An Extendable Toolkit for Managing Quality of Human-Based Electronic Services

David Bermbach, Robert Kern, Pascal Wichmann, Sandra Rath, Christian Zirpins, David.Bermbach,| Christian.Zirpins|
Robert.Kern@kit.edu, Pascal.Wichmann|Sandra.Rath@student.kit.edu
Karlsruhe Institute of Technology, Institutes AIFB/KSRI
Englerstrasse 11, 76131 Karlsruhe, Germany

Abstract

Micro-task markets like Amazon MTurk enable online workers to provide human intelligence as Web-based on demand services (so called *people services*). Businesses facing large amounts of knowledge work can benefit from increased flexibility and scalability of their workforce but need to cope with reduced control of result quality. While this problem is well recognized, it has so far only rudimentarily been addressed by existing platforms and tools. In this paper, we present a flexible research toolkit which enables experiments with advanced quality management mechanisms for generic micro-task markets. The toolkit enables control of correctness and performance of task fulfillment by means of continuous sampling, dynamic majority voting and worker pooling. While we demonstrate its application and performance for an OCR scenario building on Amazon MTurk, the toolkit supports the development of advanced quality management mechanisms for a large variety of people service scenarios and platforms.

Introduction

Interactive Web technologies enable scenarios, where – just like IT-resources in cloud computing – human intelligence is utilized on demand. *Human computation* [von Ahn, 2005] or *people services* (pServices) are terms that have been coined to describe this phenomenon. In particular, pServices have been defined as Web-based services that deliver scalable human workforce [Kern, Zirpins, and Agarwal, 2009].

Traditionally businesses hire a large workforce, where in case of high workload experts need to perform laborious and repetitive tasks that could be done by less skilled workers, or tasks are even left completely undone. Such tasks can now be outsourced to the "crowd". Among the potential advantages are cost-efficiency, scalability as well as flexibility. pServices particularly lend themselves to repetitive tasks where automated approaches still fail.

While a number of applications are being considered, we focus on Web-based micro-task markets like Amazon Mechanical Turk (MTurk). Here, service requesters can publish small tasks in form of an open call. Micro-task markets can be considered typical examples of pServices as their (mainly monetary) incentive scheme allows for active scaling of human resources (as opposed to games [von Ahn and Dabbish, 2004] or CAPTCHAs [von Ahn et al., 2008]).

Typical application scenarios of pServices revolve around information, i.e. the generation of information and the improvement of information quality all along the information lifecycle. Furthermore, existing applications cover virtually all fields of human intelligence. Human visual perception, for instance, is used for digitizing services, i.e. deciphering handwritings [Little et al., 2009] or translating document scans into machine-encoded text (Optical Character Recognition, OCR) [Kern, Thies, and Satzger, 2010].

As there is limited control over the individual contributors in pService scenarios, quality management (QM) is a vital necessity [Kern, Zirpins, and Agarwal, 2009]. Existing toolkits like Crowdflower (www.crowdflower.com) or TurKit [Little et al., 2009] however do not pay specific attention to efficiency and do not provide goal-based quality management capabilities. In this paper, we present the CSP/WMV toolkit, a convenient, flexible and generally platform-independent toolkit for the quality management of pServices as an attempt to close this gap.

In the following, we will recap the conceptual background and describe the toolkit design and architecture as well as its implementation. We use an OCR scenario to evaluate the toolkit in both simulations and real-time experiments. Finally, we discuss related work before we conclude and give an outlook on the future development of the toolkit.

Conceptual Background

Our toolkit builds on former work on statistical quality management for pServices [Kern, Thies, and Satzger, 2010]. We refer to it as *CSP/WMV approach* which indicates the building blocks of the mechanism: the *continuous sampling plan CSP-1* [Dodge, 1943] and the *weighted majority vote (WMV)* [Kern et al., 2010]. It is a specific form of a majority vote approach that aggregates the results of multiple workers in order to gain reliable results. Existing applications typically apply a fixed level of redundancy to each individual task, i.e. each task is performed by a well-defined number of multiple workers. From the perspective of quality management this means that the quality of each individual task is validated. However, concepts of statistical quality control (SQC) show that the quality management effort can

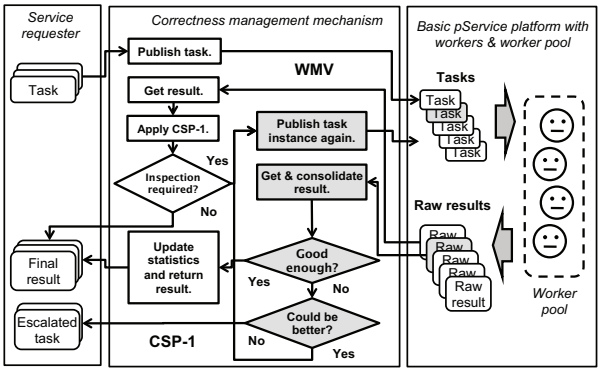

Figure 1: Basic concept of CSP/WMV quality management approach for pServices

usually be reduced drastically by using sampling instead of performing a full inspection of all individual items [Montgomery, 2008]. Moreover, a fixed degree of redundancy is both inefficient and incapable of assuring a certain level of result quality because the level of agreement (and so the expected result quality) varies depending on the error rates of the involved workers.

Correctness Management

The approach proposed by [Kern, Thies, and Satzger, 2010] increases the efficiency of the traditional majority vote (MV) by using a combination of a statistical quality control mechanism with a dynamic form of a majority vote mechanism, the weighted majority vote (WMV). The QM effort is reduced in two ways: in horizontal direction by validating only a sample of tasks rather than all tasks and in vertical direction by dynamically adjusting the level of redundancy rather than working with a fixed level of redundancy. Moreover, the approach guarantees a certain quality level.

Figure 1 gives a conceptual overview of the CSP/WMV approach. One instance of each task submitted by a service requester is published to the pService platform immediately. Based on the raw result and the worker ID returned from the platform, the CSP-1 mechanism decides whether an inspection of the worker is required or not. If not, the raw result is assumed to be the final result which is returned to the service requester. In case inspection is required, the WMV is initiated in order to validate the raw result. This is being done by publishing additional instances of the task until the consolidated result either meets the required inspection quality level or until it turns out that the required level of quality will unlikely be met even when adding additional redundancy. In the latter case, the task is escalated back to the service requester so he can check the task or improve the task description. Once the inspection quality level is met, the statistics of all involved workers are updated and the final result is returned. More details can be found in our previous paper [Kern, Thies, and Satzger, 2010]).

Because the WMV as well as the traditional MV assume that the raw results delivered by multiple workers can be compared to each other or aggregated into a consolidated result, such mechanisms work only for deterministic tasks i.e. for tasks that have a certain well-defined optimal result [Kern, Thies, and Satzger, 2010]. Within the multi-faceted dimensions of quality, only the correctness dimension is addressed as the ability to return a minimum percentage of results that are free of errors [Kern, Zirpins, and Agarwal, 2009].

Performance Management

Beyond correctness control, the toolkit addresses performance by dynamic adjustment of the worker pool size. Concerning correctness the QM effort would be minimal when only assigning the most qualified workers to the tasks. However, this would compromise scalability, which is supposed to be the major advantage of the pService concept: if only the 2-3 top workers do all the tasks the execution time would increase drastically. Yet, increasing performance by adding less qualified workers to the pool comes at the price of additional redundant submissions that are needed to compensate for the less reliable results. Most use cases, though, will likely require both: a minimum level of result quality and a maximum execution time, i.e. a deadline by which all tasks must be completed. The CSP/WMV toolkit addresses this requirement by adapting the worker pool size dynamically with the goal of meeting the deadline more or less exactly. In descending order, only the most qualified workers are added to the worker pool. This keeps the QM overhead minimal while still completing all tasks in time.

Based on these considerations, we use two pools of workers: one whose members are allowed to participate (active) and one whose members are not (pending). Using these pools we then propose to include the following algorithm:

1. Check progress, i.e. calculate the ratio of completed assignments $\sum_t ca_t$ to requested assignments cr.
2. Given the total number of completed assignments ca_t during the time period t with the length T and the size of the active pool n, calculate $p_t = ca_t/T \cdot n$, which is the *average productivity per worker* in period t.
3. Estimate p^F_{t+1}, the average productivity per worker in the next period $t+1$. This might be done based on different forecasting methods (see below).
4. Adapt n, so that – in case $p_{t+1} = p^F_{t+1}$ – progress will have reached 100% at the deadline.
5. Restart at step 1 until $\sum_t ca_t = cr$ or until the deadline has been reached.

This algorithm checks progress in regular time intervals and forecasts future productivity based on progress in past intervals. If the projected completion time exceeds the deadline, the size of the active worker pool is increased. Example: if after half of the time only one quarter of the tasks have been completed, the size of the worker pool is tripled. We propose a set of forecasting mechanisms that differ regarding the number of past intervals and the weight with which they are taken into account.

Based on practical experience, we propose to use a slight variation of the algorithm, which never decreases the size of the active pool. This adds a small overhead in terms of costs

but avoids annoying the workers by continuously blocking and unblocking some of them due to small random fluctuations in the average productivity per worker. Furthermore, this strategy reduces the risk of exceeding the deadline.

Toolkit Design and Architecture

Following the considerations discussed so far, our toolkit offers the following features:

- Support of the CSP/WMV approach to manage correctness
- Support of dynamic worker pool management to control performance
- Generic approach adaptable to various pService platforms (specific MTurk support)
- Multiple scenarios and aggregation methods with seamless switch-over
- Simulation mode to gain reproducible and comparable results
- Extensibility regarding quality management mechanisms and quality dimensions
- Easy set up for particular scenarios using the configuration manager

From a generic perspective (fig 2), the toolkit architecture builds on three main components: the *core services*, the *correctness plugin* and the *time constraints plugin*.

Within the core services the *worker pool manager* is responsible for creating qualification tests and evaluating results. Furthermore, workers are sorted into pools in order to control which group may access certain tasks and which may not (active/pending pool). This component also provides import and export functionality for worker data and triggers synchronization with the pService platform.

The second component is the *task result collector*, which holds and persists worker submissions locally. As correctness control generally requires redundant work this component also groups redundant submissions for the same task. Plugins may add worker submissions or lock and retrieve a complete set of worker submissions.

The *task result fetcher* periodically polls the pService platform for worker submissions and ensures that a single submission is processed only once. Furthermore, it provides an internal queuing system which may be used by plugins. It also provides the so-called *incoming results queue* to which it sends a message for every new worker submission. For this purpose it first checks whether the submission was already processed. If not, it requests all existing submissions for that task from the task result collector as well as a lock on those and enqueues them within one message on the incoming results queue. To support multiple pService markets as well as a simulation mode, access to the respective systems is wrapped inside a *platform wrapper* component.

The *correctness plugin* requires three general steps: first, retrieve a message from the incoming results queue, second, process the message, third, return the data to the task result collector and release the lock. This minimum requirement

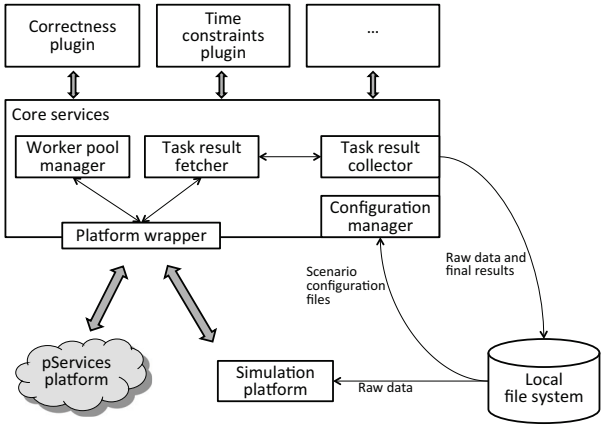

Figure 2: Component overview of CSP/WMV toolkit

is already included within the core services. A concrete implementation uses CSP-1 and WMV for processing the message. It checks incoming submissions, decides whether the task needs to be inspected (i.e. requires a redundant submission) based on the statistics held within the worker pool manager or whether it should be escalated (i.e. returned to the requester). If necessary, it then requests a redundant assignment via the platform wrapper. If a sufficient number of submissions exist for a given task, the correctness plugin uses WMV to determine which result is most likely the correct one. It then updates worker statistics via the worker pool manager and returns the results to the task result collector. Also, if necessary, qualification values of the worker are adjusted to block workers delivering poor quality.

As discussed, the worker pool manager controls access to tasks by sorting workers into pools. This mechanism is leveraged within the *time constraints plugin* which uses it to complete all tasks as close to the deadline as possible. For this purpose, the plugin offers three standard forecasting algorithms. Additional ones can easily be added. The first algorithm follows a simple last value strategy - it just assumes that the average productivity per worker as defined above remains constant. The other two implementations use some form of weighted averages where recent productivity values weigh more than older values. This is done to decrease exposure to random fluctuations.

The task result collector offers export functionality for final results as well as raw data. In the *SimInput* mode, raw results can be generated at a fixed level of redundancy which can be used for repeated simulations afterwards. We have also implemented a *simulation platform*, which mimics the behavior of workers on pServices platform by reading raw submissions from the file system whenever a task or redundant submission is required. To enhance the illusion of a real pServices platform simulated workers need some time to submit results and may be blocked by our platform. Also, a submission is either selected by chance or following the order in which the data was originally persisted. Future work will also consider the actual order of events as well as delays in between to give an even more realistic impression of

a live test.

Implementation

The toolkit was developed using Java SE 6 and is currently maintained as an Eclipse 3.6 project. All components described in the previous section are either implemented as separate classes or as a set of classes. Extensibility is generally achieved through usage of abstract classes and interfaces combined with reflections. Regarding application design, the toolkit is primarily designed as a library which can be used for multiple purposes, e.g. within a web application or a standalone tool. Initializer classes can easily be added by adapting the existing standalone initialization class.

Originally, the project targeted the MTurk platform and, hence, includes an implementation of the platform wrapper component for this provider. MTurk distinguishes between *HITs* (Human Intelligence Tasks), *HIT types* and *assignments*. HIT type describe a particular kind of task, e.g. OCR. HITs are concrete instances of such HIT type, e.g. one word which is to be transcribed. For each HIT multiple assignments may exist, i.e. several workers each submit an independent result (the "assignment") for the very same task. It is not only possible to specify from the beginning a particular level of redundancy (i.e. dynamically request multiple assignments for each HIT) but also to extend a HIT, i.e. request an additional assignment for an existing HIT.

The above features were mainly used within the correctness plugin: In the beginning a new HIT type and a corresponding number of HITs are created using the batch mode provided by the standard MTurk library for Java. Later on, the task result fetcher periodically polls the MTurk platform for assignments. Using the unique assignment ID it identifies duplicates which have already been processed and ignores those. All other assignments are grouped by the corresponding HIT ID which is also used as primary key within the task result collector. The current implementation parses the assignment content. This is an XML document using a custom answer comparison class, which is instantiated using reflections and a properties file. Based on these comparisons, it then identifies unique answers by which it groups the assignments. The worker ID of the assignment is then used to identify a unique worker object within the worker pool manager and its correctness values are retrieved. Based on this information the probability of an answer being correct is calculated for all existing unique answers. If this probability exceeds a threshold for one of the answers that particular answer is marked as correct, all answers are returned to the task result collector and the worker correctness values are updated. For complex scenarios, multiple answers can also be merged into a new answer which meets the quality needs better than any of the answers that were submitted by the workers. If an additional answer would be required as the probabilities are all below the threshold, the algorithm calculates whether the HIT should be escalated, i.e. returned to the requester. This is especially useful in tie situations, which usually occur when a task is particularly difficult. In an OCR scenario we have often observed this for the letters 'a' and 'o' or 'n', 'm', 'r' and 'v'. If still an additional submission is required and the task must not be escalated, the toolkit sends an extend HIT request to the platform wrapper which in turn requests an additional assignment from the MTurk platform.

For the time constraints plugin, we leveraged multiple MTurk features: First, MTurk offers qualification tests which can be used to test the workers prior to allowing them to work on a given task. We use this feature for initializing the correctness plugin with initial correctness values for the workers. Based on the results of the test we also assign a qualification to every participating worker. As our time constraints plugin requires control over worker pools we initially assign each worker a value of 1 if he is part of the active pool, for the pending pool we assign a 0 and spammers are set to -1 to permanently block them. In the HIT type of our task we then specify the access restriction of having a quality control qualification value of 1. If the time constraints plugin realizes that the size of the active pool should be increased it requests the x best workers within the pending pool from the worker pool manager and issues a *move to active pool* command. The worker pool manager executes this command and sends, via the MTurk platform wrapper, an update qualification value request with the parameter 1 to MTurk so that the corresponding workers are allowed to work on that HIT type. As described earlier, we avoid scaling the active pool down. For this reason, it is important to start with a small active pool and a large pending pool so that all scaling requests can be fulfilled. When starting the platform including the time constraints plugin, all known workers are sorted into their respective pools based on an imported worker pool file and their qualification values are updated accordingly. If it is not desired to use this plugin it is always possible to specify a HIT type which has no access restrictions or to move all appropriate workers to the active pool from the beginning. Furthermore, the toolkit supports having different qualifications for different kinds of tasks.

Evaluation

For the evaluation of the CSP/WMV toolkit, two different approaches have been used. While the correctness management capabilities were evaluated using a realistic scenario on top of the MTurk platform, the performance management feature was so far only evaluated by a simulation using dummy data.

The toolkit was evaluated using the same scenario and data set as in [Kern, Thies, and Satzger, 2010]: an OCR scenario which was performed on top of MTurk. The data set consists of 1176 handwritten words. In each of the tasks, a worker was asked to type in a single handwritten word which was displayed as an image file (JPEG). The expected optimal result (gold standard) was specified by the author of the handwriting himself. A compensation of 0.01 USD per assignment was paid to the workers. In order to exclude spammers and workers who submit poor quality right from the start, only such users were allowed to participate who had passed a qualification test. The test consisted of a series of 10 simple OCR tasks (10 words).

The actual evaluation is divided into two parts: in the first part, the QM approach was simulated based on a batch of

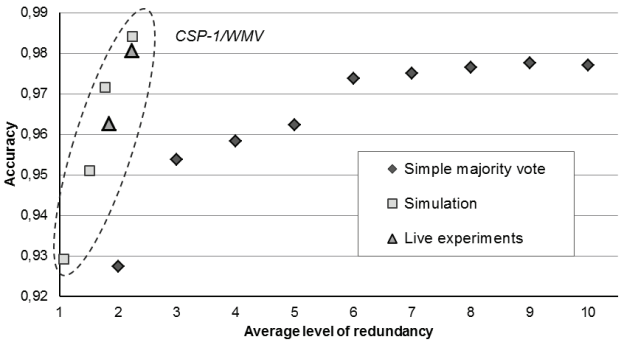

Figure 3: Accuracy vs. redundancy during live experiments and simulations with the correctness plugin

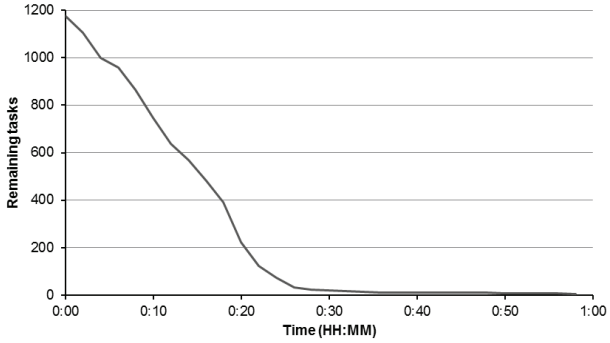

Figure 4: Decreasing number of remaining tasks during live experiments

raw results. In the second part, the QM approach was applied in real time.

Simulation Results

In order to compare the QM approach with the traditional majority vote mechanism, the first part of the evaluation was performed as a simulation on the basis of raw results from a batch comprising 10 redundant assignments per task. For simulating the traditional majority vote, a fixed number of raw results were used. For simulating the WMV, a varying number of raw results were used according to the dynamic concept of the approach.

Figure 3 shows the results of the QM approach compared to the traditional MV approach for four different quality goals: 0.925, 0.950, 0.975 and 0.990. The traditional MV was simulated based on the same data as the WMV by averaging all possible combinations of 2 to 9 answers within each set of 10 available answers per task for the twofold up to the 9-fold MV. With 0.984 the CSP/WMV approach almost perfectly meets the inspection quality goal of 0.99. It even outperforms the accuracy of a ninefold traditional MV (0.978). That is a remarkable result given that the CSP/WMV is 4 times more efficient as it requires only 2.25 workers per task compared to 9 workers per task for the basic ninefold MV approach. In other words: the CSP/WMV approach has reduced the quality management effort by some 75 percent compared to the traditional MV approach.

Within our simulation environment we have also performed first promising tests with the performance plugin which we are going to continue in our future research. Such experiments are time consuming because they require prior knowledge of a huge number of workers and hence only make sense for large batches of tasks.

Live Experiments

For the live experiments, the same data set was used as for the simulated experiments. However, initially only one instance (assignment) of each task was published. Depending on the results returned and the historical error rate of the worker who worked on the task, the QM approach dynamically decided in real time whether additional assignments had to be published.

In the CSP/WMV live experiment the number of remaining tasks decreases almost linearly until most of the tasks have been performed. Then, the execution performance slows down dramatically and asymptotically approaches zero (figure 4). This behavior can be explained by the fact that towards the end, the continuous stream of tasks is interrupted because there are temporarily no tasks available any more. Therefore, the workers consider the process to be completed and look for some other tasks to work on. However, as the CSP/WMV increases the redundancy sequentially, new assignments might be published even after all the available work had already been completed. This applies in particular to complex tasks for which a higher level of redundancy is required because there is less agreement among the workers. This effect is further increased by cherry-picking of the workers: some of the words are so hard to read that hardly anybody wants to take the risk of making a mistake. In order to avoid discontinuity and speed up the finalization of the process, the QM mechanism should switch to a fixed level of redundancy at the very end of the process.

The upper triangle in figure 3 indicates that in the live experiment, the efficiency of the CSP/WMV for a quality goal of 0.99 has been comparable to the simulation. It is noteworthy to mention that one of the 13 workers participating in the experiment has returned 238 incorrect results of a total number of 255 tasks he had worked on (the worker accidentally capitalized all words). Nevertheless, the CSP/WMV was able to cope with the situation and has still delivered satisfactory results.

For a quality goal of 0.95, the live experiment outperforms the simulation at a slightly higher redundancy (0.963 vs. 0.951). The average fraction inspected (AFI) of the live experiment shows a characteristic comparable to the simulation but it decreases more slowly with the increasing number of tasks (Figure 5). This difference can be explained with the dynamic nature of the WMV. There is a varying delay before the new assignments are grabbed by the workers and before they return a result. As this delay is not covered by the simulation, it takes more time in the live experiment until the workers build up reputation and until they reach the random inspection phase of the CSP/WMV. On the long run however, the same AFI should be reached as in the simulation.

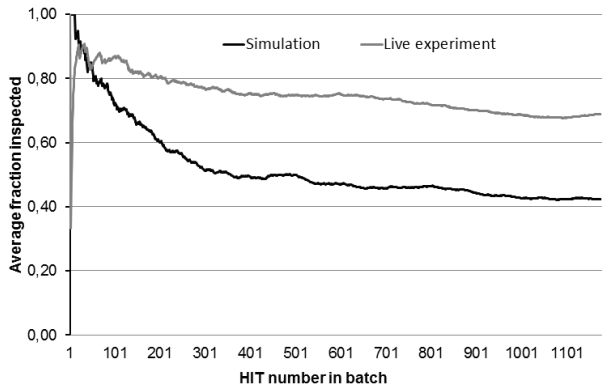

Figure 5: Average fraction inspected (AFI) with increasing number of tasks

Related Work

The toolkit we presented in this paper is not the only software addressing quality control of pServices.

MTurk already incorporates a basic quality control mechanism: Workers may be required to pass a qualification test. Furthermore, tasks can be reviewed and be rejected or accepted. The rejection rate is reflected within the workers KPIs which in turn can be used to control access to tasks, e.g. to block all workers with a rejection rate of more than 5 percent.

Crowdflower.com provides Labor-on-Demand, e.g. using the worker pool of Mechanical Turk, but augmenting this service by additional quality mechanisms. Gold standard tasks are randomly injected into the usual work processes to assess the workers performance (a job-specific worker trust). Multiple redundant assignments can be published to increase the confidence. Various aggregation methods can be set using the CrowdFlower Markup Language (CML), e.g. a single answer returned by the worker with the highest confidence or an average suitable for numeric answers. The functionality of Crowdflower is accessible via API.

[Little et al., 2009] present TurKit, a Java/JavaScript API specifically designed to handle iterative tasks on MTurk, i.e. dependent tasks that build on each other. TurKit stores results of precedent tasks so that they can be used as the basis for follow-up tasks. TurKit uses a combination of improvement and voting tasks to ensure the quality of the task result does not deteriorate.

Our approach is insofar different as it is the only one that offers a more descriptive QM approach. Here, it is possible to set goals which the CSP/WMV toolkit tries to fulfill using statistical measures. All other QM solutions aim to increase correctness while not offering any concrete information regarding the actual correctness off the results. Also, to our knowledge no solution exists which addresses performance management.

Conclusion and Outlook

In this work we have presented the design and implementation of a flexible toolkit which can be used to support various quality control approaches for pServices of different platform providers. Furthermore, we have presented a novel approach for managing the execution performance of pServices as and additional dimension of quality. First evaluation results based on an OCR scenario confirm the effectiveness of the tool and of the underlying approaches.

We hope that the scientific community will make good use of our toolkit and that our work will help to elevate QM efforts for pServices to a new level. Interested readers may obtain a copy of the CSP/WMV toolkit from the authors. Also, we intend to publish the toolkit as open source software in the near future.

There still remains a lot of work to be done on our toolkit. Right now, only three scenarios are supported. To improve convenience we plan to include more scenario presets, also to be able to evaluate our existing quality control mechanisms for different kinds of tasks. For example, regarding multi-labeling there is another publication in preparation.

Another aspect would be to further improve the simulation mode by not only using live answers but also using the time parameters originally observed with that particular answer, e.g. how long did it take from requesting the submission until the particular worker started working on it and how long did he take to complete the task. This would be especially useful for evaluation and comparison of various forecasting mechanisms which each include unique strengths and weaknesses. Based on these considerations, it should then be possible to identify the most accurate approaches for performance prediction in pServices.

References

Dodge, H. 1943. A sampling inspection plan for continuous production. *The Annals of mathematical statistics* 14(3):264–279.

Kern, R.; Bauer, C.; Thies, H.; and Satzger, G. 2010. Validating results of human-based electronic services leveraging multiple reviewers. In *Proceedings of the 16th Americas Conference on Information Systems (AMCIS)*.

Kern, R.; Thies, H.; and Satzger, G. 2010. Statistical quality control for Human-Based electronic services. In Maglio, P.; Weske, M.; Yang, J.; and Fantinato, M., eds., *Service-Oriented Computing - Proceedings ICSOC 2010*, LNCS, 243–257. Springer.

Kern, R.; Zirpins, C.; and Agarwal, S. 2009. Managing quality of Human-Based eServices. In *Service-Oriented Computing - ICSOC 2008 Workshops, ICSOC 2008 International Workshops, Sydney, Australia, December 1st, 2008, Revised Selected Papers*, volume LNCS 5472 of *Lecture Notes in Computer Science*, 304–309. Springer.

Little, G.; Chilton, L. B.; Goldman, M.; and Miller, R. C. 2009. TurKit: tools for iterative tasks on mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, 29–30.

Montgomery, D. 2008. *Introduction to statistical quality control*. New York, USA: Wiley and Sons, 6th edition.

von Ahn, L., and Dabbish, L. 2004. Labeling images with a computer game. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*. Vienna, Austria: ACM.

von Ahn, L.; Maurer, B.; McMillen, C.; Abraham, D.; and Blum, M. 2008. reCAPTCHA: Human-Based character recognition via web security measures. *Science* 321(5895):1465–1468.

von Ahn, L. 2005. *Human computation*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA. Adviser Blum, Manuel.