Crowdsourcing Mobile Workflows with Tweetflows

Under Review for Publication

Vienna University of Technology
Information Systems Institute
Distributed Systems Group

Martin Treiber[1], Daniel Schall[1], Schahram Dustdar[1], Christian Scherling[2]
lastname@infosys.tuwien.ac.at[1], office@ikangai.com[2]

TUV-1841-2011-02 March 31, 2011

The use of mobile devices and applications (Apps) offer users ubiquitous access to arbitrary Services. In this paper, we study the applicability of established SOA concepts in mobile computing scenarios. In particular, we investigate the application of existing SOA infrastructure principles like Service registries on App stores and investigate the concept of App as a Service. Based on our findings, we introduce a light-weight flow language that is tailored to mobile Apps with regard to SOA principles. These efforts are complemented with a discussion on crowd sourcing aspects, which drive our proposed approach. We present a prototype architecture and show how our approach can be used in a real world application scenario.

Keywords: service-oriented architectures, mobility, apps, crowdsourcing, tweetflows

Noname manuscript No.
(will be inserted by the editor)

Crowdsourcing Mobile Workflows with Tweetflows

Martin Treiber · Daniel Schall · Schahram Dustdar · Christian Scherling

Received: date / Accepted: date

Abstract The use of mobile devices and applications (Apps) offer users ubiquitous access to arbitrary Services. In this paper, we study the applicability of established SOA concepts in mobile computing scenarios. In particular, we investigate the application of existing SOA infrastructure principles like Service registries on App stores and investigate the concept of App as a Service. Based on our findings, we introduce a lightweight flow language that is tailored to mobile Apps with regard to SOA principles. These efforts are complemented with a discussion on crowd sourcing aspects, which drive our proposed approach. We present a prototype architecture and show how our approach can be used in a real world application scenario.

Keywords service-oriented architectures, mobility, apps, crowdsourcing

1 Introduction

Mobile phones have become increasingly powerful in terms of available memory and CPU resources. Handsets like the Samsung Galaxy S or the Apple iPhone provide a Gigahertz CPU, 512 MB of RAM and offer up to 32 GB of storage capacity to the user. This kind of hardware makes it attractive for users to install a broad range of software on their devices. On average, users install approximately between 14 and 40 Apps on their devices[1]. These Apps offer different functionalities; social networking, weather, sports, location information, dictionaries and games are among the most common classes of Apps being downloaded. A major advantage of modern mobile phones and wireless communication infrastructures is the ubiquitous Internet access. With good 3G network coverage, users can access remote Web services from practically everywhere. Tailored implementations of the SOA stack for mobile devices allow users to consume remote Web services. While as an implementation technology for mobile SOA, adopting the already broadly supported Web services (WS) is justified, we approach the subject of mobile SOA from a different perspective. The reason is that Web service technology has not been successful for mobile devices and did not receive the required support. Instead, vendors have chosen App markets as primary means for distribution of software (Apps). We argue that in order to bring SOA to mobile devices, we need to study their applicability of existing infrastructures like App Stores and mobile Apps in a SOA context.

In this context, we tackle the challenges found in mobility-enhanced service-orientation from three perspectives:

– First, we analyze existing SOA infrastructure concepts (e.g., Service registries, Service brokers, Service providers) with regard to their counterparts in the mobile domain (e.g., App stores, Apps). We establish a conceptual mapping between these two infrastructures to provide the foundation for a lightweight composition and communication approach.

M. Treiber, D. Schall, S. Dustdar
Distributed Systems Group, TU Vienna
E-mail: {lastname}@infosys.tuwien.ac.at

C. Scherling
ikangai solutions
E-mail: office@ikangai.com

[1] http://blog.nielsen.com/nielsenwire/online_mobile/the-state-of-mobile-apps/

– Second, we introduce a lightweight programming language called *Tweetflows*[2] that provides the communication mechanisms to invoke Apps remotely and consequently the means to compose Apps. In this regard, we study the application of Twitter and short text messages as communication means using mobile services in a *social context*.
– Finally, as a cross cutting concern, we investigate the Human-Provided Services [38] ideas for the provisioning of human expertise in the context of our proposed framework. In particular, we investigate *crowdsourcing* aspects for the discovery and provision of mobile services in social networks of mobile users.

The three conceptual pillars provide us with the foundation for an infrastructure concept that will allow us to exploit social aspects (follower or friend networks of users) for mobile SOA.

Paper structure. Section 2 introduces a small application scenario and highlights the challenges for mobile, App-based service provision. We introduce key design principles in Section 3. In Section 4, we discuss the application of SOA concepts in the context of mobile App-based Service provision. After establishing the conceptual foundation of our approach, we show in Section 5 how to use Tweets and short text messaging - so called Tweetflows - for mobile App-based Service provision. Based on the conceptual framework and Tweetflow communication primitives, we present the architecture of our prototype in Section 6 and illustrate the use of our prototype in Section 7. We conclude the paper with related work in Section 8 and an outlook for future work in Section 9.

2 Motivating Scenario

A mobile application is typically used in combination with a Service hosted by an application server. So an App requests its data usually from Services that are available on the Internet. An App is thus not seen as a Service that can be offered but rather as a user front-end to interact with 'technical Services'. We propose to take a different perspective on mobile applications by emphasizing the social context in which Apps are used which shows clearly the service aspect of mobile Applications. The inclusion of the social context (albeit for search purposes) was introduced in [18] as the *village paradigm*. I. In villages, knowledge dissemination is achieved socially: information is passed from one person to another person, and eventually the right person is found who is able to answer a particular question. We follow the same paradigm, but focus on mobility aspects. In order to show how the village paradigm is applied in the context of mobile Apps, we discuss two versions of the same scenario. In the scenario, a person (A) needs information about good restaurants located nearby and asks some other person (B). A typical conversation would look like this:

– **Person A:** Do you know good restaurants nearby?
– **Person B:** Well, let me think . . . I've an App on my mobile that lists all restaurants in the vicinity. (Starts App, looks at the App)
– **Person B:** Here, look! I just found this. It's on...

If we abstract from the scenario, we can observe the following roles. Person A acts as Service requestor. The Service request is to find a *good* restaurant and the corresponding information Service is offered by Person B, or B's App, respectively. The observed conversation pattern is peer to peer: Person A asks Person B. The actual access to the Service (the App on B's mobile device) is mediated by Person B, thus Person B also acts as Service Broker.

In a variant of the scenario above, the question is posed to several persons at once. These persons are not in direct physical reach, thus a communication means to distribute a Service request to a (potentially mobile) group of persons is required. This leads to the following situation:

– **Person A:** Writes a text message to several persons (B, C, D and E) Do you know a good restaurant nearby? I'm in Museumsquartier.
– **Person B - Calling A:** Yes, I'm just sitting in a nice place. It's on Gasser street in 1st district. Take U2 to Karlsplatz (exit Musikverein), then the first on the right.
– **Person C - Texting A:** I know this place on Kirchengasse. It's called the Blue Banana.
– **Person D - Texting A:** The Blue Banana. I've heard that's really good.
– **Person E - Calling A:** I've just called Person F and asked him. He told me that there is this new Italian restaurant called Fratelli.

Abstracting from this version of the scenario, we can find similar roles like in the first version. Again, Person A acts as Service requestor, requesting Information (good restaurant) from several Service providers (Person B, C, D and E). We can also identify Services that are present both versions. Person C, Person D play the role of recommendation Services and Person B provides a routing Service, while Person E forwards the initial Service request to another Service (Person F) which

[2] `http://www.ikangai.com/blog/tag/tweetflows`

provides the desired information. In both scenarios, the social context in which the Services are provided plays a key role: Person A knows Person B, C, D and E. In addition, Person E forwards the request to another Person F, extending the range of the initial request. The presented versions of the scenario highlight several aspects of mobile Service provision in a social environment. We will discuss these aspects in the following section where we will introduce the main concepts of our approach.

3 Key Design Principles

The idea of using crowds of people for dedicated purposes (e.g., estimation of weight of physical items) has been discussed by Surowiecki [41]. The term crowdsourcing has been employed later by Howe [19]. In mobile environments, there are approaches like [30] that investigate mobility in a crowdsourcing setting. We divide the concerns of mobile service provisioning in a crowdsourced environment into four (conceptual) layers, as depicted in Figure 1.

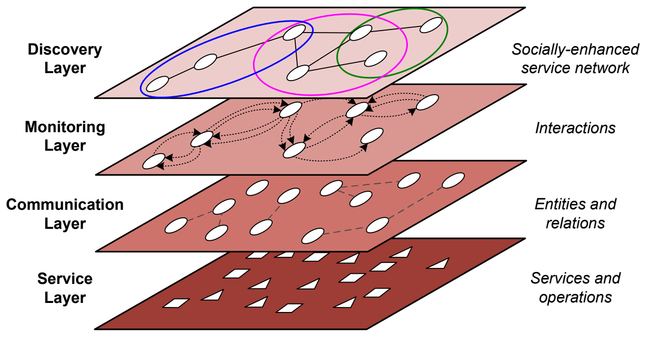


Fig. 1 Crowdsourcing layers.

– **Service Layer.** The bottom layer of our conceptual architecture consists of mobile Services which can be provided either by humans or software. Software-based Services are embodied as Apps which are hosted on mobile devices. Both human-provided [38] and software services, comprise the functional underpinning for our proposed mobile crowdsourcing infrastructure. By treating humans and Services alike, we lay the foundations for the creation of complex, mobile SOAs which encompass social aspects..
– **Communication Layer.** Set on top of the Service layer, we position the communication layer. As the name implies, the communication layer handles all communication between human- and software-based Services. Taking mobility into account, this layer abstracts from the actual communication means, while providing the necessary features to cope with obstacles that derive from mobility: loss of connectivity, decoupling of messages and different communication means. To provide a loosely coupled communication system, we use a (logically) centralized communication bus which handles the communication between Services and humans in a transparent, technology-agnostic manner. We provide for a lightweight abstraction for service interactions that is both human- and machine-readable. This communication approach enables the transparent access from various (mobile) platforms, without the need for installing complex tools. For a structured communication we foresee a set of communication primitives that structure messages and provide meta information like addressing or classification.
– **Monitoring Layer.** The monitoring layer extracts all interactions between Services and humans which are created during the execution of various tasks. The main task of the monitoring layer is to filter, weight and organize Service interactions. For example, as illustrated in working example, the Service requestor needs to filter multiple responses to a service request, in order to select the appropriate Service.
– **Discovery Layer.** The discovery of Services is based on a combination of centralized and localized repositories. The latter emerge from social network structures around Service requestors, in which Service discovery is supported by a context-sensitive query that is based on contextual information such as topic tags is performed to discover human and software Services (the detailed mechanism can be found in [37]). Simultaneously, by forwarding messages to other network members we achieve a distribution of discovery requests.

4 Applying SOA on Mobile Apps

The central paradigm of SOA is the triangle relation between Service registry, Service provider and Service requestor. The underlying principle can be summarized as publish-find-bind-execute cycle which offers flexible solutions with respect to manageability and adaptivity.

Mobility requires a modified view on the traditional SOA triangle. Figure 2 shows the entities which constitute our proposed mobile SOA in crowd sourcing settings and shows the interactions between these entities. We can see still find the Service requester, that interacts with the provider (crowd), and binds to a Service (App) in the crowd. The discovery is sup- ported by the App store (registry) which also serves as distribution channel. However, additional considerations are necessary when implementing this principle on mobile devices in a crowd sourcing context. First of all, we need to find a mapping between existing SOA artifacts like Services, Service registries and Service bindings to the available mobile infrastructures. We discuss this in sections 4.1 and 4.2 where we show how Apps can play the role of (mobile) Services regarded and explain the role of App stores as registries.

Second of all, we will investigate *crowds* with regard to the application SOA principles. In this respect, we consider a network of mobile App users as App provider Network. We will discuss this in section 4.3 where we explain the binding process of mobile Apps in the provider network.

4.1 The App as a Service

In the Service world, Service are considered as self- contained, independent entities, which offer a certain functionality and do not depend on the context or state of other Services [4]. These principles are the foundation for the creation of complex Service-based software systems. We argue that by applying SOA principles on mobile Apps, we can provide a powerful abstraction which allows the creation of complex, mobile Service-oriented software systems. In particular, we regard mobile Apps as Services because they offer well-defined functionality, are self-contained and do not depend on the state of other Apps. Similar to Services, Apps can be discovered in App repositories (App Stores) and can be accessed through well-defined (graphical) interfaces. Table 1 gives an overview of the characteristics of Services and *Apps as a Service.*

There are several aspects of mobile Apps that need additional considerations. First of all, Apps can serve as access points to remote (centralized) Services, providing the interface or the access point to the actual Service. Consider the example of a weather forecast App: the App connects to a central server that hosts a Restful Web service providing weather information. In such a case, the creation of an additional App instance does not lead to a replication of the Service, which remains in the backend. Simply put: the Service provider, albeit hidden by the App interface, stays the same and only the Service access point is replicated. If the App does not access a remote Service, replicating the App by downloading it to another mobile device has a different effect. Take for example a mobile information App with a local database containing touristic information. If this App is downloaded to another mobile device, the Service provider is also replicated.

We refer to Service replication as shallow replication or endpoint replication when the replicated App acts as an access point to a remote Service. Deep replication replicates the Service provider as well, providing additional service capacities and adding fault tolerance and availability in crowds. Viewed from a user's perspective, there is no notable difference between these two replication approaches. Both cases lead to additional endpoints of the Service (App) in the crowd. In both cases, the consequence of the replication is that more users in the crowd are able to provide the Service and the likelihood of discovering a Service (App) increases.

Another aspect is the possibility for providers to dynamically extend available Services. In contrast to SOA Service providers, mobile Apps can be installed on demand. Thus Service (App) providers are able to dynamically modify their Service offerings: if an incoming request requires a certain functionality that cannot

Characteristic	App	Service
Execution	Local	Remote
Crowd Replication	Yes	No
Meta Data	Human readable descriptions, Pre defined App Categories	Interface descriptions, Ontologies
Well Known Address	Yes	Yes
Dynamic Binding	Yes	Yes
Interface Descriptions	None	Yes, different standards
Human readable information	Yes	Yes, available
Automated Discovery	Yes, Web Portals	Yes, supported
Distribution	Yes	No
GUI	Yes	No

Table 1 Comparison of Apps and Services.

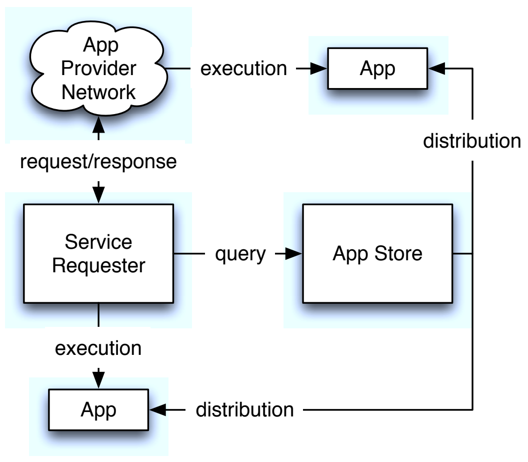


Fig. 2 SOA with mobile Apps.

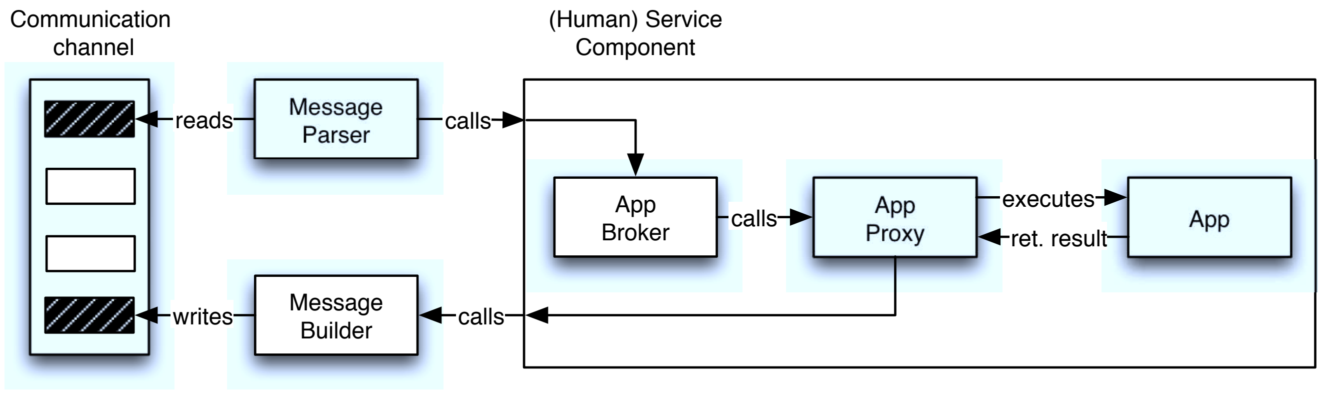


Fig. 3 Conceptual Architecture of Apps as a Service.

be provided, the corresponding App can be discovered in the App store and installed to fulfill the request.

Since Apps do not offer standardized interface descriptions similar to (Web) Services, we hide the interface of Apps behind a proxy architecture that forwards the requests to the App and returns the result to the requestor (see Figure 3). Note that in principle, all components - except the App itself - (Message Parser, App Broker, Message Builder, App Proxy) can be either implemented by a human, i.e., the user of the mobile device, or by software. However, mobile phones are personal devices and require the interaction with the owner of the device. It is the owner who effectively decides if a Service request leads to an invocation of an App. As illustrated in the motivating scenario, it is the user who receives a request, executes the App and returns the result to the Service requestor, e.g., by using his voice. This is a very flexible way of interaction, because humans are capable of mediating between different message formats or different languages (e.g., English, Japanese). The tradeoff of this approach is the active involvement of the user. Obviously, users are a potential bottleneck for the messaging system, because of the limitation in terms of message processing. However, at the same time, users are capable to make informed decisions and to react in an intelligent manner to Service requests. For example, as stated above, if a service request requires an App which is not available on the mobile device, the user can actively search for an App, install it and provide the required service functionality.

4.2 App Store as a Registry

Until not too long ago, Web service registries [10] like UDDI [9] served as a universal and centralized repository for Service related information (e.g., endpoints, descriptions). However, after IBM, Microsoft and SAP abandoned their public UDDI registries in 2006 [28], there are virtually no public registries available that offer a considerable set of Service information to customers. The only exceptions are registries like Seekda[3], or academic prototypes (e.g., AWSR [42] or [27]), though the latter are not available for broader public use. The success of App Stores like the Android Store and the iTunes App Store suggests that centralized repositories work, albeit in a slightly different manner than Service registries. In contrast to Service registries, access to the App Store is realized by dedicated App Store Apps on mobile devices, making the Service consumer implicitly aware of the location of the App Store. Furthermore, App Stores typically provide a pre-defined set of categories that classify the Apps in a broad sense. Unlike Service registries, App Stores do not provide mechanisms to include rich semantic meta information like ontologies to discover Apps automatically; the selection process relies on human judgement. For example, the iTunes App Store offers 20 Categories (e.g., Productivity, Entertainment, News) and does not offer subcategories for a hierarchical drill-down refinement during the discovery process. Like Service registries, the search process is supported by keywords which classify the App.

App stores also provide a recommendation system - like the one of the iTunes Store (e.g., new and noteworthy, staff picks, top selling, top grossing) - and the ability to sort the presented Apps according their popularity or creation date. Complementary to Apps stores, Web portals exist that access the App store data, and these portals offer links to the Apps in the App store[4]. These portals offer additional information like community ratings, App rankings of reviewers or App reviews to browse trough, being conceptually close to Web service registries.

[3] `http://webservices.seekda.com/`
[4] `http://www.androidpit.com/`

Characteristic	App store	Service registry
Discovery mechanisms	Keywords, Classifications	Keywords, Software Assisted Reasoning
Meta Data	Human readable descriptions, Pre defined App Categories	Interface descriptions, Ontologies
Well Known Address	Yes	Yes
Dynamic Binding	Yes	Yes
Interface Descriptions	None	Yes, different standards
Human readable information	Yes	Yes, available
Automated Discovery	Yes, Web Portals	Yes, supported
Distribution	Yes	No

Table 2 App stores versus Service registries

These portals also provide additional information like community ratings, App rankings of reviewers or App reviews to browse trough, being conceptually close to Service registries. A key difference between Service registries and App stores is the fact that App stores actually provide the App itself. Customers can download and install the App directly from the App store. In contrast, Service registries provide links to Services and do not provide direct access to the Services itself. Thus, App stores play an active role in the distribution of Apps, whereas Service registries offer information for accessing a Service and provide no means for the actual distribution of Services. We have summarized the discussion in Table 2 where provide an overview of Registry and App store characteristics.

4.3 App Binding

In SOA, the discovery and binding of Services is typically supported by a (centralized) repository which provides information about the Service and the used protocol. Critical is the access to interface and binding information which is required to generate stubs for the invocation of remote Services. This kind of information is often presented as WSDL [8] which is a well adopted standard designed to represent Service interface and binding information. However, in recent years, alternative Service descriptions or extensions to WSDL were proposed. For example, Services that follow the REST paradigm were described with WADL [17], semantic enriched descriptions (among them descriptions such as WSDL-S [2], WSML [13], SAWSDL [12]) were proposed to add meta information to make Services automatically discoverable and to provide the means for automatic Service selection during the Service binding process. A common challenge for all previously mentioned approaches is that the Service provider is not known in advance. Thus, the choice of Service to bind to must be done at runtime. This requires the use of a Service broker that is able to map Service requests to concrete service endpoints. Basically, the algorithm works in three steps: (i) Select candidate Services, (ii) rank the candidates in a particular order, and (iii) select the most appropriate Service to bind to. After the binding process is completed, the invocation of the Service takes place.

In mobile crowdsourcing environments, complex descriptions of Services (i.e., Service meta data) or Service invocation mechanisms (e.g., SOAP Service calls) are not available. Thus, the process of Service binding needs to consider the role users in the App provider network. Generally speaking, the binding of Apps as Services is user-centered and requires the expertise of the App provider network user for the binding of Apps to Service requests. As discussed in the motivating scenario, when a user is asked to provide information about a restaurant, the user determines if an App is available (candidate selection and ranking) that can be used to fulfill this kind of inquiry (Service selection and binding). If not, the user can decide to forward the request to others. In this case, the binding process is extended with an additional crowd-sourced - discovery step: the request is forwarded into the App provider network in which the actual App (Service)binding takes place. Alternatively, the user can either (1) recommend an App to the service requester to install or (2) install an App himself to provide the required Service. The former can be considered as Service brokerage - the user recommends an App and the Service requester can install the App to access the Service. The latter is related to dynamic Service provision; the Service provider is able to ad hoc add Apps (Services) to extend the variety of provided Services.

5 Mobile Communication

After establishing a mapping of existing mobile infrastructures (App, App stores) to SOA infrastructure concepts (Services, Service registries) in the previous sections, we now need to address the communication issues of mobile App (Service) provision. One of the benefits of SOA is the ability to compose new Services from existing Services with the help of composition languages like BPEL [1], YAWL [43]. However, we do not strive to provide a complex, full-fledged composition language, since the applicability of complex messaging on mobile devices is limited. Instead, we focus on a lightweight language that supports the communication between mobile

App (Service) requesters and App (Service) providers. By exploiting social aspects on an architectural level of (see Section 4.1) we require the language to have as little as possible message overhead and to retain a certain degree of human readability. The latter is important, since we explicitly exploit social aspects of the App (Service) requestors.

With regard to our conceptual crowdsourcing layers as introduced in Figure 1 we address the aforementioned requirements with a new light-weight flow language called *Tweetflows*. The goal of Tweetflows is to support concepts found in each layer: (i) Service layer by letting people provide Services in (mobile) crowdsourcing environments, (ii) communication layer using structures and profiles in social networks (e.g., Twitter follower graph to retweet requests), (iii) monitoring of interactions through a message bus, and (iv) discovery layer to find appropriate crowd members (offering Services) that can, for example, answer specific questions.

From a technical point of view, Services communicate primarily over protocols such as HTTP, exchanging for example messages via HTTP GET/POST requests. Generally speaking, Service-centric architectures do not dictate any kind of communication infrastructure; its possible to exchange Service messages over other protocols, like SMTP, as well. In principle, mobile devices are capable of providing HTTP communication required for Services.

In our approach, we focus on text based communication using HTTP messaging and text messages for the exchange of Service messages. Our intent is to create a simple, human readable language that addresses the aforementioned requirements. Furthermore, in order to address the social aspect of users, we use the Twitter follower structure of users. We use Twitter communication (Tweets) as medium for Service messaging.

Conceptually, the communication and message exchange in Twitter follows a publish/subscribe pattern [11]. 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 [29]. When using such communication means, the limited length of Tweets and text messages demands for a specification of a compact syntax to enable communications and control of mobile Services (embodied by mobile Apps). We address this challenge with the introduction of a set of Twitter communication primitives that enable a seamless fabric of human and App (Service) interactions. This not only allows users to request a particular App, but also facilitates the discovery of Services in crowdsourcing environments. Table 3 shows the most essential Tweetflow primitives that have been devised for Tweeflows. In our further discussions we will establish the correspondence of Tweetflow primitives and concepts that we have adopted from SOA and applied to mobile Apps:

– By using Twitter as 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 microblogging service, limits the amount of data 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 [20].
– 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 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 built-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.
– We provide for basic Services compositions that are described by unix pipe inspired syntax. Service compositions, Service choreographies respectively, are directly embedded into Tweets. In such a setting the responsibility of handling the execution flow of a Service composition is on Service providers side.

Abbreviation	Description
SR	Service Request
RE	Service Response
RT	Retweet Service Request
DS	Delegate Service Request
TF	Tweetflow Compositions
RJ	Reject Service Request
ST	Service State Request
SE	Service State Reply
SP	Service Announcement

Table 3 Tweetflow primitives.

5.1 Tweetflow Primitives

In this section, we detail communication principles using Tweetflow primitives and provide simple examples.

Passing Data to and from Services. The limitation of Twitter messages requires special considerations concerning the access of input and output data for Services. We consider two possible ways of passing data to Services with Twitter. We use standard URL-encoding to pass data inline, i.e., the Tweet contains all data that is passed to the Service. Note that inline data is limited to 140 characters, due the size limitation of Tweets. To overcome the size limitation, we use 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.

```
SR doTranslate.Japanese http://bit.ly/9qFRGL
#English #Japanese #Translation
```

Listing 1 Passing data to a translation Service.

Announcing Services with Tweetflows. 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 as well as Twitter hashtags (see Listing 2). We consider hashtags to play a similar role like tags in folksonomies [47]; a distributed, bottom up classification schema for Services which are made available over Twitter. In addition, the retweet mechanism allows to spread Service announcements over the Twitter network [24] providing for a social network based Service publication.

```
SP <operation>.<servicename> <url>
<hashtags>
```

Listing 2 Announcing services with Tweetflows.

Discovering Services with Tweetflows. The discovery of services in Twitter does not follow the pull approach as with existing SOAs where a Service requester searches for candidate Services in Service registries. Instead, Tweets are posted that describe the required Service and provide the capabilities to bind and execute Services (see Listing 3).

– *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. This includes optional information about the operation that is required, hashtags describing the Service with keywords or a link to Service input data.
– *Direct Service Request:* The Service request can also be directed toward Twitter followers. In this case, the Tweet addresses the user directly and binds the Service to the Service provider.
– *Delegation of Service Request:* If Twitter followers are not able to handle the Service request, but happen to know someone able to provide the Service, 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 not 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.

```
// Publication of a Service
SR <operation>.<servicename>
(<url>|<data>) <hashtags>

// Direct Service request
SR @<user><operation>.<servicename>
(<url>|<data>) <hashtags>

// Delegation of a Service request
DS @<user><operation>.<servicename>
(<url>|<data>) <hashtags>

// Spreading a Service request
RT @<user><operation>.<servicename>
(<url>|<data>) <hashtags>
```

Listing 3 Syntax elements related to discovery.

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 invoked. The actual addressing of the Services uses the built-in Twitter addressing mechanisms which sends Tweets directly to followers.

```
SP@<user> <operation>.<servicename> <url>
<hashtags>
```

Listing 4 Claiming a Service with Tweetflows.

It is worth noting that claiming of a Service uses a syntax that close the syntax for the publication of Services. The reasoning behind this is that we consider the Service claim as directed Service publication. More specifically, we bind directly to an operation.

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:* 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's also possible to reject a Service request from another user.

```
// Service Response on Twitter
RE @<user> <operation>.<servicename>
(<url>|<data>)

// Service State Request on Twitter
ST @<user> <operation>.<servicename>

// Service State Reply on Twitter
SE @<user> <operation>.<servicename>
<state>

// Service Request Reject on Twitter
RJ @<user> <operation>.<servicename>
```

Listing 5 Syntax elements related to execution.

Bootstrapping and Composing Tweetflows. The process of creating or bootstrapping a Tweetflow consists of tweeting a set of Service requests to find adequate Service providers. As discussed in previous sections, the originator, i.e. the coordinator, of the Tweetflow posts one or more Tweets that contains set of actions which need to be executed by different users. This Tweet marks the beginning of the execution of a Tweetflow.

Tweetflows do support several means to structure the execution of Tweets. The current version of the Tweetflow syntax provides for two basic means to coordinate and structure the execution of Tweetflows.

Similar to unix shells, Tweetflows can be structured with *pipes*, leading to a sequence of Service (App) invocations which pass the output from one invocation to the other. Listing 6 shows the syntax of Tweetflow Service pipes.

```
TF <name>[@<user> <operation>.<servicename>
(<url>|<data>) | @<user> <operation>.<servicename>
(<url>|<data>)]
```

Listing 6 Service pipes on Twitter.

Related Service pipes, are so called *open sequences.* Open sequences are consecutive Tweets that are part of one Tweetflow which do not impose a particular ordering of the execution of the Services (Apps). Listing 7 shows the syntax of open sequences.

```
TF <name> @<user> <operation>.<servicename>
(<url>|<data>)
TF <name> @<user> <operation>.<servicename>
(<url>|<data>)
...
```

Listing 7 Open sequences on Twitter.

It is worth noting that the control over the execution is distributed [6]: upon completion of a Service in the service pipe, 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 is possible to make modifications to the Tweetflow during the execution. For example, a Service request can be delegated to another Twitter follower upon receiving a request, allowing for Service replacements on the fly. In order to track the execution of Tweetflows, we require each Tweetflow Tweet to contain a hashtag with the name of the Tweetflow as shown in Listing 6 and Listing 7.

5.2 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 Service 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 4.

Tweetflow Primitives	SOA Concept
SR, RT, DS, RJ	Service Discovery
SP	Service Binding
RE	Service Response
TF, [], —	Service Composition
ST, SE	Service Monitoring
SP	Service Publication
@	Service Addressing

Table 4 Mapping SOA principles to Twitter.

6 Architecture

On a conceptual level, our proposed architecture (the architecture overview is shown in Figure 4) follows prin-

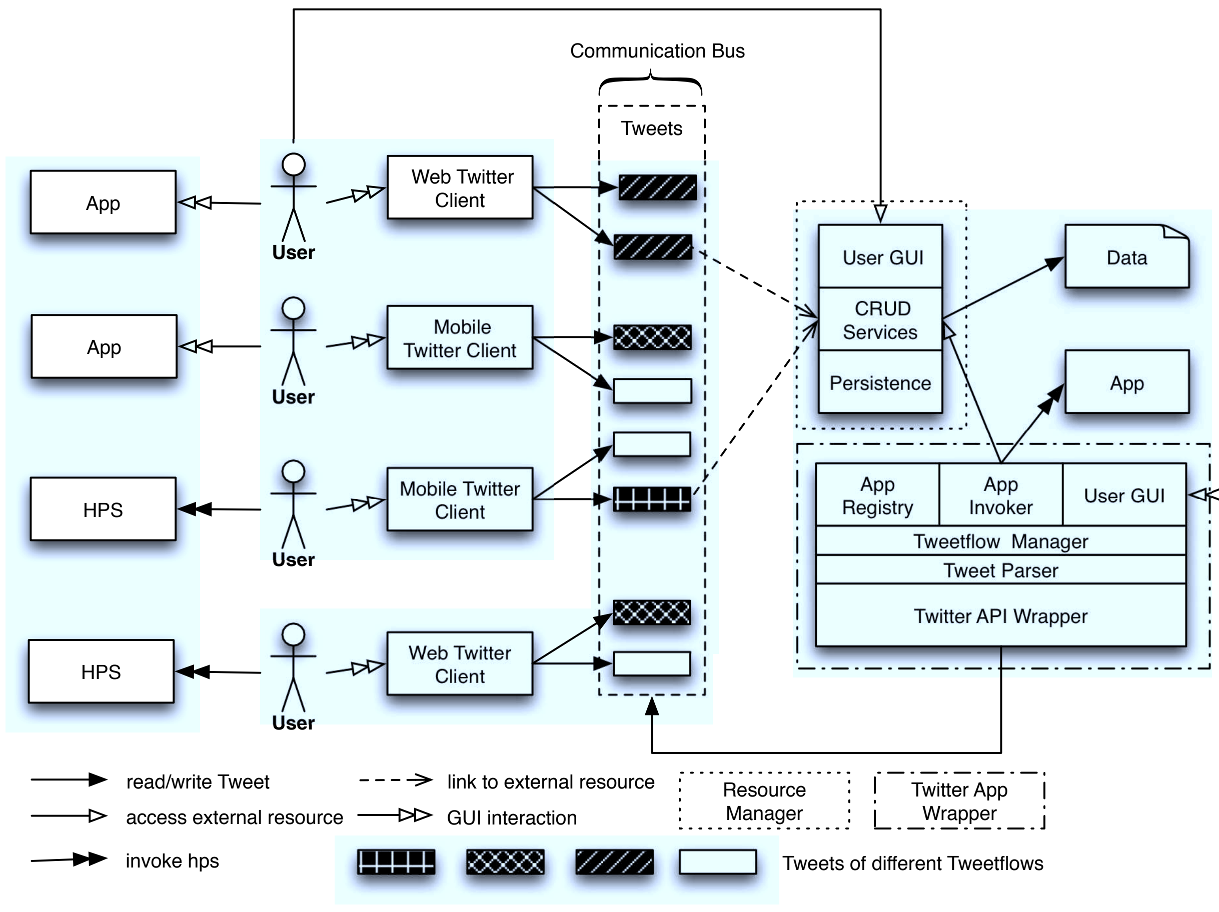


Fig. 4 Tweetflow architecture for collaborative crowds.

ciples that can be found in ESBs (Enterprise Service Buses), e.g., see open source projects such as ServiceMix[5]. Like ESBs, which provide an abstraction layer on top of an implementation of an enterprise messaging system, our architecture introduces an abstraction layer over existing messaging systems. Having one common communication channel, we are able to plug Services, Apps respectively into a common communication infrastructure. By taking social aspects into account, we also provide for a lightweight directed multicast communication system.

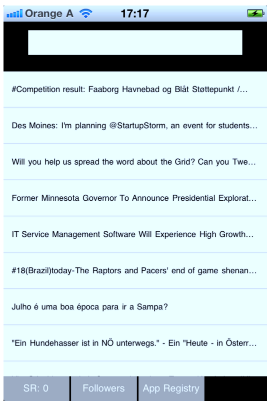
As shown in the overview, our architecture consists of several components which are arranged around a bus-like infrastructure providing the central communication means. Users interact with the Tweet Bus using different clients from different platforms (e.g., third party Twitter clients from mobile devices [15] or the Twitter Web page). The Tweet Bus itself contains several Tweetflows simultaneously, thus Tweetflows are intertwined and the respective Tweets are arranged according to their timestamp. Consequently, we require the client to filter Tweets, which our prototype implementation does with the help of regular expressions. The filtered Tweets are sorted into respective Tweetflow queues in the Tweetflow manager component. Each queue works according to the FIFO principle; our current implementation does not provide for priority ordering of Tweets. The access of the mobile App is handled by the App invoker component which executes the App.

Currently, we do not support fully automated App binding and require the user to manually bind a App. Note that the binding of an App can be temporary: App bindings can be assigned to different Tweetflows which are identified by their names or hashtags. It is possible to change the binding of an App during the execution of a Tweetflow, thus providing for dynamic App bindings. Figure 5 shows screenshots of the Tweetflow prototype App that is able to handle Tweetflow requests.

Optional data which can be passed to the App is stored by an external Service that provides the means to create, read, delete and update simple (text) resources[6]. These resources are referenced from Tweets and comprise (optional or the only) input for Apps

[5] `http://servicemix.apache.org/home.html`

[6] `http://www.infosys.tuwien.ac.at/staff/treiber/wwwdemo/textwriter.html`



#Competition result: Faaborg Havnebad og Blåt Støttepunkt /...

Des Moines: I'm planning @StartupStorm, an event for students...

Will you help us spread the word about the Grid? Can you Twe...

Former Minnesota Governor To Announce Presidential Explorat...

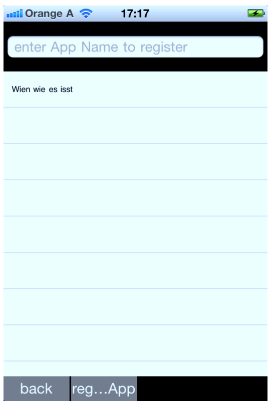
IT Service Management Software Will Experience High Growth...

#18(Brazil)today-The Raptors and Pacers' end of game shenan...

Julho é uma boa época para ir a Sampa?

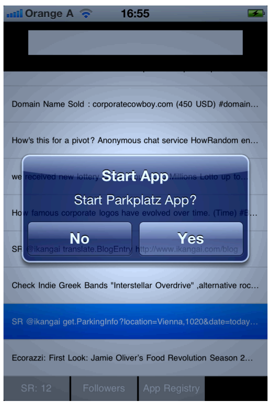
"Ein Hundehasser ist in NÖ unterwegs." - Ein "Heute - in Österr...

(a) Tweetflow App screen.



Wien wie es isst

(b) Registering an App.



Domain Name Sold : corporatecowboy.com (450 USD) #domain...

How's this for a pivot? Anonymous chat service HowRandom en...

we received new lottery ... Millions Lotto up to...

How famous corporate logos have evolved over time. (Time) #B...

SR @ikangai translate.BlogEntry http://www.ikangai.com/blog

Check Indie Greek Bands "Interstellar Overdrive" ,alternative roc...

SR @ikangai get.ParkingInfo?location=Vienna,1020&date=today...

Ecorazzi: First Look: Jamie Oliver's Food Revolution Season 2...

(c) Execution of an App.

Fig. 5 Screenshots of Tweetflow Prototype App.

that are invoked by Tweets. As shown in the architecture overview, Tweetflows are per se not bound to a particular software infrastructure on mobile devices. Tweetflows can also be accessed from third party clients (e.g., mobile Twitter clients) or even from Web pages that are accessed from the mobile device.

7 Application Scenario

In this section, we illustrate the application of our proposed approach in a real world scenario derived from our motivation example. Consider a person who want to go out for dinner after work with two friends. This person does not have specific requirements concerning the type of restaurant, but owns a car and thus need a parking place. The corresponding *open Tweeflow sequence* for requesting car parking information, restaurant information and the availability for dinner is shown in Listing 8.

```
TF didBegin.Tweetflow reservation
#dinner #restaurant #carpark
TF reservation SR @jumpne recommend.restaurant?
location=Vienna,1040&date=today&time=20:00
TF reservation SR @wokung recommend.restaurant?
location=Vienna,1040&date=today&time=20:00
TF reservation SR @ikangai get.ParkingInfo?
location=Vienna,1040&date=today&time=20:00
TF reservation SR @johannes2112 get.Availability?
location=Vienna,1040&date=today&time=20:00
TF reservation SR @redali75 get.Availability?
location=Vienna,1040&date=today&time=20:00
TF didFinish.Tweetflow reservation
```

Listing 8 Requesting a restaurant recommendation

After a while, several Twitter followers answer to the request. Some of the followers have an App installed containing a database of restaurant recommendations[7]. After receiving the request, the user opens the App (which has been registered for this purpose) looks for a restaurant and tweets the result (see Figure 6).

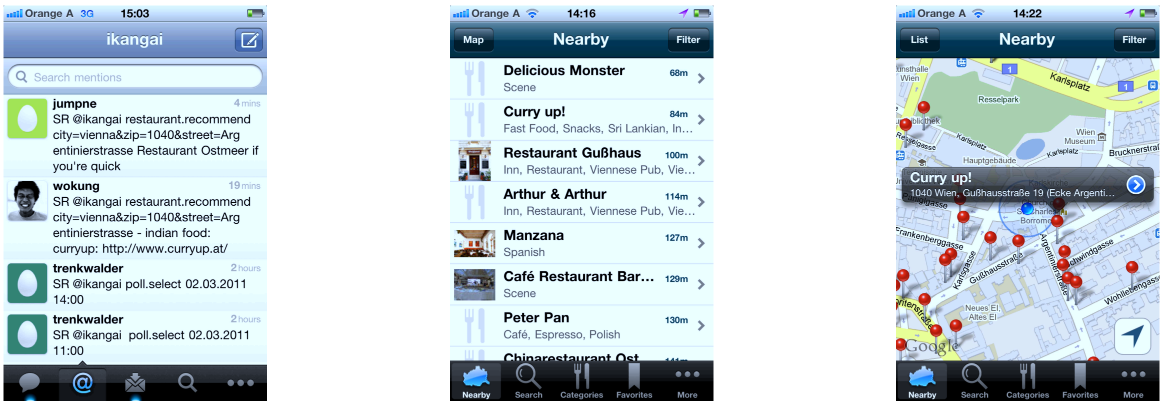
[7] Wien wie es isst.

It is worth noting that the actual answer tweet did not conform to the exact syntax we have been discussing in this paper. The reason was that the users did not have the prototype Twitter client installed which provides a graphical user interface and supports the creation of Tweetflow-compliant Tweets. Furthermore, the address in the request contained a typo; the correct street name is *Argentinierstrasse* and not as mistakenly written in the Service request Tweet *Argentierstrasse*. However, the flexibility of human proxies can handle this kind of syntactic ambiguities easily and provide a correct answer nevertheless.

Another issue concerns the aggregation of multiple answers. This is a typical crowdsourcing problem, since a service requester can expect several answers for crowdsourced requests. In our current prototype implementation, the service requester receives a list of results from which the requester can choose manually. A future extension of the approach could be to establish algorithms that automatically select the *best* response (for example, by considering the reputation of the users that provided the response). The integration of reputation mechanism has been briefly discussed in our conceptual stack (see Figure 1). At this stage, we have started to integrate various techniques based on social network analysis methods.

8 Related Work

In Service-oriented environments, standards have been established to model human-based process activities and tasks (WS-HumanTask [14]). However, these standards require the precise definition of interaction models between humans and Services. In our approach, we com-



Orange A 3G 15:03

ikangai

Search mentions

jumpne 4 mins
SR @ikangai restaurant.recommend city=vienna&zip=1040&street=Arg entinierstrasse Restaurant Ostmeer if you're quick

wokung 19 mins
SR @ikangai restaurant.recommend city=vienna&zip=1040&street=Arg entinierstrasse - indian food: curryup: http://www.curryup.at/

trenkwalder 2 hours
SR @ikangai poll.select 02.03.2011 14:00

trenkwalder 2 hours
SR @ikangai poll.select 02.03.2011 11:00

Orange A 14:16

Map | Nearby | Filter

Delicious Monster Scene	68m	>
Curry up! Fast Food, Snacks, Sri Lankian, In…	84m	>
Restaurant Gußhaus Inn, Restaurant, Viennese Pub, Vie…	100m	>
Arthur & Arthur Inn, Restaurant, Viennese Pub, Vie…	114m	>
Manzana Spanish	127m	>
Café Restaurant Bar… Scene	129m	>
Peter Pan Café, Espresso, Polish	130m	>
Chinarestaurant Ost		

Nearby | Search | Categories | Favorites | More

Curry up!
1040 Wien, Gußhausstraße 19 (Ecke Argenti…

(a) Tweetflow screen. (b) Nearby restaurants. (c) Recommendation.

Fig. 6 Screenshots of mobile Tweetflow App.

bine 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) [38] supports flexible Service-oriented collaborations across multiple organizations and domains. Similarly, emergent collectives as defined by [36] 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 [39], but without focusing on the realization as a Service-based systems. While existing platforms (e.g., MTurk [5]) 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 introduced by [46] in the context of image labeling that is performed by humans.in the context of image labeling performed by humans. From the technical point of view, TurKit [25] is a crowd-computing framework based on MTurk. The availability of rich and plentiful data on human interaction in *social networks* has closed an important loop [23], allowing to model social phenomena and use these models in the design of new computing applications.

Semantic Web service communities as introduced by [26] foster the creation of structured communities with predefined community interfaces and functionality. However, ontology structures are not well suited for crowds, because crowd structures emerge bottom up and are difficult to capture with regard to functionality and interactions between crowd members. Also, value networks [3] are of interest when business aspects are investigated in crowd settings, i.e., the value that can be generated by such networks based on crowds.

On an architectural level, we follow the blackboard architectural pattern [16] with a shared space for the exchange of information. 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 [7] have a strong similarity to our proposed 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 public visibility of Tweets, the 140 character limit for messages and the ability to forward (retweet) messages arbitrarily to other users that do not listen to the communication bus, i.e., to push messages into an unstructured community of followers are the main differences.

Principles of SOA are well studied in literature [35]. However, the move of SOA into the mobile domain focuses mainly on implementation details and address limitations of mobile devices. Juszczyk et al present [22] present a middleware for Service-oriented communication which run on mobile devices. Mobile Web services Architectures [32] aim at providing alternative representations other than XML-based SOAP and fast communication transport options for mobile Web services [33] [21]. Other approaches use aspect-oriented programming to facilitate the access to Services from mobile devices [34]. The work presented in [45] describes an infrastructure and middleware design which is based on the Jini Surrogate Architecture Specification. Web services for Devices[8] aim at porting as much of the SOA stack as possible to embedded and mobile devices and make use of gSOAP implementation [44]. The authors

[8] `http://www.ws4d.org/`

of [31] present a lightweight mobile SOA-based architecture based on J2ME and aim at minimizing the traffic to and from mobile devices. [40] investigates the use of short messages as communication mean between mobile devices to invoke Services asynchronously.

9 Summary and Outlook

We have presented an approach for the application of SOA principles on mobile Apps. Specifically, we have investigated the mapping of existing infrastructure (Apps, App Store) to SOA concepts like Service, Service registry and Service consumer. After establishing the mapping, we have introduced a lightweight communication schema (Tweetflows) that uses social network structures and provides for the integration of human-provided services.

Our next steps will be the extension of the communication schema to support more complex processes and the investigation of the distributed execution of Tweetflows in crowd scenarios. In particular, we are going to define Loops and conditional expressions within Tweetflows. Another area of future work is the implementation of intelligent message forwarding mechanisms for the crowd. If we want to move our approach beyond the personal character of mobile Service, we need to consider the crowd as a whole as a Service provider network. If a message is sent to the crowd, the message needs to be automatically forwarded to other users, without any need for user intervention. To increase the probability of reaching the intended service, we require a directed message forwarding, based on heuristics with Twitter user profile data.

We will investigate the use of private microblogging servers within LANs, addressing some of the privacy issues of Tweetflows. We will look at mechanisms that restrict the access to Tweets on an architectural level by installing local microblogging severs and restricting the access to these servers. In addition, private messages and realms of trust will also be investigated.

Finally, we are going to extend our prototype with a graphical interface to support the creation of complex Tweetflows (Service pipes, closed sequences, conditional expressions) directly on mobile devices.

Acknowledgments

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).

References

1. Business Process Execution Language for Web Services (BPEL), 2003.
2. R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma. Web Services Semantics – WSDL-S, 2005.
3. V. Allee. Reconfiguring the value network. *Journal of Business Strategy*, 21(4), August 2000.
4. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services - Concepts, Architectures and Applications.* Springer, October 2003.
5. Amazon.com. Amazon mechanical turk, last access: 2010. available online: http://www.mturk.com.
6. J. Balasooriya, J. Joshi, S. Prasad, and S. Navathe. Distributed coordination of workflows over web services and their Handheld-Based execution. In *Distributed Computing and Networking*, volume 4904 of *Lecture Notes in Computer Science*, pages 39–53. Springer Berlin / Heidelberg, 2008.
7. D. Chappell. *Enterprise Service Bus.* O'Reilly Media, Inc., 2004.
8. R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) 2.0, 2007.
9. L. Clement, A. Hately, , C. von Riegen, and T. Rogers. UDDI Version 3.0.2, 2004.
10. S. Dustdar and M. Treiber. A view based analysis on web service registries. *Distributed and Parallel Databases*, 18(2):147–171, 2005.
11. 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.
12. J. Farrell and H. Lausen. Semantic Annotations for WSDL and XML Schema, August 2007.
13. D. Fensel, H. Lausen, J. de Bruijn, M. Stollberg, D. Roman, A. Polleres, and J. Domingue. Wsml a language for wsmo. *Enabling Semantic Web Services*, pages 83–99, 2007.
14. M. Ford et al. Web Services Human Task (WS-HumanTask), Version 1.0., 2007.
15. S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt. Micro-blog: sharing and querying content through mobile phones and social participation. In *MobiSys '08*, pages 174–186. ACM, 2008.
16. D. Garlan and M. Shaw. An introduction to software architecture. Technical report, Pittsburgh, PA, USA, 1994.
17. M. Hadley. Web Application Description Language, August 2009.
18. D. Horowitz and S. D. Kamvar. The anatomy of a large-scale social search engine. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 431–440, New York, NY, USA, 2010. ACM.
19. J. Howe. The rise of crowdsourcing, June 2006.
20. J. Huang, K. M. Thornton, and E. N. Efthimiadis. Conversational tagging in twitter. In *HT '10*, pages 173–178. ACM, 2010.
21. F. Jammes, A. Mensch, and H. Smit. Service-oriented device communications using the devices profile for web services. In *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–8, New York, NY, USA, 2005. ACM.
22. L. Juszczyk and S. Dustdar. A middleware for service-oriented communication in mobile disaster response environments. In *MPAC '08*, pages 37–42, New York, NY, USA, 2008. ACM.

23. J. Kleinberg. The convergence of social and technological networks. *Commun. ACM*, 51(11):66–72, 2008.
24. 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.
25. 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.
26. B. Medjahed and A. Bouguettaya. A Dynamic Foundational Architecture for Semantic Web Services. *Distributed and Parallel Databases*, 17(179–206), 2005.
27. A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, and S. Dustdar. Towards recovering the broken soa triangle: a software engineering perspective. In *IW-SOSWE '07: 2nd international workshop on Service oriented software engineering*, pages 22–28, New York, NY, USA, 2007. ACM.
28. Microsoft. Uddi shutdown, 2006.
29. 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.
30. D. G. Murray, E. Yoneki, J. Crowcroft, and S. Hand. The case for crowd computing. In *MobiHeld '10*, pages 39–44. ACM, 2010.
31. Y. Natchetoi, V. Kaufman, and A. Shapiro. Service-oriented architecture for mobile applications. In *SAM '08: Proceedings of the 1st international workshop on Software architectures and mobility*, pages 27–32, New York, NY, USA, 2008. ACM.
32. S. Oh and G. C. Fox. Hhfr: A new architecture for mobile web services: Principles and implementations. Technical report, 2005.
33. S. Oh and G. C. Fox. Optimizing web service messaging performance in mobile computing. *Future Gener. Comput. Syst.*, 23(4):623–632, 2007.
34. G. Ortiz and A. G. D. Prado. Improving device-aware web services and their mobile clients through an aspect-oriented, model-driven approach. *Inf. Softw. Technol.*, 52(10):1080–1093, 2010.
35. M. P. Papazoglou and W.-J. Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.
36. C. Petrie. Plenty of room outside the firm. *IEEE Internet Computing*, 14, 2010.
37. D. Schall and F. Skopik. Mining and composition of emergent collectives in mixed service-oriented systems. In *CEC '10*. IEEE, 2010.
38. D. Schall, H.-L. Truong, and S. Dustdar. Unifying Human and Software Services in Web-Scale Collaborations. *Internet Comp.*, 12(3):62–68, 2008.
39. D. Shahaf and E. Horvitz. Generalized task markets for human and machine computation. In *AAAI*, 2010.
40. R. Singh, S. Mishra, and D. S. Kushwaha. An efficient asynchronous mobile web service framework. *SIGSOFT Softw. Eng. Notes*, 34(6):1–7, 2009.
41. J. Surowiecki. *The Wisdom of Crowds*. Anchor, 2005.
42. M. Treiber and S. Dustdar. Active web service registries. *IEEE Internet Computing*, 11(5):66–71, 2007.
43. 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.
44. R. van Engelen. Code generation techniques for developing light-weight xml web services for embedded devices. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 854–861, New York, NY, USA, 2004. ACM.
45. A. van Halteren and P. Pawar. Mobile service platform: A middleware for nomadic mobile service provisioning. In *Wireless and Mobile Computing, Networking and Communications, 2006. (WiMob'2006). IEEE International Conference on*, pages 292 –299, 19-21 2006.
46. L. von Ahn and L. Dabbish. Designing games with a purpose. *Commun. ACM*, 51(8):58–67, 2008.
47. J. Voss. Tagging, folksonomy & co - renaissance of manual indexing? *CoRR*, abs/cs/0701072, 2007.