Towards a Reusable Infrastructure for Crowdsourcing

Tenshi Hara, Thomas Springer, Klemens Muthmann, Alexander Schill
Computer Networks Group
Technische Universität Dresden, Faculty of Computer Science
Dresden, Germany
{firstname.lastname}@tu-dresden.de

Abstract—**In the course of the last few years crowdsourcing has received growing research focus due to its conception of solving complex tasks with the help of a flexible group of contributors of whom each needs to only contribute a simpler task part. Hence, the crowd can contribute by collecting data from distributed locations, completing map information, or voting on product ideas, et cetera. However, even though it is a necessary conceptual feature, the participation of large numbers of users with heterogeneous devices, generic infrastructures for crowdsourcing can be hardly found. For example, the management of users, mobile devices and contributed data has to be repetitively implemented in new projects. To ease the development of crowdsourcing applications, in this paper we propose a generic platform for simplified crowdsourcing deployment while supporting diverse crowdsourcing scenarios, the ability to handle large numbers of users and the involvement of heterogeneous mobile devices. The focus therein is put on the deployment process. Hence, the evaluation is based on an actual deployment, namely the migration of *Cyface*, an existing crowdsourcing project build from scratch, into using our proposed infrastructure.**

I. Introduction

The idea of dividing complex tasks into manageable subtasks is not new, e.g. early gatherers spread out in search of berries in small groups. Once a promising gathering ground was located, the entire group could focus on gathering; hence, the complex task of locating the gathering ground was crowdsourced in simplified subtasks of searching smaller territories.

This ancient idea of dividing and conquering complex tasks is topical even today; however, located on a different level of tasks. The growing complexity of modern applications, especially in mobilised civilisations with smartphones, smart sensors, wireless sensor nodes, etc. generates an almost insoluble mass of scenarios for data aggregation, data processing, result extraction and task deployment. Often, these kinds of challenges apply to limited resources situations, e.g. non-profit, low budget or start-up organisations, but also education and research. One then is dependent on the support of unsalaried or minimal payment contributors.

And this is where crowdsourcing comes into play by providing the crowdsourcer a means of distributing their tasks, and allowing the contributors on demand access to individual tasks. Of course, this requires a reliable infrastructure for task division, task distribution, result aggregation, result processing, result verification, reward redistribution, community layout, et cetera.

A good definition of crowdsourcing was provided by Estellés-Arolas and González-Ladrón-de-Guevara in [1]:

Crowdsourcing is a type of participative online activity in which an individual, an institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task. The undertaking of the task, of variable complexity and modularity, and in which the crowd should participate bringing their work, money, knowledge and/or experience, always entails mutual benefit. The user will receive the satisfaction of a given type of need, be it economic, social recognition, self-esteem, or the development of individual skills, while the crowdsourcer will obtain and utilize to their advantage that what the user has brought to the venture, whose form will depend on the type of activity undertaken.

An example in recent years is the Berkeley Open Infrastructure for Network Computing[1] (BOINC) which allows almost all of these aspects to be addressed with exception of data aggregation. Another example is Amazon Mechanical Turk[2] (MTurk); however, it is a highly specialised solution in which control over the crowdsourcing process must be surrendered to Amazon.

Therefore, we decided to design an open infrastructure [2], which is deployable on standard web server infrastructures. This allows the crowdsourcer to deploy corwdsourcing without having have to facilitate a server dedicated to crowdsourcing alone; existing servers can be utilised. At the same time, this enhances control over their crowdsourcing process as it is deployed within their own sphere of control, not on a third party server. This said, while still being able to easily distribute tasks to an available crowd of contributors.

The paper is organised as follows: In the beginning we explore requirements for a generic crowdsourcing platform and discuss related work. Then, we present our concept for a reusable crowdsourcing platform focussing on the deployment of crowdsourcing projects. After that, we discuss how an existing crowdsourcing project, which was built from scratch could profit from the migration to our generic platform. Finally we draw a conclusion and present intended future work.

II. Requirements Analysis

As more processes in business and research depend on the aggregation and processing of quantities of data distributed in area as well as type, we have identified some key requirements in order to sufficiently acquit these dependencies. In our belief, a prime factor of crowdsourcing success, is to actually make crowdsourcing a feasible solution from the technology oblivious crowdsourcer's point of view. This can only be achieved when crowdsourcing is an easily understandable and implementable process. Therefore, the definition of crowdsourcing processes should be self-explanatory and abstract enough in order to not derange potential crowdsourcers in database, communication or user and data management details. Easing deployment processes is a second step to successfully make crowdsourcing an acclaimed process. Potential crowdsourcers must be able to easily deploy their crowdsourcing process after having it defined as described above. This requirement goes back to back with abating crowd convocation; namely, the crowdsourcer should not be burdened with finding their crowd for the crowdsourcing process.

The next requirement we believe is imperative is having the crowdsourcers maintain control over their own crowdsourcing process. Crowdsourcers should be enabled to have control over their

[1]http://boinc.berkeley.edu/
[2]http://aws.amazon.com/de/mturk/

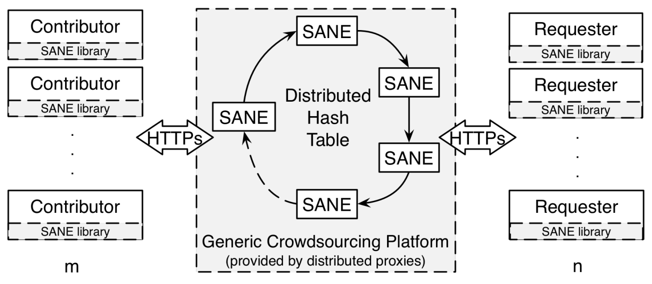

Fig. 1. Architecture of the crowdsourcing Platform

crowdsourcing process from deciding where data is aggregated, who collects it and where it is processed, e.g. on hardware under their own physical control.

From a more pragmatic point of view, one can simply require lessening programming efforts for potential crowdsourcers; hence, allowing access to crowdsourcing to potential crowdsourcers coming from other domains than IT.

However, from the crowd's point of view, requirements differ. We have identified a set of functional and non-functional requirements which partially contradict each other. In our belief, the key requirements include but are not limited to the following: Simple acquisition of necessary applications, allowing installing no harder than for any other application a user would expect; simple sign-up process, in order to quickly start participating and acknowledging something happens; single sign on (SSO), to seamlessly integrate with existing applications in use; an emancipated choice of when and where to participate in which crowdsourcing project, in order to elevate the users' self-awareness of what is happening on their devices and when and where their data is transmitted or shared; choice of anonymity towards the crowdsourcer, allowing participation and rewarding without risk of exposure towards the crowdsourcer, ideally allowing participation in crowdsourcing projects one would normally not be eligible for; choice of disclosing achievements like badges, points, high scores, et cetera in order to actively compete with other members in the crowd and being able to ascertain one's participation in comparison to the rest of the crowd, or a selected subset of the crowd; and of course, some sort of reward, be it monetary or ideally. A non-universal requirement is abidance by highest standards of privacy and data protection.

Combining crowdsourcers' and the crowd's requirements into a set of requirements towards the system proves challenging. For the concept presented in this paper, we have endeavoured at fulfilling the intersecting set of maximum size.

III. Concept

The basic idea of our approach is to identify functionality shared in a multitude of crowdsourcing projects and to provide it as a reusable component. Thus, the provided infrastructure can not relief the requester from the burden of implementing a client component (i.e. a Web application or mobile App) for interacting with the user and a server component for processing submissions but we aim to make the development and deployment of crowdsourcing projects as easy as possible.

A. Architecture

Fig. 1 depicts the architecture of our generic platform. The main component encapsulating the reusable crowdsourcing functionality it called Server Access Network Entity (SANE). The responsibility of a SANE is to handle the entire user, client device, submission management and forwarding for a multitude of crowdsourcing projects ensuring their anonymity, security, fault tolerance and performance.

As shown in figure 1 the generic crowdsourcing platform is organised as a federation of cooperating SANE components to ensure the scalability and fault tolerance of the platform. The SANE components are organising themselves utilising a distributed hash table (DHT), proven to be efficient even with a vast number of nodes. Each SANE instance maintains an area of hash values corresponding to unique identifiers of each client device. Thus, each client device can be mapped to a particular SANE instance. Neighbouring SANE instances act as backup for each other to ensure fault tolerance. Further details about the self-organisation of SANE components can be found in [2]. World-wide distribution of SANEs and solutions to the problem of finding a SANE handling one's corresponding hash area while minimising communication distances are described naively utilising the Domain Name System in [3], and sophisticated utilising a location-aware DHT algorithm in [4].

The capturing of submissions at the contributors side and the processing and assessment of submissions at the requesters side is performed by software provided by the crowdsourcing requester. To ease the development of the contributor and requester software SANE functionality can be accessed using the *SANE library* available for the client and server side of crowdsourcing projects. In particular, the SANE library provides functions for creating user accounts and user authentication as well as for creating, encrypting, signing and sending submissions according to well defined submission types to a SANE instance. In addition, submitters can refer to previous submissions and are able to withdraw them.

B. Submission handling

Submissions arriving at a SANE component are encrypted, signed and contain the contributors credentials. The SANE instance decrypts the submission, verifies the submission type, contributors credentials and signature and stores each submission into a local database. It further replaces the contributors credentials with an anonymised submitterID, signs and encrypts it with its own keys and forwards it to the requester. Finally, the requester can decrypt the submission and verify the signature. The anonymised submitterID does not uncover the identity of the contributor but allows the requester to associate submissions provided by the same submitter. After processing, an acknowledgement message combined with a feedback about the quality of the submission is send to the SANE. Again this message is encrypted and signed. Based on the feedback the SANE can rate, reward or ban contributors. Further details about submission handling can be found in [2].

C. Deployment

The entire deployment process can be summarised into six more or less simple steps:

1) decide which data to aggregate, thereby define variables and desired database structure,
2) define methods and targets,
3) consolidate definitions into a single crowdsourcing properties file (XML file) to be uploaded to a SANE of one's choosing,
4) deploy crowdsourcing derivate on SANE by calling a special deployment method,
5) implement receiver on server under own control, and
6) make client module available.

The first step is inevitable and has to be conducted whether utilising our concept, or not. Hence, as crowdsourcers are required to define

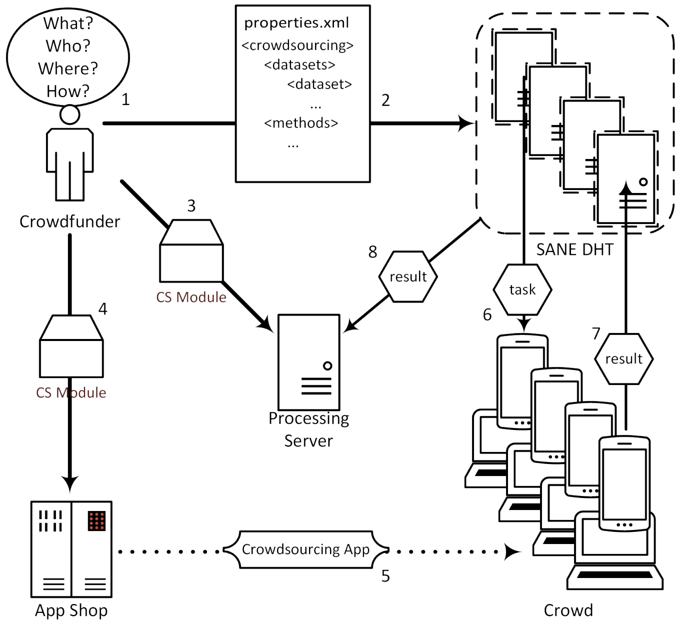

Fig. 2. Simplified deployment process: 1. conceive process, 2. deploy XML specification, 3. deploy server-side module, 4. deploy client-side module, 5. download client app, 6. fetch task from SANE DHT, 7. submit result to SANE DHT, and 8. forward result to server.

what they want to aggregate in order to achieve their goals, it is a simple means of specifying it in term of variables for the data and a comprehensive means of storing the so defined data.

In the second step, the crowdsourcers need to define in what manner the defined data shall be aggregated. There might be only one type of data to be aggregated in one particular situation, but often different types of datasets for different tasks and situations occur. Defining methodical rules for these altering conditions is natural, and this step simply formalises them into methods that can be executed within the crowdsourcing infrastructure.

Next up, the third step simply combines the results of steps one and two into a machine readable format; namely, an XML file. All that then needs to be deployed into the crowdsourcing infrastructure then is this file; hence, having the infrastructure handle the rest automatically.

Following, the fourth step is the actual deployment of the crowdsourcing process within the crowdsourcing infrastructure. As the SANEs are distributed in a self-organising Distributed Hash Table, it is sufficient to instantiate the deployment on one SANE. The rest can then be handled automated.

On a side note, one needs to mention that these four steps can be provided by one integrated user-frontend. Within that, the definition of crowdsourcing processes can be reduced to a simple click&deploy task, ideally utilising friendly drag&drop techniques. However, simple the interface, providing this functionality has not been in our research focus, yet. Nevertheless, one of the next research tasks of ours is providing this type of graphical user interface.

As easy as the first four steps present themselves, in this conceptual design, the last two steps remain as challenges for potential crowdsourcers. Programming skills are required for the aggregation and reception of data. However, corresponding server-side and client-side modules handling the communication and distribution details of the crowdsourcing infrastructure are currently in development, reducing

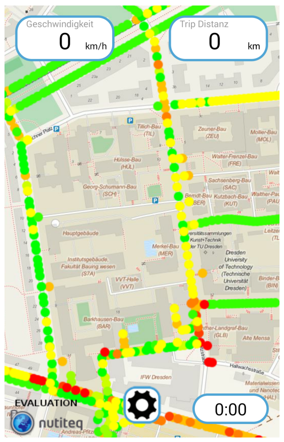

Fig. 3. Example of the current Cyface client application.

the final demands to core conception and implementation tasks that are inevitable. Easing of the provision process in this manner helps potential crowdsourcers to focus their time and effort on the before mentioned inevitable tasks, namely conceiving the conceptual crowdsourcing process (what from whom, where and when?), the conception and implementation of the data aggregation sub-process (client-side front- and backend), the conception and implementation of the data processing sub-process (server-side backend), as well as the conception and implementation of the information extraction (result computation) on the processed data (server-side frontend). – The entire handling of the crowdsourcing process is removed from the crowdsourcers programming scope; hence, easing the development manifold.

The deployment process as well as an exemplary task provision and result submission is depicted in figure2.

IV. Evaluation

We demonstrate the applicability and advantages of our generic crowdsourcing platform with the discussion of the migration of a proprietary crowd sourcing system called *Cyface*.

A. *The Cyface Crowd Sourcing Application*

The Cyface Project is a service to increase cycling comfort. The problem it tries to address is the poor quality of many streets encountered, especially in urban areas. In contrast to cars and other vehicles, bikes are not fitted with enough suspension to make driving cobblestone roads for example comfortable. Additionally suspension also requires more energy, which bikers are not so fond of spending. This is a problem especially for E-Bikes which results in drastically reduced range if the wrong roads are taken.

The goal of Cyface is to provide a mobile app that may be used by the crowd to capture data about street quality via smartphone sensors. This data is collected and aggregated to draw a complete picture of the street quality for some region and update this picture as street quality increases or decreases due to repairs or use. Using this data it is possible to create a navigation software capable of providing not only a short route but also a comfortable one. Figure 3 shows a screenshot of how the current application looks like.

Currently our app is implemented from scratch and sends its data directly to a central Cyface server for analysis. The same server is also used to send the aggregated data back to the app for visualisation purposes.

B. Cyface – SANE Integration

Integrating Cyface with the SANE infrastructure requires three distinct adaptations to the current infrastructure.

At first we need to write the interface description as described in Section III-C. The current Cyface mobile application provides its data via a JSON-Array using an HTTP-REST Post request directly on the Cyface server. There are two ways to declare this transportation using the SANE XML description. One way would be to extract the different variables from the JSON format and declare each one as a separate variable. This way it becomes easier to get an overview of the required parameters via the interface description as well as view the raw data inside the SANE database. It also lays the foundation for extending SANE with review functions working on the data transmitted by the crowd. The second approach is to declare the whole payload of one data recording in a single payload variable. This second approach has the advantage that it becomes possible to encrypt the whole payload as well as the advantage that the data processing interface on server side may continue to work with the current format.

Considering the two before mentioned options, an exemplary XML specification matching the Cyface dataset like `[{"ax":0.1,"ay":0.3,"az":0.2,"gpstime":2014-09-08T14:48:27:815+02:00,"lat":23,"lon":165,"speed":5.1,"label":"mountainbike","vecx":3.1,"vecy":2.7,"vecz":6.5,"z":0.1,"time":2014-09-08T13:47:15:758+02:00},{"ax":0.1,"ay":0.3,"az":0.2,"gpstime":2014-09-08T14:48:27:815+02:00,"lat":23,"lon":165,"speed":5.1,"label":"mountainbike","vecx":3.1,"vecy":2.7,"vecz":6.5,"z":0.1,"time":2014-09-08T13:47:15:758+02:00},...]` (i.e. an array of variable element count over a defined set of variables) is shown for the multiple variables variant in listing 1 and for the single variable variant in listing 2. – Note, for the current version of our XML specification, the variant using multiple variables allows fine granular constraints in form of regular expressions for each variable, whereas the single variable variant only supports the constraint "JSON," and no finer constraints can be defined for the elements of the JSON-string. The variable amount of array elements represented in the exemplary JSON-string are respected by the "Array(0)" constraint (array of minimum length 0).

Listing 1. Shortened XML for multiple variables

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<crowdsourcing xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
     xmlns="http://the-tester.de/SANE/0.1"
     xs:schemaLocation="http://the-tester.de/SANE/0.1
␣␣␣␣http://the-tester.de/SANE/0.1.xsd">
  <name>Cyface</name>
  <!-- ... -->
  <datasets>
    <dataset name="RoadQuality">
      <row name="ax" type="DECIMAL(2,1)" />,
      <!-- ... -->
      <row name="time" type="VARCHAR(29)" />
    </dataset>
  </datasets>
  <methods>
    <method name="submitRoadQuality" type="Setter"
         dataset="RoadQuality" target="SANE"
         stub="false" deprecated="false">
      <description>...</description>
      <optionalSystemVars />
      <input>
        <var name="ax" minsize="3" maxsize="3">
          <constraints>\d\.\d&amp;&amp;Array(0)</constraints>
          <description>The X-axis acceleration.</description>
        </var>
        <!-- ... -->
        <var name="time" minsize="29" maxsize="29">
          <constraints>\d{4}(\-\d{2}){2}T(\d{2}:){3}\d{3}+\d{2}:\d{2}
            &amp;&amp;Array(0)</constraints>
          <description>The device time (may be offset to gpstime).</description>
        </var>
      </input>
      <outputEncoding>text/plain</outputEncoding>
    </method>
  </methods>
</crowdsourcing>
```

A second step requires the adaptation of the current mobile application to communicate with the SANE instead of with the Cyface server directly. This involves adding appropriate headers to the client applications Post request. Since the SANE uses an HTTP based communication as well the remaining adaptations should be straight forward.

Listing 2. Shortened XML for a single variable

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<crowdsourcing xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
     xmlns="http://the-tester.de/SANE/0.1"
     xs:schemaLocation="http://the-tester.de/SANE/0.1
␣␣␣␣http://the-tester.de/SANE/0.1.xsd">
  <name>Cyface</name>
  <!-- ... -->
  <datasets>
    <dataset name="RoadQuality">
      <row name="data" type="VARCHAR(2048)" />
    </dataset>
  </datasets>
  <methods>
    <method name="submitRoadQuality" type="Setter"
         dataset="RoadQuality" target="SANE"
         stub="false" deprecated="false">
      <description>...</description>
      <optionalSystemVars />
      <input>
        <var name="data" minsize="0" maxsize="2048">
          <constraints>JSON</constraints>
          <description>The road quality data.</description>
        </var>
      </input>
      <outputEncoding>text/plain</outputEncoding>
    </method>
  </methods>
</crowdsourcing>
```

Finally as a third step it is necessary to adapt the server interface to receive SANE messages in exchange for the current direct communication. For a short transition period it would even be possible to run both interfaces simultaneously and receive messages from the previous application as well as the new one.

In addition to the crowd based data capturing the Cyface server provides an interface to fetch aggregated data. This interface is used to realise the street quality visualisation and navigation functions. Since it requires no data acquisition itself and can be used anonymously it does not need to be proxied by SANE. Instead applications using the Cyface data are able to access this REST interface directly as it is.

C. Advantages of the Cyface – SANE Integration

We see several advantages when using the Cyface – SANE integration instead of the direct communication between the Cyface application and the Cyface Server. Most of these advantages result from the typical requirements for an m:n communication between clients and servers, which are satisfied by using a mediator like the SANE infrastructure. In contrast to direct communication the Cyface application and corresponding servers are loosely coupled, which simplifies exchanging or modifying either end of the communication.

In addition the SANE infrastructure relieves the Crowdsourcer of managing user accounts, thereby providing the user with a hopefully trusted third party. The big advantage for the user of the Cyface application is increased privacy, since the Cyface server provider gets only anonymised data from the SANE server. Due to the loose coupling it becomes possible to move the Cyface servers from one physical location to another, without breaking crowd data capturing. Even though the clients are notified of the servers current inactivity, they would still be able to transmit data to the SANE infrastructure for intermediate storage. As soon as the server comes back online it is able to fetch the accumulated data and continue its work on a complete data set.

V. Related Work

In IT, crowdsourcing is a rather young concept in the field of computer science, emerging in the late 1990s or early 2000s respectively. For example, the *SETI@home* project, a *Crowdworking* derivate, was released to the public on 17 May 1999, making the concept of a distributed calculation known to the general public. After that, manifold derivates of crowdsourcing have been investigated, e.g. [5], [6]. Commercial uses of crowdsourcing include but are not limited to *Crowdvoting*, e.g. *Threadless.com*[3], *Crowdwisdom* , e.g. Wikipedia[4], *Crowdfunding*, e.g. Kickstarter[5], *Crowdpurchasing* , e.g. *Letsbuyit.com*[6], as well as *Crowdworking*, e.g. NASA's 2006 *Stardust* project[7].

Social sensing can be considered as a particular type of crowdsourcing. Focussing on humans and data strongly correlated to humans and their social surroundings [7]–[9], allowing information retrieval in a social context.

Another IT-area of immense interest for crowdsourcing are locations-based services. Exemplary, *OpenStreetMap*[8] utilises a steadily growing crowd in order to create map data of high quality within a very short time. The aggregated data even provides better information than comparable solutions of commercial tenderers [10]. This allowedly limited list by itself already sufficiently reveals the high potential of crowdsourcing in various IT areas. However, the aspect of infrastructure targeted for this manifold of potential uses is limited.

As mentioned earlier, the Berkeley Open Infrastructure for Network Computing (BOINC) can be considered a well established Crowdworking infrastructure solution. The original idea was to provide an open source middleware for grid computing by means of distributed private volunteers. However, in our definition this is Crowdworking; hence, Crowdworking projects operate LAMP[9] systems, providing work units to the crowd, receiving calculated results, integrating results of different members of the crowd into one dataset, as well as rewarding credit points. On the side of the crowd, each client consists of shared components for BOINC communication beneath an exchangeable client-side application logic differing from project to project. This allows participation of one client in several projects at the same time; processing time is shared amongst each application following a user-definable set of rules. Nevertheless, BOINC is entirely focussed on Crowdworking; data/information aggregation, et cetera is not considered. – Here we propose a more generalised concept, supporting all derivates of crowdsourcing, including Crowdworking. Due to the limitations of our current prototype, it is only important to operate on TCP/IP networks with HTTP as communication protocol and SSL/TLS encryption and authentication at this time. Hence, our proposed concept also requires no less and no more than LAMP servers. Further, the application logic is separated into a client part – the ascertainment logic – and a server part – the information extraction – for each crowdsourcer. While BOINC has the clients communicate with the crowdsourcer's servers directly and provide credit information from the BOINC website, our proposal envisages the SANE proxy to be mandatory for crowdsourcing-related client-server communication as well as user management. Our proposal only requires the crowdsourcer to provide data-related functionality (aggregation and processing), while all other tasks are handled by the SANEs within the crowdsourcing architecture.

Another well established infrastructure example is Amazon Mechanical Turk (MTurk). As [11] precisely summarises, this commercial infrastructure can be utilised for many beneficial properties; namely, crowdsourcing processes can be conducted very fast and with restricted financial means, the result quality is of high acceptability, and the diversity of workers is very good. However, Mechanical Turk has several known drawbacks [12] which shall not be recollected here. – Our concept differs in key aspects from MTurk. Rather than awarding payments to the crowd, we wish to focus on subliminal rewards. The focussed reward desire of any member of the crowd should be fulfilled by using the infrastructure a priori, not by using the infrastructure in order to be rewarded. Hence, from the users' point of view, having delivered functions the crowd desires anyway, the crowdsourcing process is merely a side product of application utilisation. For example, using a navigation application could be the focus of the user; by using our infrastructure, map corrections are provided to the map service provider. In return – and this is the reward – the user gets enhanced maps. Of course, we also strive to support classic MTurk-like reward-based crowdsourcing processes; however, rewarding in means of crowd points, et cetera is *not* supported, yet. The challenge will be, to provide a rewarding system with comparable rewards over diverse crowdsourcing processes.

A not to be neglected aspect of crowdsourcing processes is the workflow and division of tasks into subtasks per crowd member. Turkomatic [13] addresses these issues by attempting to provide a means of crowdsourcing workflow design for MTurk. Even though imperative, this workflow aspect is *not* addressed in our concept. Of course, we deem automated division of tasks within the crowdsourcing infrastructure important in order for the crowdsourcer not having have to handle division themselves, so this aspect needs to be addressed in future updates of our concept. Another aspect currently *not* supported by our concept is related to the prior. Having tasks subdivided automatically yields the question of how these tasks are processed. In general, parallel processing is desirable; however, one might deem iterative processing fruitful. Exactly this problem is addressed by TurKit [14]. Pending validation as imperative feature, automated division into iterative processing units could prove a desirable extension of our concept.

Very similar to our concept proves Curio [15]. Important insights identified are equivalent to ours, and the drawn conclusion may surpass our current conceptual state; however, Curio focusses on observable crowdsourcing processes with real-time interventions from the crowdsourcer, whereas we strive towards an entirely automated crowdsourcing process in which the crowdsourcer may or may not intervene at post-submission time[10]. Additionally, Curio focusses merely on Human processing, where in contrast our proposed solution shall support any type of client-side application. Especially, this includes automated background crowdsourcing which does not require user interaction; in [3] we declared these types of crowdsourcing "Unaware Direct Crowdsourcing" (UDC) and "Unaware Indirect Crowdsourcing" (UIC), in contrast to aware (in)direct derivates (A$*$C).

Of growing interest we also identify support of mobile crowds, both in human as well as IT resources, namely supporting devices bound to mobile communication technologies. [16] points out some interesting aspects to this type of support, especially in use cases of taking advantage of current mobile abilities, e.g. location-based tracking. However, the manifold possibilities are limited when confronted with unsteady or slow connections, et cetera. – We aim to focus these issues in the future; for example, by the use of a token-based submission system, or by utilising a transport protocol optimised for unstable (wireless) connections. From the perspective of our current

[3]http://beta.threadless.com/
[4]http://en.wikipedia.org/wiki/Main_Page
[5]http://www.kickstarter.com/
[6]http://www.letsbuyit.com/
[7]http://stardust.jpl.nasa.gov/home/index.html
[8]http://www.openstreetmap.org/
[9]Web-server operating on Linux, Apache httpd, MySQL and PHP

[10]Of course, when shortening the post-submission time to a very minimum, a quasi real-time intervention is thinkable.

SANE system, the utilised transport protocol is irrelevant to the communication protocol used, i.e. HTTP with SSL/TLS.

VI. Conclusion

In this paper we presented an extensible, self-managing crowdsourcing infrastructure in order to ease deployment of crowdsourcing processes. Basic requirements are fulfilled, and the successful migration of an existing application into utilising our infrastructure proves the concept valid. However, major aspects need to be addressed in future refinement to the concept, e.g. allowing definition of campaigns, division of tasks, a reward system, or a more refined constraint specification for data.

References

[1] E. Estellés-Arolas and F. González-Ladrón-de-Guevara, "Towards an integrated crowdsourcing definition," *Journal of Information Science*, 02 2012, dOI: 10.1177/016555150000000.

[2] T. Hara, T. Springer, G. Bombach, and A. Schill, "Decentralised Approach for a Reusable crowdsourcing Platform utilising Standard Web Servers," in *PUCAA 2013: First International Workshop on Pervasive Urban Crowdsensing Architecture and Applications (UbiComp 2013 Workshops)*, 2013.

[3] T. Hara, "Towards a reliable Architecture for crowdsourcing in the Context of the MapBiquitous Project," Thesis (Diplomarbeit), Technische Universität Dresden, October 2012.

[4] S. Zhou, G. R. Ganger, and P. A. Steenkist, "Location-based node IDs: enabling explicit locality in DHTs," *Carnegie Mellon University – Research Showcase @ CMU*, vol. 2003, http://repository.cmu.edu/cgi/viewcontent.cgi?article=3206&context=compsci.

[5] D. Brabham, "crowdsourcing as a Model for Problem Solving – An Introduction and Cases," *Convergence: The International Journal of Research into New Media Technologies*, vol. 14, no. 1, 2008, DOI: 10.1177/1354856507084420.

[6] J. Howe, "crowdsourcing: Why the Power of the Crowd is driving the Future of Business," 2008, http://www.bizbriefings.com/Samples/IntInst%20---%20crowdsourcing.PDF.

[7] R. Ali, C. Solis, M. Salehie, I. Omoronyia, B. Nuseibeh, and W. Maalej, "Social Sensing: When Users Become Monitors," *ACM D.2.2 [Softwareengineering]: Design Tools and Techniques*, vol. ESEC/FSE'11, 09 2010, ACM Code: 978-1-4503-0443-6/11/09.

[8] A. Madan, M. Cebrian, D. Lazer, and A. Pentland, "Social Sensing for Epidemiological Behaviour Change," *ACM I.5.4 [Pattern Recognition]: Applications; ACM H.4.m [Information Systems]: Miscellaneous*, vol. UbiComp'10, 09 2010, ACM Code: 978-1-60558-843-8/10/09.

[9] O. Telhan, "Social Sensing and Its Display," Master Thesis, Massachusetts Institute of Technology, 08 2007, Bilkent Univeristy, Ankara.

[10] P. Neis, D. Zielstra, and A. Zipf, "The Street Network Evolution of Crowdsourced Maps: OpenStreetMap in Germany 2007-2011," *Future Internet*, vol. 4, no. 1, pp. 1–21, 12 2011. [Online]. Available: http://www.mdpi.com/1999-5903/4/1/1/

[11] O. Alonso, "Perspectives on Infrastructure for crowdsourcing," in *CSDM 2011: Workshop on crowdsourcing for Search and Data Mining (WSDM 2011), Hong Kong, China*, 2011.

[12] P. Ipeirotis, "Plea to Amazon: Fix Mechanical Turk," Blog, 10 2010, http://behind-the-enemylines.blogspot.com/2010/10/plea-to-amazon-fix-mechanicalturk.html.

[13] A. Kulkarni, M. Can, and B. Hartmann, "Collaboratively crowdsourcing Workflows with Turkomatic," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, ser. CSCW '12. New York, NY, USA: ACM, 2012, pp. 1003–1012. [Online]. Available: http://doi.acm.org/10.1145/2145204.2145354

[14] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller, "TurKit: Tools for Iterative Tasks on Mechanical Turk," in *Proceedings of the ACM SIGKDD Workshop on Human Computation*, ser. HCOMP '09. New York, NY, USA: ACM, 2009, pp. 29–30. [Online]. Available: http://doi.acm.org/10.1145/1600150.1600159

[15] E. Law, C. Dalton, N. Merrill, A. Young, and K. Z. Gajos, "Curio: A Platform for Supporting Mixed-Expertise crowdsourcing," http://www.aaai.org/ocs/index.php/HCOMP/HCOMP13/paper/viewFile/7534/7471.

[16] D. Schlagwein and F. Daneshgar, "User Requirements of a Ccrowdsourcing Platform for Researchers: Findings from a Series of Focus Groups," in *18th Pacific Asia Conference on Information Systems (PACIS 2014), Chengdu, China*, 2014.