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 of the main components and descriptions, Aino and WADL, which are used for importing and exporting data (i.e. service and composition descriptions).

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 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 which enables him to save photos to different albums, associate keywords to them and decide which photos are public. Uploading all his photos manually

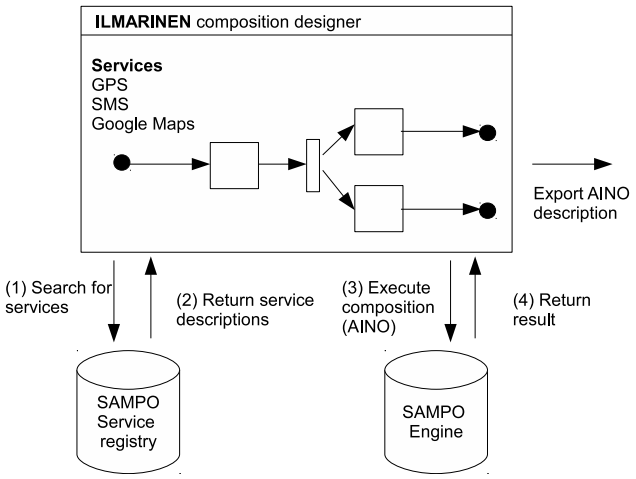

Fig. 1. The main steps of the approach

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. He opens the service composition editor 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. 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 are they 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 [9] 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 service in aid of 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 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 editor 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

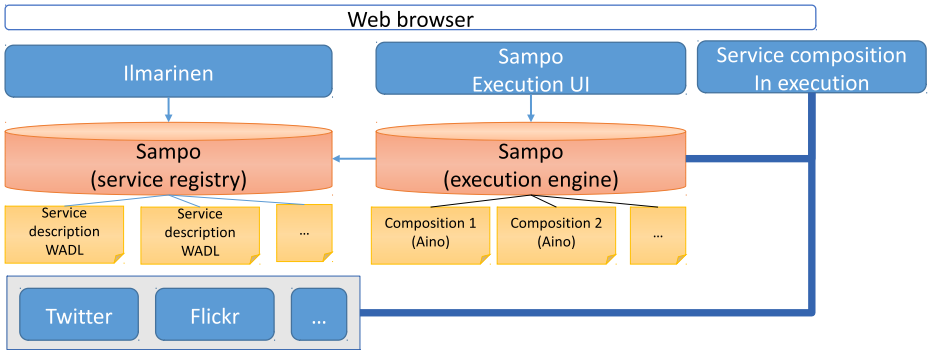

Fig. 2. High level architecture of the system

The prototype implementation consists of two main components: Designer Ilmarinen and Sampo Engine and Service registry. Sampo executes the services compositions, stores the service descriptions and offers Ilmarinen access to the information. Figure 2

illustrates the high-level architecture of the system. The user uses browser-based Ilmarinen to create service compositions. A service composition is a service. Its inteface is defined as a WADL document and its execution instructions are defined as an Aino description. Both XML documents are stored in Sampo. The user interacts with Sampo engine component 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 as a WADL description. WADL description defines the web resources, provided methods and their parameters, as well as data types. Data types can 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 Sampo Engine

Sampo engine 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 the user can complete the information and extend the suggested categorizations.

The given meta-information 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 meta-information 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 an API for executing Aino service 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.

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 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.

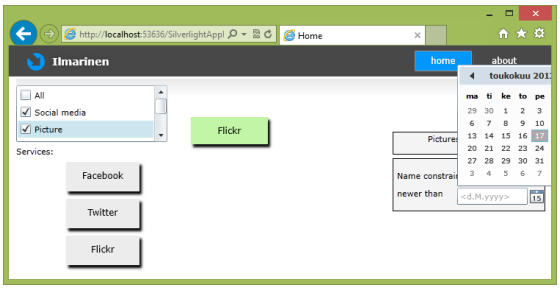

Fig. 3. Screenshot of Prototype of Ilmarinen

4.4 Composite description Aino

Aino description defines the resources involved in the composition and the composite dataflow among resources. 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. Composite dataflows include 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. In addition, data structures used for the resource presentation can be defined by attaching an XML schema to a dataflow or referring to a corresponding WADL file.

The composite dataflow can be modeled as an acyclic graph structure, which consists of resources, control nodes, and dataflow elements between them. Control nodes are used for manipulating resource representations. The main elements to compose the composite dataflow graph are shown in Fig. 4. Each resource is expected to have at most one incoming and outgoing dataflow element.

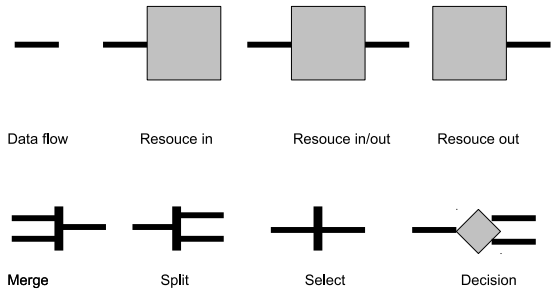

Fig. 4. Dataflow modeling

To enable importing and exporting of the Aino descriptions, composite dataflow graphs are transformed in XML format. The XML description consists of two main parts: resources and dataflow. The former describes all the resources involved in the composition. The latter defines the composite dataflow among the resources.

A simple composite dataflow consists of a sequence of method invocations, which are executed by the composite service on the constituent resources. These are presented as GET, PUT, POST, and DELETE elements in the XML description. In addition, the composite service can receive method calls. 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 [10] proposal.

An example of Aino description is given in the listing below. It presents an example of uploading photos from Twitter tweets to Flickr. Resources part define two resources, Twitter and Flickr, which participate in the composition. The dataflow 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 Flickr.

```
<?xml version="1.0" encoding="UTF-8"?>
<description name="tweet2flickr" >
<doc>Upload photos send to twitter to flickr.</doc>
    <services>
        <service name = "twitter" id="id1"/>
        <service name = "flickr" id="id2"/>
    </services>

    <resources>
        <resource uri="https://api.twitter.com/1.1/statuses/
               user_timeline.json"
        resource_id ="r1" service_id = "id1" />
        <resource uri="http://api.flickr.com/services/upload/"
 resource_id ="r2" service_id = "id2" />
    </resources>

    <variables>
        <variable name="screen_name" type="string" />
        <variable name="photos" type="photolist" />
    </variables>

    <dataflow>
    <onGET>
        <request>screen_name</request>
        <response></response>
        <resource_id>r_comp</resource_id>
        <sequence>
            <GET>
                <request>screen_name</request>
                <response>photos</response>
                <resource_id>r1</resource_id>
            </GET>
            <POST>
                <request>photos</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 two variables, which correspond to input and output message types of the used GET and POST methods. *screen_name* variable presents a user name and it is passed as an input message for the GET method. A return message of the operation call is stored in *photos* variable and it is passed as an input message to the POST method.

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.

In addition to a sequence flow, Aino supports splitting, merging, and conditional branching of data flows. Example structures for *merge*, *split*, and *if-else* patterns are shown in the following listing.

```
<merge>
   <operand>
        activity
    </operand>
    <operand>
```

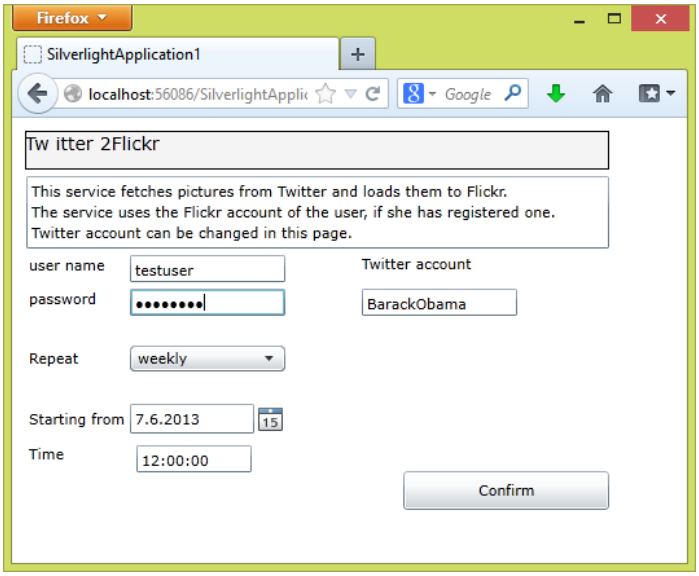

Fig. 5. A Control User Interface for the service Compositions

```
      activity
    <operand>
</merge>
<sequence>
    some activity
</sequence>

<sequence>
    some activity
</sequence>
<split>
    <operand>
        activity
    </operand>
    <operand>
        activity
    </operand>
</split>

<if>
   <condition>some conditon expression</condition>
   activity
   <elseif>*
      <condition>some condition expression</condition>
      some activity
   </elseif>
   <else>?
      some activity
   </else>
</if>
```

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 situation is to provide 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 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 [11] 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 [10]. 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 [12], 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 [13], 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 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 [14]. 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 [15], 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 [16, 17], a service-oriented platform for end-user mashups development have 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, way of communicate 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 [18] 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 [19,20], 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 [21], 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, end-user driven approach for personal service composition have been presented. The proposed tool support includes an editor running in a web browser and a server-side engine for storing and executing service compositions. The editor is designed for the end-users and it is used for sketching 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 services.

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. Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services Version 1.1, May 2003. http://www.ibm.com/developerworks/.
2. W3C, http://www.w3.org/TR/wsdl. *Web Services Description Language (WSDL) 1.1*, 2001.
3. W3C, http://www.w3.org/. *Simple Object Access Protocol (SOAP) 1.2*, 2007. Last visited December 2011.
4. Anna Ruokonen, Lasse Pajunen, and Tarja Systa. Scenario-driven approach for business process modeling. *Web Services, IEEE International Conference on*, 0:123–130, 2009.
5. Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.
6. Tommi Mikkonen and Arto Salminen. 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.
7. Mukesh Singhal, Santosh Chandrasekhar, Tingjian Ge, Ravi Sandhu, Ram Krishnan, Gail-Joon Ahn, and Elisa Bertino. Collaboration in multicloud computing environments: Framework and security issues. *Computer*, 46(2):76–84, 2013.
8. W3C, http://www.w3.org/Submission/wadl/. *Web Application Description Language (WADL)*, 2009.
9. Internet Engineering Task Force (IETF), http://tools.ietf.org/html/rfc6749. *The OAuth 2.0 Authorization Framework*, 2012.
10. Cesare Pautasso. RESTful web service composition with BPEL for REST. *Data Knowl. Eng.*, 68(9):851–866, September 2009.
11. Cesare Pautasso. Composing RESTful services with JOpera. In *International Conference on Software Composition 2009*, volume 5634, pages 142–159, Zurich, Switzerland, July 2009. Springer.
12. Enrico Marino, Federico Spini, Fabrizio Minuti, Maurizio Rosina, Antonio Bottaro, and Alberto Paoluzzi. HTML5 visual composition of rest-like web services. In *4th IEEE International Conference on Software Engineering and Service Science (ICSESS 2013)*, 2013. To appear.
13. Saeed Aghaee and Cesare Pautasso. 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.
14. Antonio Bottaro, Enrico Marino, Franco Milicchio, Alberto Paoluzzi, Maurizio Rosina, and Federico Spini. 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.
15. Erik Grönvall, Mads Ingstrup, Morten Pløger, and Morten Rasmussen. Rest based service composition: Exemplified in a care network scenario. In Gennaro Costagliola, Andrew Jensen Ko, Allen Cypher, Jeffrey Nichols, Christopher Scaffidi, Caitlin Kelleher, and Brad A. Myers, editors, *VL/HCC*, pages 251–252. IEEE, 2011.
16. D. Lizcano, J. Soriano, M. Reyes, and J.J. Hierro. 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.
17. David Lizcano, Javier Soriano, Marcos Reyes, and Juan J. Hierro. 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.

18. Irum Rauf, Anna Ruokonen, Tarja Systä, and Ivan Porres. 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.
19. Xia Zhao, Enjie Liu, G.J. Clapworthy, Na Ye, and Yueming Lu. 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.
20. Haibo Zhao and P. Doshi. Towards automated RESTful web service composition. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 189–196, July.
21. Rosa Alarcon, Erik Wilde, and Jesus Bellido. 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.