Modeling Context-aware and Socially-enriched Mashups

Martin Treiber[1], Kyriakos Kritikos[2], Daniel Schall[1], Dimitris Plexousakis[3], Schahram Dustdar[1]

[1]Vienna University of Technology, [2]Politecnico di Milano, [3]University of Crete

{m.treiber, schall, dustdar}@infosys.tuwien.ac.at, kritikos@elet.polimi.it, dp@csd.uoc.gr

Abstract

Mashup platforms and end-user centric composition tools have become increasingly popular. Most tools provide Web interfaces and visual programming languages to create compositions. Much of the previous work has not considered compositions comprising human provided services (HPS) and software-based services (SBS). We introduce a novel HPS aware service mashup model which we call socially oriented mashups (SOM). The inclusion of HPS in service mashups raises many challenges such as a QoS model that must account for human aspects and the need for flexible execution of mashups. We propose human quality attributes, for example delegation, and a context model capturing various information including location and time. The QoS and context model is used at design-time and for runtime adaptation of mashups. In this paper, we show how to model context-aware SOMs that include HPS and SBS and demonstrate the first results of our working prototype.

Categories and Subject Descriptors D.2.2 [*Software Engineering*]: Design Tools and Techniques; H.3.5 [*Online Information Services*]: Web-based Services

General Terms Mashups, Composition, Context, QoS

1. Introduction

The role of humans in service compositions and workflows has gained tremendous attention. With the proliferation of Web service mashups [21], human aspects became important for designers and the end-users as well. The ability of non experts to create mashup applications for their personal use is considered to increase the productivity of employees.

Currently there is little support for the exploitation of these dynamic aspects in workflows, where humans are integrated into service mashups. In order to tap resources for the creation of flexible and human oriented service compositions, a framework is required that offers the required flexibility and simplicity for the end-user. In particular, we propose the seamless integration of HPS into service mashups. In this context, we refer to these human augmented service mashups as *socially oriented mashups* (SOM) because of phenomena like (sub-)task delegation to other humans during the execution of the mashup. Furthermore, if we consider humans as part of service mashups, we add automatically reasoning capabilities to the mashup. Thus, we make a service mashup flexible and adaptive to unpredictable situations, where human expertise is needed to adapt to new situations.

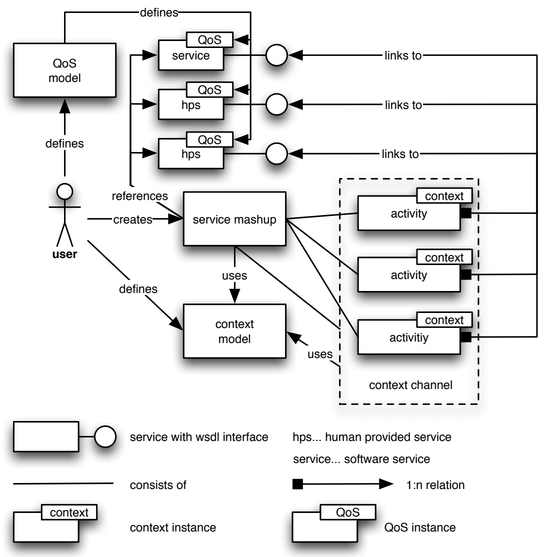
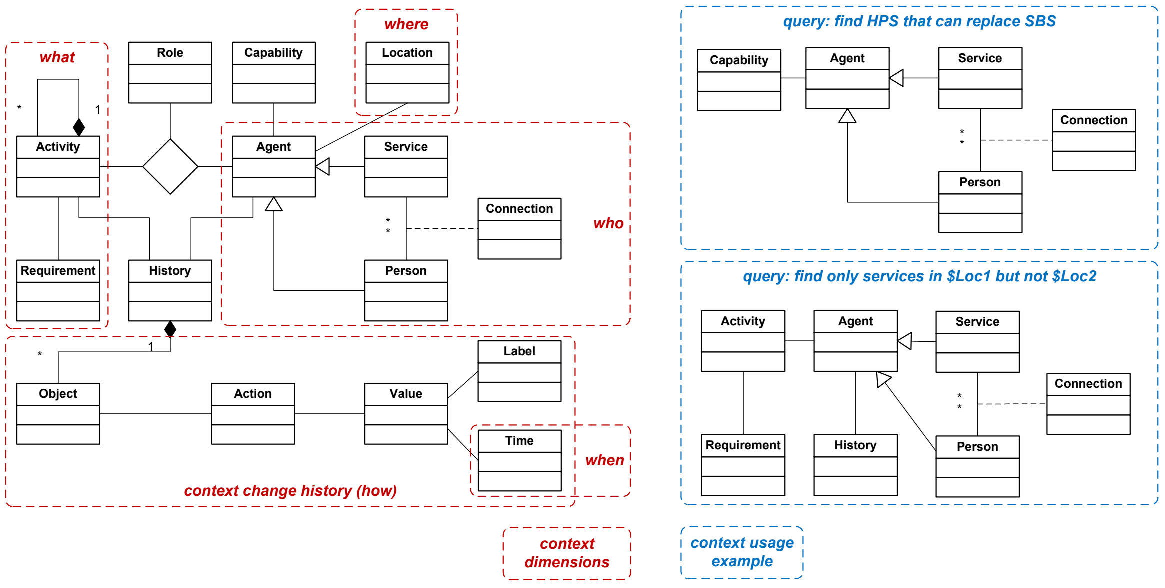


Figure 1. Conceptual Architecture of SOM

However, with the additional flexibility we also face a set of challenges which we address in our work. In particular, we address these challenges with a service mashup model which includes hooks for *context information* and human related QoS attributes that can be exploited during the execution of the mashup (see Figure 1).

In the first step, we create a *mashup model* consisting of HPS and traditional SBS using a lightweight composition language. The focus of this step lies on the functional part of the composition. Since HPS offer the same interface as tradi-



(a) Different Dimensions of Context (b) Using the Context Model

Figure 2. Context Model for Social based Mashup Applications

tional SBS, we do not require additional considerations concerning interface compatibility of HPS and SBS. Considering plain SBS, their description follows the WSDL de-facto standard. By following the HPS concept proposed by [26] we do not require special considerations for the creation of HPS service descriptions. We use WSDL based descriptions that serve as interface to HPSs. This abstraction gives us the benefit of having standardized interfaces to humans which make them *composable* together with SBS in a well defined manner without having to cope with the inherent complexity of human interfaces.

In the second step, we attach *context* related information to the service composition. We refer to this context as design time context in which the creator of the mashup makes assumptions about the expected mashup execution context. The context of the mashup provides information about the environment and introduces global constrains that must be met during the execution of the mashup. For instance, a mashup might not use external services but only internal ones.

Fine tuning of the service mashup takes place in the third step. Within the mashup, the designer is required to define parts which allow a certain *degree of flexibility*, like delegation or splitting. This activity attaches detailed context information to parts of the service mashup which are expected to change during the execution and the given context. For instance, in a mashup, there might be a critical HPS which must not be delegated to others or some services are not allowed to be split among several services.

The rest of the paper is organized as follows. In Section 2 we define our context model for social-based mashup applications. Section 3 introduces our proposed QoS model with emphasis on humans. In the following section we show how to create social service compositions with a lightweight service composition language. The initial results of our prototype are demonstrated in Section 4. We conclude the paper with related work in Section 5 and an outlook for future work in Section 6.

2. Context Model for SOM

We propose a context model that enables the weaving of context related information into social service compositions that are described in a flexible composition language. Our proposed context model aims at satisfying the following properties:

1. The model should be lightweight pertaining information regarding the most relevant entities in SOM. We do not attempt to provide a general purpose context model, but rather focus on SOM applications.
2. Context information comes from various sources inducing physical, e.g., physical location context, and logical sources, for example, a calender containing information regarding the user's location in a given time interval.

Figure 2(a) depicts our context model and some examples of its usage in SOM (see Figure 2(b)). Our model captures the well know dimensions of context (e.g., see [2]) *what* denoting the activity of an agent, *when* capturing time aspects, *who* denoting a person or service, and *how* a set of actions that were executed in the course of an activity. On the other hand (Figure 2(b)) we show examples how to use the context model in SOM (context queries).

The context model contains aspects related to the design of mashups and to their execution (i.e., runtime). At design time, concepts that need to be supported by the mashup tool are *Activity*, *Agent*, *Role*, and *Requirement*.

- **Activity**: Mashup applications comprise a set of activities required to model basic flows (processes). However, the context model does not contain details regarding the process model. Instead, context is related to information such as *Agent* and their *Roles* in *Activities*; roles may include creator of an activity or contributor.
- **Agent**: Both HPS and SBS can contribute to an activity [26].
- **Requirement**: A requirement may restrict the set of actors, which can be invoked in an activity, the roles, and most importantly the values of QoS attributes.

The other model entities address the dynamic nature of context. An *Activity* as well as *Agent* is associated with a *context change history*. Context change, or more precisely, context change events are depicted as *Object*, *Action*, and *Value* triples.

- Events are **actions taken by agents** a) in an activity that is part of a mashup (composition) or b) independent of an activity (person A moves to a new location X). In the first case, an object (person or service) triggers an action (the activity acts as the container for actions) while in the second case, objects change their state independent of any activity.
- Events are **actions acting upon agents**. Such an event may capture invocations of SBS or HPS. In other words, actions capture *functional* capabilities (e.g., interface capabilities) that could change over time. However, actions are typically driven by the context to enable flexible executions of SOM. Changing actions are important for the context-based evolution of mashup-based compositions.

Label and *Time* provide additional metadata to context change events. Labels provide additional information about the action. The value gives the result of an action, whereas labels can be regarded as ”tags” to further quantify actions.

Let us discuss two example context queries as depicted by Figure 2(b) to explain the role of context in SOM:

- Replaceability (*find HPS that can replace SBS*): Attempts to discover a set of HPS that can replace SBS that might be, for example, unavailable due to faults or other technical problems. The HPS must satisfy functional interface characteristics and capabilities in terms of QoS properties. For example, slower response time of a human is acceptable in a given SOM.
- Restriction and filtering (*Find only services in $Loc1 but not $Loc4*): This query selects and filters services based on location information (e.g., exclusive combination of input variables $Loc1 and $Loc4 expressed as *Requirement*). The *History* is used to determine whether an agent, for example, a person has moved to a certain location (changes in location context).

3. QoS Model for Service Compositions

In contrast to interface issues, quality aspects of HPS require special considerations with regard to service compositions. Context has strong impact on QoS properties due to changing availability of services. HPS exhibit fundamentally different quality than SBS. For example, SBS can handle several hundreds requests at once while humans are limited to a small number of parallel requests. On the other hand, humans are able to make complex decisions and to handle data outside the original task specification while SBS can handle these situations only to a very limited extent. Generally speaking, humans maintain a more complex notion of context and are able to maintain multiple contexts at a time. They also have the ability to resolve context when required. Thus, when combining human and software-based services, these differences must be considered in the design of compositions. We propose the extension of existing QoS models with regard to human attributes that reflect human behavior [15].

3.1 QoS Attributes

Table 1 shows a set of quality attributes, along with their definition, that can be applied to both human and machine-based services or to only one of these two types of services. The latter fact (i.e., application) is exhibited with the use of the third and fourth columns of the table. Finally, the last column shows if it is meaningful to aggregate the same quality attribute of both human and software-based services for the composition. If the answer is no, then we can have only local quality constraints on specific tasks of the composition, depending on whether the quality attribute is meaningful to model the quality of the human or machine-based service that is mapped to this task.

We are going to analyze the differences between the same quality attribute for the human and software-based services and we are going to argue why we did not model this attribute for one or the other service type in the corresponding cases that can be drawn from the last column of Table 1.

Throughput: Defines the maximum number of requests that can be completed in a given time interval. Again, there is no conceptual difference between humans and software-based services beside the scale of the respective throughput.

Availability: SBSs are usually available 99.9 percent of their time. On the other hand, the availability of humans varies as it also depends on their context and current load. However, there are usual patterns inferred from users context where availability can be approximately defined for humans (e.g. user’s usual working schedule, days off and holidays, health and mental situation, etc.).

Data quality: This attribute expresses the quality of the data produced by the service. In the way we define HPS, this attribute has the same meaning for both service types that we consider. Moreover, this is an attribute for which we cannot definitely say that its value is better for one service type

Attribute	Description	SBS	Human	Aggregation
Throughput	The number of completed service requests over a time period	yes	yes	yes
Availability	Availability of the service provided to customers	yes	yes	yes
Data Quality	The ability of a data collection to meet user requirements, defined as the proximity of a value v to a value $v^{'}$ considered as correct	yes	yes	yes
Trust	Indicates the service's trustworthiness	yes	yes	no
Delegation	Ability to delegate task to another service	no	yes	no
Soft Completion	Ability to end a task's execution untimely due to time restrictions but with a concrete result produced	no	yes	no

Table 1. Service Quality Attributes

Attribute	Metric/Type/Unit/Monotonicity	Aggregation Pattern
Throughput	Maximum/Positive Integer/(Calls/Sec)/Positive	$th_k\left(CEP\right) = \min_{sp_m^k \in ep_k} \sum_{\substack{t_i \in sp_m^k \\ (t_i,s_j) \in CEP}} th_j$
Availability	(Uptime/Total-Time)/Real in [0.0,1.0]/–/Positive	$av_k\left(CEP\right) = \prod_{\substack{t_i \in ep_k \\ (t_i,s_j) \in CEP}} av_j$
Data Quality	MAPE [17]/Real in [0.0,1.0]/–/Negative	$dq_k\left(CEP\right) = \min_{\substack{t_i \in ep_k \\ (t_i,s_j) \in CEP}} dq_j$
Trust	–/Real in [0.0,1.0]/–/Positive	$tr_k\left(CEP\right) = \prod_{\substack{t_i \in ep_k \\ (t_i,s_j) \in CEP}} tr_j$

Table 2. Service Quality Metric and Aggregation Pattern

than the other one. Depending on the application domain and the context, the situation changes so there is no winning service type. Finally, the data quality of an HPS can increase because a human can learn from the repetition of a task or by exploiting the knowledge acquired or derived from its social network.

Trust: The way this quality is measured for these entities is different because for humans it depends on both subjective and objective criteria while for SBS it depends on only objective criteria [29]. Moreover, an SBS's trust is usually more constant as compared to a human's trust. So, we believe that it is not meaningful to aggregate this quality for a composite service containing both human and software-based services.

Delegation: This attribute concerns the ability of a human to delegate a part or the whole task he is running to other humans, including this human's coordination capability in coordinating the splitted task's execution. SBS can partially support this attribute by delegating a whole task's execution to other instances of the same service. However, they cannot easily delegate parts of a task and coordinate them, if the task has already executed, as this requires special mechanisms. So it is not meaningful to model this attribute for SBS.

Soft Completion: Soft completion refers to the incomplete result of a service execution which is still useable for further activities. For example, a HPS that analyzes images for the occurrence of objects (e.g., trees) might not qualify all objects. However, it might be the case that only a yes/no decision of object occurrences is necessary in the context of the mashup/workflow. This is not true for the side of SBS, as they have to end all their activities before they can produce a specific and complete output.

3.2 Quality Aggregation Analysis

Several quality attributes can be associated with a service. Each attribute would have a short definition, a metric, a value type, a monotonicity and an aggregation pattern associated with it. Monotonicity concerns the way the values that the dimension takes can be compared. In this paper, we distinguish between positive and negative dimensions. A dimension is positive (negative) if the higher the value the higher (lower) the quality or energy level. The aggregation pattern of a dimension defines how the value of this dimension for a composite service can be determined based on the value of the component services. In this paper, we have considered only the most significant and widely-used quality attributes that appear in many research approaches. Table 2 summarizes these attributes based on the above analysis.

CEP denotes a concrete execution plan [13] of a composite service (either HPS or SBS). Each CEP can be transformed [1] to many (e.g., K) concrete execution paths symbolized with ep_k containing a subset of the tasks t_i of the CEP. In addition, each execution path ep_k has a set of *subpaths* (i.e. paths not having parallel tasks) that are indexed by m and denoted by sp_m^k. Every execution path ep_k is associated with a set of aggregated attributes denoted with $attr_k$. The set of services S_i to be executed for a task t_i are called *candidate services* and are denoted with s_j. Each service s_j has a specific set of attributes $attr_j$ derived from its service profile.

3.3 QoS and Context

We argue that QoS attributes are driven by the context in which they are measured. In this regard, we refer to context free QoS attributes and context sensitive attributes. We consider QoS attributes like availability as context sensitive for HPS, because the location context of the human (provided service) influences the availability quality attribute. For instance, if the location of a human changes, the human might not be able to provide the service. Another example for a context sensitive attribute is delegation. Delegation might not be allowed in scenarios where a certain instance of a HPS (e.g., expert that provides an expertise service) is required. In contrast, context free quality attributes are not affected by context. An example is a valid security certificate which is required to invoke a service. In the next section we discuss the use of context and QoS in our proposed mashup model in greater detail.

4. Architecture and Implementation

The main concepts of our approach were defined in the previous sections. This section is dedicated to analyzing the architecture and implementation details of our approach. As discussed earlier, our goal is to benefit from both, human flexibility and the efficiency of SBS for tasks. Conceptually, the composition process consists of two main steps as depicted in Figure 3.

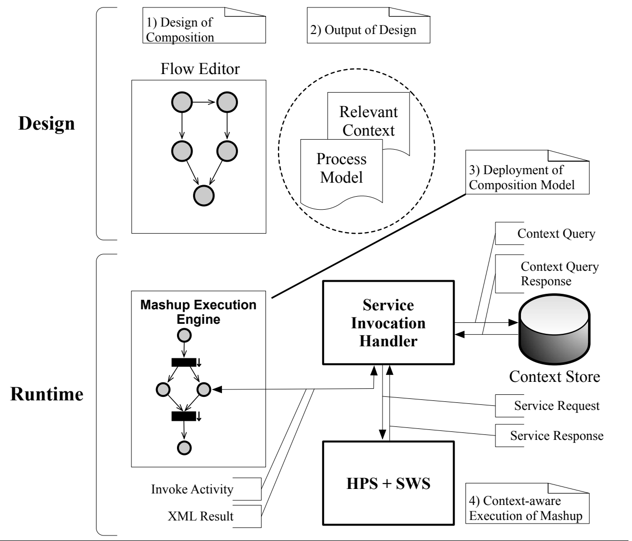


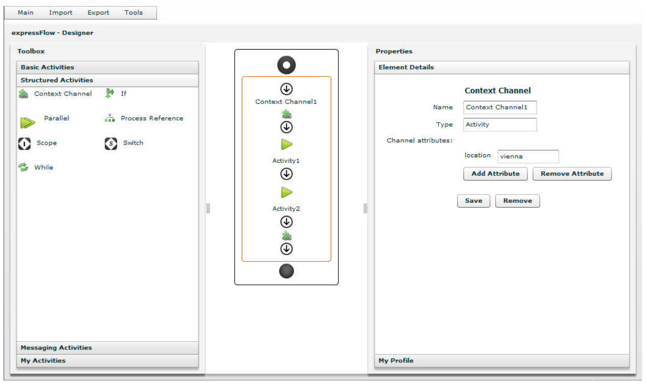
Figure 3. Architecture and Deployment

1. **Design** Firstly, one needs to define the structure of the service mashup. The structure of a service mashup defines from an abstract perspective how services and humans cooperate and which services are used by humans and vice versa. We support this activity with a online tool that enables the creation of social mashups (*Flow Editor*). It must be noted that the flow model contains additional information regarding the *relevant context* that is used during execution of the process model (e.g., context queries).

2. **Runtime** With the use of uniform WSDL interfaces we lay the foundation for a later generation of BPEL workflows. The composition model is deployed in the *Mashup Execution Engine*. An activity is usually performed by invoking a set of services. We call these invocations *actions* that can be performed in a context-aware manner. Thus, the *Service Invocation Handler* interprets context associated to activities to obtain context information via queries from the *Context Store*. Interactions with HPS and SBS (service request and service response) happen in an equivalent service oriented manner through the exchange of SOAP messages. Notice that our approach is not limited to BPEL, since the abstract definition of mashup can be transformed into other languages as well.

4.1 Flow Editor

For a proof of concept prototype implementation, and to illustrate the end user support, we used ExpressFlow [31]. We integrated concepts that are required for the creation of social service compositions in Expressflow, with regard to the limitations imposed by HPS and SBS in service mashups. In our approach, a service mashup structures activities and defines context channels which encapsulate context related information. With the help of a graphical tool (see Figure 4) we support the mashup design process. At this level, the mashup designer defines the basic structure of the mashup using different activities. The tool also provides basic service registry features which supports the designer in selecting services that are suitable in a given context (e.g., filtering of all services that operate at certain locations).



Context Channel
Name: Context Channel1
Type: Activity
Channel attributes:
location: vienna

Figure 4. Screenshot of ExpressFlow online tool

We incorporate context related information directly into the code of the mashup. This approach enables context unaware services to be fully integrated into context aware mashups. By following the separation between context and services we gain the needed flexibility to integrate humans, HPS respectively, into mashups. We discuss the implementation and integration of the basic concepts and their implementation in a top down manner in the following subsections and illustrate our approach with short XML examples that are generated by the Expressflow tool, after having specified the mashup graphically.

4.2 Modeling Context-aware Mashups with Expressflow

Mashup: In service mashups, we embed context channels to structure context information and group actions in activities. These elements can be structured with constructs like *if then else* or *parallel*. Listing 1 illustrates two parallel context channels with different context information using *Parallel* and *ParallelBranch* constructs. Each context channel specifies a context scope (e.g., location, time). During the execution these branches are executed in parallel with the context being evaluated independently.

```
<Process efid="5c21c032-091f-45a0-aaf4..."
name="OOPSLA Service Mashup Demo"
type="Service Mashup" ... >
<Parallel name="Parallel1" type="Activity" ... >
  <ParallelBranch>
    <Context name="Context Channel1"
    type="ContextChannel"
    location="Vienna" time="Today">
   ...
    </Context>
  </ParallelBranch>
  <ParallelBranch>
    <Context name="Context Channel2"
    type="ContextChannel"
    delegation="No" availability="100">
    ...
    </Context>
  </ParallelBranch>
</Parallel>
</Process>
```

Listing 1. Definition of Mashup Comprising Two Parallel Context Channels

Context Channel: Context channels act as flexible containers for service related context information which is specified during the design of the mashup. Conceptually, a context channel defines the scope and the type of context (e.g., location, time, delegation) for nested activities. During the execution of the mashup, all activities of a context channel access the predefined context information and perform the context dependend actions (e.g., data transformation) which are retrieved from a context store.

```
<Context name="Context Channel1"
  type="ContextChannel"
  location="Vienna" time="Today">
  <Activity name="Activity1"
  type="Activity" previous="null" ... />
  <Activity name="Activity2"
  type="Activity" previous="Activity1"
  next="Assignment1" ... />
 </Context>
```

Listing 2. Context Channel Example

Listing 2 presents an example for a context channel, which defines the location context of all included activities to *Vienna* and the time context *Today*. It it worth noticing, that, unless specified differently, all activities are per default executed sequentially (Activity 2 follows Activity 1).

Activity: Activities structure actions which are the hooks for the actual service invocation. The example in Listing 3 shows an activity (*Activity1*) which copies three different values (*appid*, *street* and *city*) to the variable *Variable7* for a sequential invocation of two SBS (*SOAPInvoke3* and *SOAP-Invoke4*) which share the same input parameters.

```
<Activity name="Activity2" type="Activity"  ... />
<Assignment name="Assignment1" type="Activity ... >
  <Copy name="Copy1" type="Activity"
  copy_from="YD–9G7bey8_JXxQP6rxl.fBFGgCdNj..."
  copy_to="$Variable7.appid" previous="null" ... />
  <Copy name="Copy2" type="Activity"
  copy_from="Argentinierstreet+8"
  copy_to="$Variable7.city"
  previous="null" next="null" ... />
   <Copy name="Copy3" type="Activity"
  copy_from="Vienna"
  copy_to="$Variable7.city"  ... />
</Assignment>
<Invoke name="SOAPInvoke3" type="SOAPInvoke"
input="Variable7" output="Variable8" ... >...
<Invoke name="SOAPInvoke4" type="SOAPInvoke"
input="Variable7" output="Variable9" ... >...
```

Listing 3. Asynchronous Activity

Action: Actions represent invocations of services that are executed in the context of an activity. Listing 4 shows how actions are modeled in ExpressFlow. Because of having specified the context on a higher level, we do not require to specify context attributes on this level. Consequently, services do not need to be aware about the context in which they are executed. We discuss how we handle the actual service invocation in Section 4.3.

```
<Invoke name="SOAPInvoke3"
type="SOAPInvoke"
input="Variable7" output="Variable8" ... >
  <Resource
   uri="http://local.yahooapis.com/MapsService...
  appid=$Variable7.appid&amp;
  street=$Variable7.city&amp;
  city=$Variable7.city"/>
</Invoke>
```

Listing 4. SOAP Invoke Example

4.3 Context Store and Service Invocation Handler

Our prototype stores context related information in a MYSQL database. Our database layout is based on the SOAF data model [30] and the context model of Figure 2(a). During runtime, we query the context store for context related actions (e.g., service request transformations) using our service invocation handler. The service invocation handler extracts context information (e.g., time, location, delegation) from SOAP message headers. In our current implementation, we use a simple keyword based search to query for context specific transformations which are represented as XSLT transformations. These transformations are retrieved as strings (streams respectively) for the use of Apache XALAN to transform the request according to the context. Notice that, in our current implementation, we support the transformation of incoming SOAP requests, but do not transform the output accordingly. This is planned for future work.

5. Related Work

Generally speaking, mashups are applications created of existing online resources. [16] categorizes mashups according to four main dimensions: a) what is mashed up, b) where to mashup, c) how to mash up, and d) for whom to mash up. Based on this categorization, there are tools that offer similar functionalities with our approach like "JackBe Presto"[1] , "Procession" [20], "Serena Mashup Suite"[2] , "Swashup" [22], "JOpera" [25] and "remash!" [4]. However, none of these tools is able to offer a context-aware and QoS-based mashup development and execution environment.

Context is any information that can be used to characterize the situation of entities [12]. Context-aware Systems (CASs) are able to adapt their behavior to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking environmental context into account [2]. The behavior of a CAS can be adapted through context in three levels/dimensions [10]: *user interface*, *content*, and *service*. Moreover, this adaptation can be performed in a static or dynamic way by different combinations of services which are independently selected on these three dimensions.

Context has been used in discovery, composition, and adaptation of SBSs. Concerning SBS discovery, context has been used for request (e.g. location info) and input completion (e.g. missing input) so as to increase the quality of the discovery result [6]. Various approaches [9, 23, 24, 28] have been proposed for SBS composition that use local [33] and global contextual constraints for selecting among the candidate SBSs for each task of the composite SBS in a static or dynamic way.

When context changes, composite SBSs may be adapted in three different ways: a) SBS access channel is changed [3], b) an SBS is substituted with another one [3, 28], c) a new concrete execution plan is executed from scratch [14]. Unfortunately, none of the existing SBS composition and adaptation approaches is able to offer simultaneously the three different types of adaptation. Moreover, no approach is able to substitute a single SBS with a new composite one, which might be the case with cooperating HPSs.

Very few mashup approaches have been proposed to take advantage of user or environmental context for adaption purposes. In [5] a system architecture is proposed featuring a context provisioning framework for utilizing local sensors in context-aware mashups. The work described in [19] presents a services mashup system which is able to perform context-aware service composition in a semi- or fully-automatic way and to adapt the results of the composition according to the user's context. Finally, "remash!" [4] offers a framework that enables the flexible binding of services at runtime depending on the changing availability of services or the situation-specific requirement of the application.

Context and its quality can affect the QoS of a service [7]. QoS has been widely studied and researched for SBS discovery, composition [13], and adaptation [1, 8, 11]. However, no QoS-related research work has been conducted for mashups or human-based workflows.

[1] `http://www.jackbe.com`

[2] `http://www.serena.com`

6. Summary and Future Work

In this paper we presented a framework for the integration of humans in socially oriented service mashups. We illustrated the mechanisms to accomplish this with regard to QoS and context. We presented our initial prototype and discussed how we addressed implementation challenges concerning the interpretation of context during the runtime. In future work, we will elaborate our approach with regard to the adaptivity of mashups and extend our core context model with complex actions (e.g., reordering of service invocations in context channels). We are going to extend our QoS model with attributes like accuracy or presentation quality and study these in the context of mashups. Furthermore, we will extend our current prototype with ExpressFlow to BPEL [32] transformations to generate executable BPEL code and study alternative approaches (e.g., using scripting languages or document based approaches [27]). And finally, we will evaluate the performance of our proposed approach thoroughly and study larger examples for service mashups and context.

Acknowledgment

The research leading to these results has received funding from the European Community Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube) and 216256 (COIN). We thank Martin Vasko for help on modeling SOM with ExpressFlow.

References

[1] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Softw. Eng.*, 33(6):369–384, 2007.

[2] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *Journal on Ad Hoc and Ubiquitous Computing*, 2007.

[3] L. Baresi, D. Bianchini, V. D. Antonellis, M. G. Fugini, B. Pernici, and P. Plebani. Context-aware composition of e-services. In *TES 2003*, volume 2819 of *LNCS*, pages 28–41, Berlin,Germany, 2003. Springer.

[4] B. Blau, S. Lamparter, and S. Haak. remash! - blueprints for restful situational web applications. In *Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web at WWW2009*, Madrid, Spain, 2009.

[5] A. Brodt, D. Nicklas, S. Sathish, and B. Mitschang. Context-aware mashups for mobile devices. In *Web Engineering (WISE '08)*, pages 280–291. Springer-Verlag, 2008.

[6] T. H. F. Broens, S. Pokraev, M. J. van Sinderen, J. Koolwaaij, and P. D. Costa. Context-aware, ontology-based, service discovery. In *Symposium on Ambient Intelligence*, volume 3295 of *LNCS*, pages 72–83. Springer, 2004.

[7] T. Buchholz, A. Küpper, and M. Schiffers. Applying web services technologies to the management of context provisioning. In *10th International Workshop of the HP OpenView University Association*, July 2003.

[8] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. QoS-Aware Replanning of Composite Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'05), Orlando, FL, USA*, pages 121–129, 2005.

[9] F. Casati, M. Castellanos, U. Dayal, and M.-C. Shan. Probabilistic, context-sensitive, and goal-oriented service selection. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 316–321, New York, NY, USA, 2004. ACM.

[10] T. Chaari, F. Laforest, and A. Celentano. Design of context-aware application based on web services. Technical Report CS-2004-5, Università Ca'Foscari di Venezia, Venezia, Italy, April 2004.

[11] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava. Adaptation in web service composition and execution. *IEEE International Conference on Web Services (ICWS)*, pages 549–557, 2006.

[12] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness, New York*. ACM Press, 2000.

[13] A. M. Ferreira, K. Kritikos, and B. Pernici. Energy-aware design of service-based applications. In *ICSOC*, LNCS. Springer, 2009.

[14] K. Fujii and T. Suda. Semantics-based context-aware dynamic service composition. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–31, 2009.

[15] R. Kern, C. Zirpins, and S. Agarwal. Managing quality of human-based eservices. *Service-Oriented Computing – ICSOC 2008 Workshops*, pages 304–309, 2009.

[16] A. Koschmider, V. Torres, and V. Pelechano. Elucidating the mashup hype: Definition, challenges, methodical guide and tools for mashups. In *Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web at WWW2009*, Madrid,Spain, April 2009.

[17] K. Kritikos. Qos-based web service description and discovery. Phd thesis, Computer Science Department, University of Crete, Heraklion, Greece, December 2008.

[18] K. Kritikos and D. Plexousakis. Mixed-Integer Programming for QoS-Based Web Service Matchmaking. *IEEE Transactions on Services Computing*, 2(2):122–139, 2009.

[19] Y. Li, J. Fang, and J. Xiong. A context-aware services mashup system. In *Seventh International Conference on Grid and Cooperative Computing (GCC '08)*, pages 702–712, Shenzhen, China, 2008. IEEE.

[20] P. S. Limited. Procession process engine data sheet. Technical report, 2008.

[21] E. M. Maximilien, A. Ranabahu, and K. Gomadam. An online platform for web apis and service mashups. *IEEE Internet Computing*, 12(5):32–43, 2008.

[22] E. M. Maximilien, A. Ranabahu, and S. Tai. Swashup: situational web applications mashups. In *OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 797–798, Montreal, Quebec, Canada, 2007. ACM.

[23] S. B. Mokhtar, D. Fournier, N. Georgantas, and V. Issarny. Context-aware service composition in pervasive computing environments. In *RISE*, volume 3943 of *LNCS*, pages 129–144. Springer, 2005.

[24] S. K. Mostéfaoui and B. Hirsbrunner. Towards a context-based service composition framework. In *ICWS '03*, pages 42–45, Las Vegas, Nevada, USA, June 2003. CSREA Press.

[25] C. Pautasso. Composing restful services with jopera. In A. Bergel and J. Fabry, editors, *Software Composition*, volume 5634 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2009.

[26] D. Schall, H.-L. Truong, and S. Dustdar. Unifying Human and Software Services in Web-Scale Collaborations. *IEEE Internet Computing*, 12(3):62–68, 2008.

[27] N. Schuster, C. Zirpins, S. Tai, S. Battle, and N. Heuer. A service-oriented approach to document-centric situational collaboration processes. In *WETICE '09: Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 221–226, Washington, DC, USA, 2009. IEEE Computer Society.

[28] Q. Z. Sheng, B. Benatallah, Z. Maamar, and A. H. Ngu. Configurable composition and adaptive provisioning of web services. *IEEE Transactions on Services Computing*, 2(1):34–49, 2009.

[29] F. Skopik, D. Schall, and S. Dustdar. The cycle of trust in mixed service-oriented systems. In *SEAA*, 2009.

[30] M. Treiber, H.-L. Truong, and S. Dustdar. Soaf –design and implementation of a service-enriched social network. *Web Engineering*, pages 379–393, 2009.

[31] M. Vasko and S. Dustdar. Introducing Collaborative Service Mashup Design. In *Lightweight Integration on the Web (ComposableWeb'09)*, pages 51–62. CEUR - Workshop Proceedings, June 2009.

[32] WS-BPEL. Business Process Execution Language for Web Services Version 2.0, April 2007.

[33] Y. Yamato and H. Sunaga. Context-aware service composition and component change-over using semantic web techniques. *IEEE International Conference on Web Services (ICWS)*, pages 687–694, 2007.