Tweetflows - Flexible Workflows with Twitter

Martin Treiber, Daniel Schall, Schahram Dustdar
Distributed Systems Group, TU Vienna
Argentinierstrasse 8, A-1040 Wien
{lastname}@infosys.tuwien.ac.at

Christian Scherling
ikangai solutions
Treustrasse 59/5/20
A-1220 Wien
office@ikangai.com

ABSTRACT

We present a lightweight coordination and collaboration platform, intertwining contemporary social networking platforms and SOA principles. The idea of our approach is to use Twitter as a platform for collaborations of human and software services in the context of workflows. We introduce primitives that provide SOA functionality like service discovery or service binding and illustrate how these primitives are embedded in Tweets. By using Tweets, we are able to reuse existing infrastructures and tools (e.g., twitter clients on mobile devices) for the communication between services and humans. Simultaneously, we exploit social network structures originating from Twitter follower networks in order to discover (human and software) resources that are required for the execution of a workflow. Finally, we are able to monitor the execution of workflows with Twitter, simply by following Tweets that represent the execution of a workflow.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications; H.3.5 [**Online Information Services**]: Web-based Services; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Design, Human Factors, Management

Keywords

Crowdsourcing, Twitter, Social Networks

1. INTRODUCTION

Over the past years, the Web has transformed from a pool of statically linked information to a people-centric Web. Various Web-based tools and services have become available enabling people to communicate, coordinate, and collaborate in a distributed manner. *Crowdsourcing* has emerged as an important paradigm in human problem solving techniques on the Web. More often than noticed, software services outsource tasks to humans which are difficult to implement in source code. Applications range from enterprise environments [27] to open Internet-based platforms such as Amazon Mechanical Turk [2] (MTurk). These online platforms distribute problem-solving tasks among a group of humans. Crowdsourcing [12] follows the 'open world' assumption allowing humans to provide their capabilities to the platform by registering themselves as services. Some of the major challenges are: monitoring of crowd capabilities, detection of missing capabilities, strategies to gather those capabilities, and the tasks' status tracking [3]. Service-oriented architecture [1] (SOA) enables the design of applications that are composed from the capabilities of distributed software services. In SOA, compositions are composed of Web services following the loose coupling and dynamic discovery paradigm. Unlike traditional SOA-based approaches, we consider complex service-oriented systems that are established upon the capabilities of human and software services [21]. The integration of human capabilities in a service-oriented manner is motivated by the difficulties to adopt human expertise into software implementations. Instead of dispensing with human capabilities, people handle tasks behind traditional service interfaces. In contrast to process-centric flows (top-down compositions), we advocate flexible compositions wherein services can be added at any time exhibiting new behavior properties. Here, we focus on coordination and collaboration principles using social networks. Specifically, we focus on the *convergence* of service-orientation in crowds and social networking platforms.

A socially-enhanced approach for combining human and software services brings a number of advantages that have not yet been exploited in existing crowdsourcing platforms: (i) Seamless coordination in crowdsourcing environments is accomplished by embedding instructions and flow structure in communications of widely used social networking platforms. (ii) The discovery of collaboration partners and services is based on social phenomena [23]. Spreading and cascading information flows in social networks provide powerful mechanisms for the discovery of resources. (iii) A task's progress can be monitored by observing traces of social interactions. Short infos and status updates that are associated with a particular context (topic) provide information regarding the current status of people and services. Here we present means and primitives for coordination and collaboration using the Twitter[1] [14] service.

[1] `http://twitter.com/`

PESOS'11 May 23-MAY-2011, Waikiki, Honolulu, USA.

Twitter, a service founded in 2006, has 106 million registered users by January 2011 and is adding new users at the rate of 300,000 a day[2]. The underlying principle is very simple: Twitter users post short messages (Tweets) about any topic within a 140-character limit and follow others to receive their Tweets. Being a follower on Twitter let users receive all the messages from those he/she follows. Within the Twitter community, a specialized markup language [13] has evolved: RT stands for retweet, addresses a user directly in a Tweet, and # followed by a word represents a hashtag [5]. This vocabulary enables them to express their Tweets with basic semantics and enables users to search for topics represented by hashtags.

This way of being able to communicate in an informal manner may play an important role for collaborative work in organizations. Twitter provides a lightweight and easy to use tool for sharing work-relevant information among employees, supports the coordination of group activities and it supports potentially novel, previously not anticipated collaboration opportunities within organizations.

Such a lightweight communication framework is certainly of interest when addressing the requirements of software systems which consist of a mix of humans, human-provided services and software services. In such a setting, services are accessed from and provided by different devices (e.g., laptops, mobile phones) and locations (workplace, during travel) and consumed either by humans or by software. Consequently, a convenient communication infrastructure is needed that allows for a flexible handling of communication environments.

In this paper, we investigate the application of Twitter as a communication infrastructure [11]. Our work focuses on how to use Tweets to enable the seamless communication between humans and (human-provided) services in different environments. In particular, we illustrate how SOA principles [6] (i.e, loose coupling, late binding, and service discovery) work with Twitter and how to organize conversations between service consumers and providers on Twitter.

Structure of this Work. We introduce a motivating example in Section 2 that illustrates some of the key challenges for Twitter-based collaborations. Based on our working example, we discuss key design principles in Section 3. We continue with Twitter conversation primitives and show how to apply these in a service-oriented setting in Section 4. We conclude with a discussion on our approach in Section 5, related work in Section 6 and an outlook on future work in Section 7.

2. MOTIVATING SCENARIO

Our motivating scenario is a real world project [3] which has the goal of porting two iPhone apps (qcard and qlauncher) to the Android platform for ikangai solutions. The project team consists of three people of ikangai solutions and four undergrad students from the Vienna Technical University. The project is embedded in a university lecture at the Vienna Technical University and lasts for four months.

A typical challenge in this kind of project setting is to define the communication channels between all project team members. The team members are distributed over three Austrian cities and travel on a regular base. Consequently, the collaboration infrastructure must be accessible from mobile devices (e.g., Android, iPhone, Netbooks) and should be as lightweight as possible. Thus, project-related messages need to be short in order to be easily accessed from mobile devices. Furthermore, the communication must be capable of tracking all project related messages and providing team members the ability to filter the content according their requirements. The sharing of information (links, documents) needs also to be considered, as well as the integration of external (human provided) services that are needed during the project. For example, team members blog during the project about the project's progress on the ikangai blog[4]. Because not all team members write English blog entries, ikangai uses a (human-provided) translation service that translates their entries into English before the entry is posted on the ikangai blog. The very same (human-provided) service also checks ikangai blog entries for grammar mistakes before they are published.

To address these challenges, the ikangai project team uses Twitter as a communication infrastructure backbone. Tweets are used to exchange project-related information and track messages between team members. All project related Tweets contain hashtags that add meta information. The latter are used to filter Tweets for project-related messages, to create message hierarchies and to invoke external (human-provided) services. Moreover, in order to utilize the follower structure of the team members (e.g., other students doing the same lecture) for expertise on different topics (e.g., programming, translation), all project communication is made public. By publishing service requests and retweeting them to Twitter followers (the Twitter follower crowd), we are able to tap into available external knowledge resources, human-provided or software-based services.

3. KEY DESIGN PRINCIPLES

In our approach, we intend to position crowdsourcing in a service-oriented setting, applying concepts from SOA and Web services such as dynamic discovery and late binding mechanisms. In particular, we investigate how to use microblogging services like Twitter for the purpose of crowdsourcing tasks or services.

3.1 Tweetflows and Social Computation

We consider our approach as technical to social computation [18]. As such, social computation is centered around the idea of integrating humans and software into complex systems. One major challenge is the communication between software and humans. Our approach provides the means for a seamless communication between humans and services.

Workflow descriptions are typically complex descriptions of activities that are executed in a particular order to achieve a certain goal. Languages like BPEL or YAWL [24] are used to specify each step that is necessary towards the completion of the workflow. In contrast, we strive for the support of simple, *ad hoc* workflows which are descriptions of activities (e.g., service requests) that are required for the completion of a task and are constructed on the fly, so called *Tweetflows*. After initial *ad hoc* workflow Tweets, which bootstrap the creation and at the same time the execution of the workflow, periodically updates extend the workflow until its comple-

[2]'Twitter Loses Its Scrappy Start-Up Status' in The New York Times: `http://nyti.ms/cBTFUC`

[3]`http://www.ikangai.com/blog/advanced-software-engineering-ase`

[4]`http://www.ikangai.com/blog`

tion, enabling dynamic changes during the execution of the workflow.

3.2 Technology Grounding

In crowdsourcing environments, complex descriptions of services (i.e., service meta data) or service invocation mechanisms are not available. Thus, the process of service publication, service discovery, service binding and service execution cannot be applied directly, but also needs to follow different principles and use different technologies.

- By using Twitter as a communication means for crowdsourcing, we impose a set of limitations concerning the length and complexity of messages that are exchanged during service discovery or service invocation. Twitter, being a mircoblogging service, limits the amount of data that is published to 140 characters per Tweet. Since we want to keep a simple one to one mapping between a Tweet and a service related message, we need to limit the amount of data (and meta data) in a Tweet information to an absolute minimum. In order to minimize to the space needed by meta information, we draw upon Twitter's hashtag mechanism to mark keywords that represent meta information in Tweets [13].
- The messaging mechanism of Twitter follows a broadcast paradigm which we use to publish service requests and service announcements. This is in contrast to having a centralized service registry that collects all service information which is queried for a service. Consequently, we observe that Twitter pushes service announcements instead of letting service requestors pull for service-related information.
- The addressing of services utilizes Twitter's build-in addressing mechanisms using the @ symbol to send messages directly to followers. Followers represent service providers and are able to forward service-related requests to other followers. This also permits to provide the basic means for service communication.

With these limitations in mind, we propose to use a set of primitives to provide for the features to find, bind, execute and monitor services in Twitter-based crowdsourcing environments. Those primitives are directly embedded into Tweets using a combination of hashtags and a priori-defined commands. We foresee the use of external resources that contain the actual data for the invocation of services, because of the aforementioned limitation of 140 characters. In our approach, *Tweetflows* represent conversations between Twitter users, and employ pre-defined communication primitives (e.g., service requests, service bindings) for the invocation of services and the exchange of data. We will discuss the implementation of the processes in greater detail in the following sections, in which we introduce a set of communication primitives that provide the needed functionality to publish, discover and bind services, and to execute simple Tweet-based workflows.

4. TWEETFLOW COMMUNICATIONS

The communication and message exchange in Twitter follows a publish/subscribe pattern [7]. By following other Twitter users, the follower subscribes to the Tweets of the user that is being followed. Given that Tweets are publicly visible to followers, conversations can be tracked by other users and messages can be retweeted, i.e., forwarded to other Twitter followers. This schema supports an efficient spreading of news [17]. The discovery of services is an important issue in many service-oriented systems. By using a spreading mechanism, publication of service-related information becomes straightforward, but also causes some challenges, for example spam or information overload.

The limited length of Tweets demands for a specification of a compact syntax to enable communications and control. For this purpose, we introduce a set of Twitter primitives that enable a seamless fabric of human and service interactions. Our proposed Twitter communication syntax is based on established Twitter primitives, like RT (re-tweet). It not only allows users to request a particular service, but also facilitates the discovery of services in crowdsourcing environments. Table 1 shows the most essential Tweetflow primitives which will be used throughout this section. In our discussions we will establish the correspondence of Tweetflow primitives and concepts found in SOA.

Symbol	Description
SR	Service Request
CS	Claiming Request
RE	Service Response
RT	Retweet Service Request
DS	Delegate Service Request
TF	Tweetflow Pipes
RJ	Reject Service Request
ST	Service State Request
SE	Service State Reply
SP	Service Announcement

Table 1: Tweetflow primitives.

4.1 Tweetflow Primitives

In this section we detail communication principles using Tweetflow primitives and provide simple examples. A complete scenario will be discussed in a later section of this paper.

Passing Data to and from Services. The limitation of Twitter messages requires special considerations concerning the access of input and output data for services. To overcome the size limitation, we propose the use of external resources that represent the input and output of service invocations. Resources are accessed with a simple HTTP GET operation and the corresponding link is stored directly in the Tweet. This allows for great flexibility, because we are able to pass arbitrary information to services. The same applies to the result of service invocations which are represented by Tweets. Listing 1 shows how a (human provided) English to Japanese translation service is called with a blog entry being the input data.

```
1 SR #service:translation.japanese #translation
2    #english #japanese http://bit.ly/9qFRGL
```

Listing 1: Passing data to a translation service.

Announcing Services with Twitter. The publication of services with Twitter consists of posting a Tweet with the service name and meta information about the service. Since the available space is limited, we use optional links

to external taxonomies to provide meta information about the service being published, in addition to Twitter hashtags (see Listing 2). We consider hashtags to play a similar role as tags in folksonomies [26] - a distributed, bottom-up classification schema for services which are made available through Twitter. In addition, the retweet mechanism allows to spread service announcements over the Twitter network [15] providing for a social network-based service publication.

```
SP #service:<name>" "<hashtags>" "<url>
```

Listing 2: Announcing services on Twitter.

Discovering Services with Twitter. The discovery of services in Twitter does not follow the pull approach as with existing SOAs where a service requester searches for candidate service in repositories.

```
// Publication of a Service Request on Twitter
SR #service:<name>"."<operation>" "<hashtags>"
   "<url>

// Direct Service Request on Twitter
SR @<user>#service:<name>"."<operation>"
   "<hashtags>" "<url>

// Delegation of a Service Request on Twitter
DS @<user>#service:<name>"."<operation>"
   "<hashtags>" "<url>

// Spreading a Service Request on Twitter
RT #service:<name>"."<operation>"
   "<hashtags>" "<url>
```

Listing 3: Syntax elements related to discovery.

- *Publication of a Service Request:* With Twitter as communication platform, service discovery follows a push approach where the user Tweets a particular service request. The request, i.e., the Tweet contains meta information about the service, the operation that is required, hashtags providing meta information about the service (data) and a link to the service input data.
- *Direct Service Request:* The service request can also be directed to Twitter followers. In this case, the Tweet addresses the user directly.
- *Delegation of Service Request:* If Twitter followers are not able to handle the service request, but happen to know someone who is, the service request can be delegated to another follower.
- *Retweet Service Request:* Also, a user can retweet the service request to his followers and spread the service request to other users that are no followers of the service requestor. The retweeting or delegating of service requests leads to a dissemination of requests in the Twitter network. As in the case of service announcements, we implicitly use Twitter's social structures during the discovery process.

Binding and Addressing Services. If a Twitter follower is able to provide the required service, the binding of a service request to the service instance happens if the follower directly answers to a service request Tweet. As the Tweet appears in the Tweetflow, the binding is complete and the service is being invoked. The actual addressing of the services uses the built-in Twitter addressing mechanism which sends Tweets directly to followers.

```
CS @<user>#service:<name>" "<url>
```

Listing 4: Claiming a service on Twitter.

Execution of Services and Monitoring. The execution of requests is associated with a state that can be requested. The detailed state model is not the focus of this work as such models have received sufficient attention in collaborative systems. Basically, states covered by our systems include *pending, inprogress, aborted, finished*, to name a few.

```
// Service Response on Twitter
RE @<user>#service:<name>" "<url>

// Service State Request on Twitter
ST @<user>#service:<name>

// Service State Reply on Twitter
SE @<user>#service:<name>" "<state>

// Service Request Reject on Twitter
RJ @<user>#service:<name>
```

Listing 5: Syntax elements related to execution.

- *Service Response:* After the service has been completed, the service provider sends a message containing the service name and a link to the result of the service invocation.
- *Status Request:* During the execution of a service, the service requestor can check the current state of the service execution.
- *Status Reply:* State requests are replied by the user.
- *Reject Service Request:* It is also possible to reject a service request from another user.

Bootstrapping, Executing and Monitoring. The process of creating or bootstrapping a Tweetflow consists of tweeting a set of service requests to find adequate service providers. As briefly discussed in the use case example, the porting of the iPhone applications requires a set of services, respectively service providers. These include, for instance, Android developers, translation services or advertisement services. After the discovery phase is completed (all service requests are bound) the actual execution phase of the Tweetflow starts.

The originator, i.e. the coordinator, of the Tweetflow posts a Tweet that contains a concatenation of actions which need to be executed by different users. This Tweet marks the beginning of the execution of a Tweetflow. A Tweetflow consists of a concatenation of service invocations, resulting in a service pipeline. In the pipeline, each service invocations passes the result to the next service in the pipeline. Listing 6 shows the syntax of Tweetflow Tweets.

```
TF #<name> @<user>#service:<name>" "<url> "|"
   @<user>#service:<name>" "<url>
```

Listing 6: Service pipes on Twitter.

It is worth noting that the control over the execution is distributed: upon completion of a service in the service pipeline, the service provider is directly responsible for tweeting the invocation of the next service by posting a service

request Tweet. Given the ad hoc character of Tweetflows, it's possible to make modifications to the Tweetflow during its execution. For example, a service request can be delegated to another Twitter follower upon receiving a service request, allowing for service replacements on the fly. This open, distributed control approach allows to exploit crowdsourcing behavior in Tweetflows:

Since each service interaction is represented by a Tweet, we imply implicit monitoring of the service execution of the service pipeline. However, in order to track the execution of Tweetflows, we require that each Tweetflow Tweet contains a hashtag with name of the Tweetflow (see Listing 7) to be able to filter for Tweets.

```
RE #<name>" "@<user>#service:<name>" "<url>
ST #<name>" "@<user>#service:<name>
SE #<name>" "@<user>#service:<name>" "<state>
RJ #<name>" "@<user>#service:<name>
```

Listing 7: Tweetflow Tweets.

4.2 Mapping between Tweetflows and SOA

Using Tweetflow principles, we are able to fully support the SOA-lifecycle consisting of *service publication*, *service discovery*, and *service binding* (interactions). However, in human-centric systems, it becomes important to support additional coordination mechanisms (i.e., routing through pipes). Similar (simple) mechanisms are already found in traditional message-oriented systems such as email. Our approach brings the benefit of seamless communication and coordination in a service-oriented manner. The analogy of these primitives to SOA concepts is summarized in Table 2.

Tweetflow Primitives	SOA Concept
SR, RT, DS, RJ	Service Discovery
CS	Service Binding
RE	Service Response
TF	Message Routing
ST, SE	Service Monitoring
SP	Service Publication
@	Service Addressing

Table 2: Mapping SOA principles to Twitter.

In the following, we show a simplified example of a human-provided document translation service as defined by Listing 8. Our approach utilizes the very same technologies used to implement software services such as WSDL and SOAP to support interactions with human-provided services.

First, let us briefly outline the features of the human-provided services (HPS) framework [21] that offers the building blocks to support flexible human interactions in SOA. HPS enhances the traditional SOA-triangle by enabling people to provide services with the very same technology as used by implementations of software-based services. Various operations for different collaborative activities indicate a provider's ability (and willingness) to participate in collaborations. Three steps are performed when using HPS: (i) *Publish:* the user can create an HPS, specify its operations, and publish the service on the Web by using a registry. (ii) *Search:* the requester can perform a keyword-based search to find HPSs. Ranking is performed to find the most relevant HPS based on, for example, the expertise of the user providing the service, or the relevance of a user's experience for a particular request. A user's experience is automatically calculated through interaction mining techniques. (iii) *Interact:* the framework supports automatic user interface generation using standardized XML forms that are generated according to exchanged complex data types in SOAP messages.

```
<?xml version="1.0" encoding="utf-8"?>
<definitions name="TranslationService" ...>
<types>
 <schema targetNamespace="http://...">
  <simpleType name="Language">
   <restriction base="xsd:string">
    <enumeration value="Japanese" />
    <enumeration value="English" />
   </restriction>
  </simpleType>
  <complexType name="GenericResource">
   <sequence>
    <element name="Location" type="anyURI" />
    <element name="Expires" type="dateTime" />
    <element name="Lang" type="tns:Language" />
   <sequence>
  <complexType name="TranslationRequest">
   <sequence>
    <element name="Request"
     type="tns:GenericResource"/>
    <element name="Tags" type="string"/>
   </sequence>
  </complexType>
  <element name="TranslationRequest"
   type="tns:TranslationRequest" />
  <element name="AckSupportRequest" type="string"/>
  <element name="GetStatus" type="tns:GetStatus"/>
  <!-- further requests/responses omitted-->
 </schema>
</types>
<message name="GetTranslation">
 <part name="Parameters"
  element="tns:TranslationRequest" />
</message>
<portType name="HPSPort">
 <operation name="GetTranslation">
  <!-- in-/output... -->
 </operation>
</portType>
<service name="TranslationService">
 <port binding="tns:HPSPort" ... />
</service>
<binding name="HPSSoapBinding" ... >....</binding>
</definitions>
```

Listing 8: WSDL specification of HPS-based translation service (simplified example).

Listing 8 shows a simplified example of a HPS-based document translation service that is provided by one or more human actor(s). The WSDL description defines complex data types such as a `GenericResource` that wraps document-related information (i.e., the location of the document, expiration time, and language attributes). The `TranslationRequest` contains request-related information and additional tags applied to a request. Every request is acknowledged through `AckSupportRequest` by automatically generating a unique identifier. Further operations are supported to obtain the status of a request (e.g., `GetStatus`). Asynchronous notifications can be generated by passing the endpoint of the requester to the service (not shown in this example). The invocation of the translation service is shown in Listing 9.

```
TF @ikangai #service:translation.getTranslation
   #translation #english #japanese
   http://bit.ly/9qFRGL
```

Listing 9: Requesting the SOAP translation service.

5. DISCUSSION

The use of Twitter for the collaboration between team members and services has several implications. First and foremost, using Twitter for communication and coordination means that messages are immediately visible to all followers. Input data of services is also visible, since it is being linked to in the Tweets. If sensitive data needs to be passed, then direct Twitter messages can be sent or the Tweets can be marked as private to prevent non-followers from reading them. Alternatively, users can send direct messages to other users without public Tweets. By using Twitter for service related messages (e.g., discovery, publication), we implicitly make use of information cascades if the corresponding message is retweeted to other followers. As has been discussed in the literature, retweets have the potential to reach large portions of the Twitter network. However, the success rate of information cascades for service requests in an open crowdsourcing setting still needs to be asserted in greater detail. As an experiment ikangai tweeted a request for a Japanese-English translation for the ikangai blog and received an answer from a Twitter user ikangai did not know before. A similar event was observed with a request for an English-Korean translation. In this case, a direct follower of ikangai answered stating that he knew a student able to provide this kind of service. This suggests that information cascades in Twitter can be used for service discovery purposes. Listing 10 shows parts of the Tweetflow that were used to set up the communication and development infrastructure which was explained in our working scenario.

```
SR #service:infrastructure.create #googledocs
#ase_2010_11
SR #service:infrastructure.create #pivotaltracker
#ase_2010_11
SR #service:document.create #kickoff
#ase_2010_11
SR #service:name.create #projectname
#ase_2010_11
SR #service:poll.create #meeting
#ase_2010_11
...
SC @ikangai #service:infrastructure.create
#googledocs #ase_2010_11
SC @ikangai #service:name.create
#projectname #ase_2010_11
SC @redali_25 #service:document.create
#kickoff #ase_2010_11
SC @stefanasseg #service:infrastructure.create
#pivotaltracker #ase_2010_1
SC @ikangai #service:poll.create
#meeting http://bit.ly/bsdbGY
...
TF #ase_2010_11 @stefanasseg
#service:infrastructure.create #pivotaltracker
TF #ase_2010_11 @ikangai
#service:infrastructure.create #googledocs |
@redali_25 #service:document.create #kickoff
TF #ase_2010_11 @ikangai #service:name.create
#projectname
TF #ase_2010_11 @ikangai #service:poll.create
#meeting http://bit.ly/bsdbGY
...
RE @ikangai #service:infrastructure.create
#googledocs #ase_2010_11 http://bit.ly/bfWcyV
RE @ikangai #service:document.create
#kickoff #ase_2010_11 http://bit.ly/bfWcyV
RE @ikangai #service:infrastructure.create
#pivotaltracker #ase_2010_11 http://bit.ly/17VtC
RE @ikangai #service:name.create
#projectname #ase_2010_11 http://bit.ly/ajp8vF
...
```

Listing 10: Tweetflow example.

The core follower structure of the project team was a complete graph because the project team was known a priori and followed each other on Twitter. Consequently, all project-related messages were observed by each team member. However, as discussed before, service requests that couldn't be served were forwarded to other followers of the team members, creating a permeable information boundary. In an enterprise setting, such structures can be mapped onto development teams which work on different areas of a product and which members are highly connected. As shown in the working example, some expertise might be missing and an open messaging infrastructure can be used to extend the expertise of a group. After the kickoff meeting, a Tweetflow for coordinating the activities for the creation of the project infrastructure was initiated. The originator of the Tweet (Twitter user ikangai), bootstrapped the Tweetflow by posting a number of service requests. The service requests were answered by the team members and claimed the services. The originator bound the services to the users and started the execution of the Tweetflow by posting a new round of Tweets. As shown in the example, there were dependencies between actions, like the creation of a Google Docs account and the creation of a Google document which had to be taken into consideration. A polling service was bound by user ikangai using the Tweet `TF #ase_2010_11 ikangai service:poll.create #meeting http://bit.ly/bsdbGY`, integrating the Restful Doodle Poll API into the Tweetflow.

6. RELATED WORK

In service-oriented environments, standards have been established to model human-based process activities and tasks (WS-HumanTask [8]). However, these standards demand for the precise definition of interaction models between humans and services. In our approach, we combine SOA concepts and social principles. We consider *open service-oriented environments* wherein services can be added at any point in time. Following the open world assumption, humans actively shape the availability of services. The concept of Human-Provided Services (HPS) [21] supports flexible service-oriented collaborations across multiple organizations and domains. Similarly, emergent collectives as defined by [20] are networks of interlinked, valued nodes (services).

Open service-oriented systems are specifically relevant for future *crowdsourcing* applications. For example, a hybrid human-computer document translation system has been discussed by [22], however not focusing on the realization as a service-based system. While existing platforms (e.g., MTurk [2]) only support simple interaction models (tasks are assigned to individuals), social network principles support more advanced techniques such as formation and adaptive coordination. Social game-based human computation has been introduce [25] in the context of image labeling that is performed by humans. From the technical point of view, TurKit [16] is a crowd computing framework based on MTurk.

On an architectural level, we follow the blackboard architectural pattern [9]. In our architecture, the *Tweet Bus* plays the role of the blackboard which holds state information, i.e., Tweets of the Tweetflows. Enterprise Service Bus Architectures (ESB) [4] have a strong similarity to our Tweet Bus architecture. Like in ESBs, our approach also uses a centralized communication channel to transport messages. Clients plug into this channel and listen to messages which are transported in a standard format. However, the pub-

lic visibility of Tweets, the 140 character limit for messages and the ability to forward (retweet) messages arbitrarily to other users not listening to the communication bus, i.e., to push messages into a community of followers, are the main differences.

7. SUMMARY AND OUTLOOK

In this paper, we presented an approach for using Twitter as a communication and coordination medium for the execution of simple workflows. A novel application of our approach are collaborations in crowdsourcing environments where people provide their skills and capabilities in a service-oriented manner. In contrast to existing crowdsourcing platforms, our approach enables the realization of *collaborative crowds* where individuals jointly work on activities and short-term projects.

We introduced Tweetflow primitives which are embedded in Twitter communications to control collaborations and to interact with human- and software-based services. By using Twitter, we implicitly provided a platform-agnostic communication backend, which possesses enterprise service bus characteristics: each client is able to plug into the *Tweet Bus* and exchange messages, independent of any implementation. Moreover, a Tweet Bus provides the means to integrate human- and software-services seamlessly. And finally, the Twitter follower structure offers the ability to exploit social structures and to utilize them for service discovery purposes as well as to tap human resources of crowds: service requests can be forwarded (i.e., retweeted) to followers and are thus distributed to the crowd. In future works, we plan to extend Tweetflow primitives to enable the creation of more complex Tweetflows. For instance, we plan to integrate branching conditions, synchronization points and execution deadlines into Tweetflows to allow for more complex Tweetflows. Our prototype will be extended with a graphical tool that facilitates the creation of Tweetflows and we intend to extend the support for SOAP-based Web services and Resful Services [19]. We will investigate Twitter user profiles to include them for service recommendations [10] or delegations.

Acknowledgment

This work received funding from the EU FP7 program under the agreements 215483 (SCube) and 215175 (COMPAS).

8. REFERENCES

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services - Concepts, Architectures and Applications*. Springer, October 2003.

[2] Amazon.com. Amazon mechanical turk, last access: 2010. available online: http://www.mturk.com.

[3] D. Brabham. Crowdsourcing as a model for problem solving: An introduction and cases. *Convergence*, 14(1):75, 2008.

[4] D. Chappell. *Enterprise Service Bus*. O'Reilly Media, Inc., 2004.

[5] M. Efron. Hashtag retrieval in a microblogging environment. In *SIGIR '10*, pages 787–788. ACM, 2010.

[6] T. Erl. *SOA Principles of Service Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.

[7] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[8] M. Ford et al. Web Services Human Task (WS-HumanTask), Version 1.0., 2007.

[9] D. Garlan and M. Shaw. An introduction to software architecture. Technical report, Pittsburgh, PA, USA, 1994.

[10] J. Hannon, M. Bennett, and B. Smyth. Recommending twitter users to follow using content and collaborative filtering approaches. In *RecSys '10*, pages 199–206. ACM, 2010.

[11] C. Honey and S. C. Herring. Beyond microblogging: Conversation and collaboration via twitter. In *HICSS '09*, pages 1–10. IEEE Computer Society, 2009.

[12] J. Howe. *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*. Crown Publishing Group, New York, NY, USA, 2008.

[13] J. Huang, K. M. Thornton, and E. N. Efthimiadis. Conversational tagging in twitter. In *HT '10*, pages 173–178. ACM, 2010.

[14] B. Krishnamurthy, P. Gill, and M. Arlitt. A few chirps about twitter. In *WOSP '08*, pages 19–24. ACM, 2008.

[15] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *WWW '10*, pages 591–600. ACM, 2010.

[16] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. Turkit: tools for iterative tasks on mechanical turk. In *HCOMP '09*, pages 29–30. ACM, 2009.

[17] M. Motoyama, B. Meeder, K. Levchenko, G. M. Voelker, and S. Savage. Measuring online service availability using twitter. In *WOSN'10*, pages 13–13. USENIX Association, 2010.

[18] D. G. Murray, E. Yoneki, J. Crowcroft, and S. Hand. The case for crowd computing. In *MobiHeld '10*, pages 39–44. ACM, 2010.

[19] C. Pautasso. REST vs. SOAP: Making the Right Architectural Decision. In *SOA Symposium*, pages 2009–01. Citeseer, 2008.

[20] C. Petrie. Plenty of room outside the firm. *IEEE Internet Computing*, 14, 2010.

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

[22] D. Shahaf and E. Horvitz. Generalized task markets for human and machine computation. In *AAAI*, 2010.

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

[24] W. M. van der Aalst, L. Aldred, M. Dumas, and A. H. ter Hofstede. Design and Implementation of the YAWL System. In *Lecture Notes in Computer Science*, volume 3084, pages 142–159, 2004.

[25] L. von Ahn and L. Dabbish. Designing games with a purpose. *Commun. ACM*, 51(8):58–67, 2008.

[26] J. Voss. Tagging, folksonomy & co - renaissance of manual indexing? *CoRR*, abs/cs/0701072, 2007.

[27] M. Vukovic. Crowdsourcing for enterprises. In *Congress on Services*, pages 686–692, 2009.