A crowdsourcing framework for software localization
Abstract

Crowdsourcing is gaining popularity as a distributed problem-solving model to provide solution to complex problems by coordinating between machines and human beings working together. The aim of the project is to design and develop a framework for software localization using crowdsourcing. The strategic objectives of this project include reducing cost, speeding up the localization process, dealing with heterogeneous workload and increasing the capacity to perform localization to new languages while maintaining quality. Quality is a major challenge in crowdsourcing systems. We overcome this challenge by using a quality control mechanism comprising multiple levels of verification, and intuitive ranking and rewarding algorithms. In order to achieve speed and scalability we adopt concepts from MapReduce framework. The preliminary results of our prototype implementation are positive and indicate the viability of the framework’s use in a real world industrial process.
Acknowledgment

I am grateful to everyone from UPC and CA Technologies, who supported me throughout my thesis work. I would specially like to thank Victor Muntés for the motivation and guidance through the project, Josep Lluís Larriba Pey for the supervision, and Patricia Paladini Adell for providing support from CA.
I would also like to thank Frederik Mijnhardt and Andrea Gritti for their help in implementation and testing of the project, and Sari Setianingsih for reviewing the report and providing helpful comments.
I am grateful to the EMDC coordinators Leandro Navarro and Johan Montelius for making these two years of Master’s degree memorable. I feel fortunate to have this wonderful EMDC family (Mariano, Kiarash, Erisa, Vasia, Sari, Nick, Wasif, Marcus, Navaneeth, Lalith, Ying, Zhenhua, Paulo).
Finally, I thank my parents, my brother and my fiancée for their love and support during my stay abroad.

Barcelona, June 2011

Jawad Manzoor
Contents

1 Introduction ........................................... 3
   1.1 Motivation ........................................... 3
   1.2 Problem Area ....................................... 4
   1.3 Solution ............................................. 6
   1.4 Contribution ....................................... 6
   1.5 Context ............................................. 7
   1.6 Organization of thesis ............................ 7

2 Related Work ......................................... 9
   2.1 Industrial applications of crowd computing .......... 9
   2.2 Crowdsourcing complex tasks ......................... 10
   2.3 Machine translation evaluation using crowd .......... 11
   2.4 Creating parallel corpora using crowd ............... 11
   2.5 Collaborative Translation ............................ 12
   2.6 Achieving professional quality translation from non-professionals .... 12
   2.7 Proofreading and editing using crowd ............... 12
   2.8 Discussion of related work .......................... 13

3 Quality Control and Scalability ......................... 15
   3.1 Two levels of quality control ....................... 15
      3.1.1 Postedit AV-Unit ................................ 16
      3.1.2 Review AV-Unit .................................. 17
   3.2 Worker selection algorithm .......................... 18
      3.2.1 Posteditor/Reviewer selection .................... 18
      3.2.2 Verifiers selection ................................ 18
   3.3 Ranking Algorithm and Trustworthiness ............... 19
      3.3.1 Post editor/Reviewer ranking ..................... 19
      3.3.2 Verifier ranking .................................. 20
      3.3.3 Exponential Histogram for ranking ............... 22
      3.3.4 Trial period ..................................... 23
   3.4 Rewarding mechanism ................................ 23
   3.5 Cost Reduction ...................................... 24
   3.6 Scalability ......................................... 25

4 System Design ........................................ 27
   4.1 Architecture ....................................... 29
List of Figures

1.1 Taxonomy of crowdsourcing ........................................ 4
1.2 General localization process illustration .......................... 5

2.1 Amazon MTurk interface ........................................ 10
2.2 Active Crowd Translation Framework ............................... 11
2.3 A sample of Urdu to English translation .......................... 13
2.4 Find-Fix-Verify pattern .......................................... 13

3.1 Action Verification Unit ........................................ 15
3.2 Two-step quality control mechanism ............................... 16
3.3 MapReduce model for crowdsourcing .............................. 26

4.1 Crowd-based localization process illustration .................. 27
4.2 Crowdsourcing framework architecture ........................... 28
4.3 Entity Relationship Diagram ..................................... 33

5.1 Home page design ................................................ 38
5.2 Model View Controller Architecture ............................... 39
5.3 Web flow of initial assessment ................................... 41
5.4 Web flow of admin modules ..................................... 42
5.5 Web flow of main application .................................... 43
5.6 Post editing interface ............................................. 45
5.7 Post editing interface with verifiers' feedback .................. 46
5.8 Verification interface ............................................. 46

6.1 Assessment test part 1 ........................................... 52
6.2 Assessment test part 2 ........................................... 53
6.3 Assessment test part 3 ........................................... 53
6.4 Individual scores of monolingual users ......................... 54
6.5 Individual scores of bilingual users ............................. 54
6.6 Histogram of users' scores ...................................... 55
6.7 Histogram of users' scores by category ......................... 57

7.1 Experiment Design .............................................. 61
# List of Tables

3.1 Description of error types ................................................. 17
3.2 TQI and quality levels .................................................. 20
3.3 Maximum error rates for different ranks ............................. 21
3.4 Error selection mechanism .............................................. 21
3.5 Post editor and Verifiers’ payment scheme ......................... 24
3.6 Reviewer and Verifiers’ payment scheme ............................ 24

4.1 Description of entities ................................................... 31

6.1 Detailed score of all users .............................................. 55
6.2 Detailed score of professionals ....................................... 56
6.3 Detailed score of students ............................................. 56
6.4 Detailed score of amateurs ............................................ 56

7.1 Distribution of workers in each phase ............................... 60
7.2 Hypothetical ranks of each category ................................ 60
7.3 Groups of verifiers ...................................................... 60
Chapter 1

Introduction

Crowdsourcing is a distributed problem-solving model in which a job is outsourced to a large group of unknown people called the crowd, through an open call [1]. The crowd returns the solutions to the job provider called crowdsourcer, who usually rewards them through a micropayment method. Achieving large-scale distributed computation in this manner is called crowd computing. The concept of "human computation" was proposed by Luis von Ahn and his colleagues in 2003, which exploits natural human abilities to compute tasks that are difficult for computers. Later, the term "crowdsourcing" was coined by Jeff Howe in 2006 [1][2]. Crowdsourcing has numerous applications these days, ranging from applications like voting system, social games and information sharing systems to algorithms, quality management and dataset creation. Figure 1.1 shows the taxonomy of crowdsourcing [3]. In this work we investigate crowdsourcing for industrial software localization [4].

1.1 Motivation

Software industry is experiencing rapid growth. There is a myriad of different businesses in the software industry, including operating systems, middleware, databases, enterprise software and security software. The global software market is forecasted by Data Monitor [5] to have a value of almost US$ 457 billion in 2013, which is an increase of 50.5% since 2008. In order to make the software globally available, the software industry cannot rely solely on English language because only 1.8 billion people out of the world’s total population of 7 billion speak English. Hence, for exploring new markets, the companies need to translate their software (user interface, websites, user manuals, etc) into several languages. The process of translating software into other languages is called software localization (L10n). Other terms used for this process are internationalization (i18n) and globalization. It usually involves two main tasks: (i) Translating the text that appear in the user interface of a product, and (ii) Translating the user manuals. This aspect becomes particularly important in an attractive but multi-lingual market like Europe. Software
1 Introduction

1.2 Problem Area

Software localization costs represent a barrier to entry for many software development companies. It is also a very time-consuming process. To speed up this process, the use of automatic Machine Translation (MT) systems has become prevalent. Despite years of research and improvement in MT, the quality of MT is still far from satisfactory. For this reason the companies cannot rely solely on MT for software localization. Good quality of translation is essential because an inaccurate translation may create doubts in people’s minds about the quality of the software product and it may also create a bad image of the company. Another drawback of a low quality translation is that the end user might misunderstand the guidelines and take wrong actions, thus, ending up in unpleasant results. To ensure good quality of translation, the industrial software localization process usually involves a hybrid approach that combines MT systems and human translators. It is a two-phase process. In the first phase an automatic MT is performed and in the second phase human translators are used for post-editing the machine-translated text. A general overview of the localization process is displayed in Figure 1.2. It involves two phases that are described next.
1.2 Problem Area

(i) Automatic translation phase: In this phase, the source text is translated into the target language using automatic MT. This step is fast, but the output is not of good quality.

(ii) Post-editing phase: In this phase, the MT engine output is manually reviewed and edited by human translators, in order to produce high quality publishable content. This is usually the most time-consuming phase in the localization process.

The use of MT reduces the time by roughly 50%. For example, in the case of software user guides translation, a human translator is able to translate 2000 to 2500 words per day, but this rate is doubled to 5000 words per day when MT is used and human translators do the post-editing phase. However, there are still many limitations of this process. In this project we study the localization process of CA Technologies Inc. and solve its limitations. The company is currently using the localization process explained above. The main limitations of this process are described below:

1. Long time-to-market periods for non-English versions of the products
   Products translated to other languages are usually released several months after the English version because the localization process is time-consuming.

2. Changing workload management
   The translation workload is heterogeneous and there are some peaks during the year when a large number of products are released together. Hence, the localization teams cannot cope with the situation forcing the outsourcing of part of the work to external translation service providers.

3. High cost of extending to countries speaking languages that are currently not translated
   In order to translate to a large number of languages, and open the possibilities for the company to explore new markets, the current approach does not work. Firstly, it is not easy to find translators for all languages. Secondly, it is financially expensive and thus infeasible in general to hire a team of translators for every language, especially for emerging markets in some countries where the number of products sold is not expected to be huge.

4. High cost of software localization
   Companies invest several million dollars in localization per year both in internal and outsourced localization. This causes several products not to be considered for internationalization even for common languages such as Spanish, German or French. To overcome these limitations we need a software localization system that is fast,
elastic, scalable and economical and gives high quality results.

1.3 Solution

We propose a new framework that solves the software localization process in a fast and cost-efficient way, scales up and down to deal with heterogeneous workload and provides excellent quality. To achieve these properties, we investigate the use to crowdsourcing for software localization process. Crowdsourcing is an effective technique for solving tasks that are hard for machines but easy for humans. Language translation is an excellent example of such tasks. MT is one of the hardest research problems of Natural Language Processing. However, multilingual human translators do this task very easily. Therefore, we have decided to exploit the power of the crowd for translation. Crowd computing provides elastic, on-demand, low cost human workforce in the same way as cloud computing provides computation resources. To exploit the potential of hundreds of thousands of multilingual speakers around the world, we propose an approach that draws concepts from distributed computing. A crowdsourcing market can be viewed as a large distributed system where the people/workers in the crowd act as processing units and solve the tasks requiring human intelligence. Some of the key challenges of distributed computing include the partitioning of tasks into smaller chunks that can be executed in parallel, assigning these chunks to processing nodes, handling dependencies, maintaining quality and ensuring reliability and scalability. Crowd computing also faces the same challenges therefore, we apply some solutions of managing distributed computing systems to crowdsourcing.

1.4 Contribution

We build a crowdsourcing framework on the concepts of distributed computing, which combines human intelligence for performing industrial software localization. Specifically, our framework aims to support:

- Task partitioning and automatic allocation to workers.
- Scalability
- Quality control mechanism
- Worker ranking algorithms
- Efficient rewarding mechanism
We present two elements that we consider essential for quality. First, we guarantee high quality of translation by proposing a comprehensive quality control mechanism. We call this mechanism the Action-Verification Unit (AV-Unit). It is a relationship pattern in which one worker performs an action/job that is verified by one or more workers from the crowd. Second, we propose ranking and rewarding systems that motivate workers by increasing their ranks when they deliver high-quality work, which in turn results in higher rewards for their work. We provide a prototype implementation using J2EE technology and a preliminary experimental evaluation.

1.5 Context

This project is done in collaboration with CA Labs Europe, which is the internal research group of CA Technologies Inc. (www.ca.com) and is in charge of fostering innovation and research in the company. CA Technologies is one of the largest independent software corporations in the world and maintains 150 offices in more than 45 countries. The company supports software which runs in mainframe, distributed, virtualized and cloud environments and is installed by a majority of the Forbes Global 2000. CA Technologies has been involved in software localization activities for more than 10 years, both from the user and the developer perspective, developing language translation technologies and tools to support translation processes. CA Technologies has a localization team in Barcelona that is in charge of translating both software and user guides to commercialize its products in Europe.

CA Labs has succeeded in acquiring funding for a European project to build and launch this crowd-based system in 2012-13. Our research paper on the current work titled 'Crowdsourcing for Industrial Problems' has been accepted at the First International Workshop on Citizen Sensor Networks at European Conference on Artificial Intelligence (ECAI 2012).

1.6 Organization of thesis

The rest of the thesis organized as follows. Chapter 2 discusses the related work in the field of crowdsourcing for industrial applications and translation. In Chapter 3 we present the quality control mechanism, ranking and rewarding algorithms, and scalability of the system. In Chapter 4 we discuss the design of the proposed system. Chapter 5 presents the implementation details. In Chapter 6 we explain the experimental evaluation and results. Finally, Chapter 7 provides conclusion and the future work.
Chapter 2

Related Work

2.1 Industrial applications of crowd computing

Crowd computing has been widely used in the industrial applications ranging from graphics designing to software testing to research and development. CrowdFlower is a major enterprise crowdsourcing platform founded in 2007. It offers crowdsourcing services including product categorization, business lead verification, content creation and many more. The quality is assured by using gold standard units and redundant reviews of each data unit. Their workflow management system divides complex tasks into smaller units and distributes them among the crowd. It also fits the crowd to the right job based on their profile.

uTest offers a huge labor pool with software testing skill set. The company offers testing services including functional, security, load, localization and usability testing. Founded in 2007, it currently has a community of 50,000+ professional testers from 180 countries. Many tech giants including Google, Microsoft, Intuit, and the British Broadcasting Corporation (BBC) are among its clients.

InnoCentive is a R&D related problem solving marketplace that brings together solution seeking companies and problem solvers dispersed all over the world. A seeker posts the problem description and the award and the registered solvers can read it and submit the solution. If the seeker finds a feasible solution, the provider of that solution receives the award. Some of the leading organizations including Eli Lilly & Company, Life Technologies, NASA, Nature Publishing Group, Popular Science and Procter & Gamble use the services of InnoCentive for innovation and problem solving.

Amazon Mechanical Turk (MTurk) is a crowdsourcing marketplace that enables companies or individuals to utilize the human intelligence to perform tasks that are difficult for computers. The requesters post tasks known as HITs (Human Intelligence Tasks) that can be viewed by workers. Requester can also put certain restrictions on qualifications that a worker needs to possess in order to take up a HIT. The requester can also accept or reject the results of a worker. The HITs on MTurk are usually simple and self-contained tasks such as identifying objects in a photo or video, providing captions to images etc. The reward for the HITs typically range from 1
2 Related Work

The MTurk API enables requesters to submit HITs and retrieve results. rMTurk interface is shown in Figure 2.1.

![Figure 2.1: Amazon MTurk interface](image)

2.2 Crowdsourcing complex tasks

Crowdsourcing markets are traditionally used for simple and independent tasks. For example, labeling and image or finding relevance between search results. CrowdForge is a framework that enables accomplishing complex and interdependent tasks using crowdsourcing markets. In [13] three case studies are provided. The experiments were performed using Amazon Mechanical Turk. The first case study is article writing. In this case study the workers collaborate using the framework to write a complete article on a topic by contributing a small portion. The second case study is decision making in which the framework is utilized to commission decision matrices to help consumers compare automobiles. The third case study is about science journalism in which a scientific paper published in an academic venue is turned into a newspaper article for the general public. The authors follow an approach similar to MapReduce for breaking down a complex problem into a sequence of simpler subtasks. TurKit [14] is a toolkit for exploration of new ideas in human computation algorithms on MTurk. The programming model of TurKit allows iterative workflows and enables tasks like editing text. It abstracts MTurk as a function call. TurKit uses crash-and-rerun programming model to overcome the issue of lengthy and time consuming tasks on MTurk. It is related to scripting languages for distributed systems, which also handle computations stretched over time.
2.3 Machine translation evaluation using crowd

In [15] the authors investigate the use of crowdsourcing for evaluation of Machine Translation for natural language processing. They hire crowd on Mechanical Turk for evaluating the output of statistical machine translation. The case study is English to Arabic translation. The workers are presented with pairs of sentences in source and target language and are asked to vote the translations as excellent, good, bad or very bad. The authors present six different normalization techniques to evaluate their relative performance including straight average, removing low-agreement judges, removing outlying judgments, weighted voting, scaling judgments and 2-stage scaling technique. In [16] the authors hire non-experts through the CrowdFlower interface to Amazon Mechanical Turk and perform evaluation of English to Arabic machine translation. The experiment is performed using both experts and non-experts and the analysis of the results shows that agreement rates for non-experts are comparable to those obtained for experts.

2.4 Creating parallel corpora using crowd

Corpus-based approaches to machine translation have become prevalent in the recent years. These approaches require parallel corpora for the source and target language pair. Therefore, this approach is not suitable for low-resource language pairs. In [17] the authors present Active Crowd Translation (ACT) framework that combines active learning and crowdsourcing. In this framework the power of the crowd is used as an alternative for expensive language experts. With the experimental results it is shown that it is possible to create parallel corpora using non-experts by providing sufficient quality assurance. The framework is shown in Figure 2.2.

![Figure 2.2: Active Crowd Translation Framework](image)
2.5 Collaborative Translation

Microsoft Research developed the Collaborative Translation Framework that utilizes community translators as MT posteditors. The framework is a mixture of translation memory, machine translation and wiki-style web publishing. The process involves three steps. In the first step the text goes through translation memory (TM), which contains mapping of already translated strings of text from source to target language. The segments of text that are not covered by TM go to step two where MT is applied using Bing Translator, which provides free translation to 35 languages in the cloud. After this step the content is published to the web. Step three is the key step in which the community translators postedit the text on the web. There is an option to publish community edits based on translation ranking. An API is provided for developers to integrate collaborative translation services into existing applications. A drawback of this approach is that there is no quality control mechanism. Alternatively, monitoring each and every community edit for ensuring quality is a bottle-neck and is practically infeasible.

2.6 Achieving professional quality translation from non-professionals

In [18] the authors demonstrate mechanisms to ensure high quality of translation by non-professional translators. The technique involves creation of redundant translations and selecting the best output among them. Some factors are proposed to select a good translation among a set of different versions. These factors include the workers’ country of residence, native language, edit rate from the other translations etc. Each factor has a different weight. In this way the total score is calculated for a specific translated text. The experiment involved Urdu to English translation and the set consisted of 1,792 Urdu sentences from a variety of news and online sources. A sample translation paragraph is shown in Figure 2.3. Experimental results showed that some translations turn out to be very close in quality to the ones made by professional translators. However, it requires high level of redundancy which will result in very high cost if used commercially as an industrial process.

2.7 Proofreading and editing using crowd

Soylent[19] is a plugin for MS Word that crowdsources the text to workers on Mechanical Turk for editing, proofreading or shortening. The authors propose Find-Fix-Verify pattern to perform these tasks. They split the tasks into a series of stages that utilize independent agreement and voting to produce reliable results. The flow of this process is shown in Figure 2.4.
2.8 Discussion of related work

There are several limitations of currently available crowdsourcing systems for translation and text improvement. Some\cite{18}\cite{19} use very small tasks, mostly at sentence level and their entire quality assurance methodology is based on it. Others \cite{18}\cite{16}\cite{15} use a golden translation for evaluation. Such systems are not suitable for implementing an industrial software localization process because of two main reasons. Firstly, by using sentence level tasks the coherence of the text as a whole is lost. Secondly, golden translation corpus is not available for less popular languages. We
propose a system that uses larger tasks of more than thousand words and provide quality assurance mechanism that involves multiple verification steps and does not depend on a golden translation corpus.

The system we propose is novel because it uses multiple levels of verification to ensure good quality, implements a quality-based ranking and rewarding system to provide incentives to workers to improve, and is built on the concept of MapReduce model to perform a large number of tasks in parallel.
Chapter 3

Quality Control and Scalability

Crowd computing is usually associated with low quality results. In general, it is difficult to establish automatic mechanisms to monitor quality in crowd-based systems. Judging quality involves subjectivity, therefore we propose to crowdsourcing quality verification also. The basic idea of our quality control mechanism is subdividing any complex task in smaller subtasks and passing each subtask through a series of Action-Verification Units (AV-Unit). An AV-Unit establishes a relationship pattern between the workers in the crowd to help them work collaboratively to provide a higher degree of quality. One worker performs an action that is verified by one or more workers. The verifiers find mistakes in the results and give their feedback that is received by the worker in Action phase, who resubmits the improved version of the work. Hence, the Verification phase in the AV-Unit acts as a quality barrier that does not allow the result to proceed to the next step until the quality of current step is sufficient. This is shown in Figure 3.1.

3.1 Two levels of quality control

We propose two AV-Units to ensure quality of translation. First is the Postedit AV-Unit that involves bilingual workers and second is the Review AV-Unit that

![Figure 3.1: Action Verification Unit](image)
3 Quality Control and Scalability

Figure 3.2: Two-step quality control mechanism

Involves native speakers. This is shown in Figure 3.2.

There are three types of workers involved in the process.

1. Posteditor - The job of a posteditor is to edit and improve the machine translated text. The post editors are bilingual because they read the text in source language and edit the MT text to provide improved translation in target language.

2. Reviewer - The job of a reviewer is to proofread and improve a given text in target language. The reviewer has to be native speaker of target language.

3. Verifier - The job of a verifier is to verify and evaluate the work done by a post editor or reviewer by point out mistakes in it and providing comments on how to improve it. The verifiers of posteditors are bilingual and those of reviewers are monolinguals.

3.1.1 Postedit AV-Unit

In the first phase a chunk of text is assigned to a worker for post-editing. The worker is shown the original text and the machine translated text that is editable. The posteditor edits and improves the machine-translated text. In the second phase the post-edited text is sent to one or more workers for verification. They are shown the original text along with the post-edited text. They check for errors in the post-edited text and provide comments for improving it. If the error rate above the threshold, no further steps are carried out and a request is sent for re-execution of this chunk. If the error rate is below the threshold, the text with highlighted errors and comments is sent back to the posteditor of first phase for revision. Posteditor corrects the errors and submits the revised text. We have defined nine error types for postedit phase.
3.1 Two levels of quality control

shown in Table 3.1.

<table>
<thead>
<tr>
<th>Error type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mistranslation</td>
<td>The target language does not accurately reflect the meaning of the source text. This may include ambiguously or literally translated passages if the meaning of the original text is lost or altered.</td>
</tr>
<tr>
<td>Omission/ Addition</td>
<td>Source text information has been deleted from the target text, or information not found in the source text has been added to the target text.</td>
</tr>
<tr>
<td>Inconsistent terminology</td>
<td>Inconsistent terminology within a product or does not follow generally accepted industry standards</td>
</tr>
<tr>
<td>Grammar</td>
<td>The translation does not adhere to the target language-specific rules with regard to grammar.</td>
</tr>
<tr>
<td>Style</td>
<td>The translation does not adhere to CA style guidelines and/or any other specifications provided.</td>
</tr>
<tr>
<td>Punctuation</td>
<td>The translation does not adhere to the target language-specific rule with regard to punctuation.</td>
</tr>
<tr>
<td>Software options</td>
<td>The user interface elements (names or menu options, windows or dialog boxes, etc.) used in the translation are not identical to the glossary or not respected capitalization as indicated in the guidelines.</td>
</tr>
<tr>
<td>Untranslatable text</td>
<td>Untranslatable variables (html tags, product names etc) or any other content indicated in the guidelines is translated.</td>
</tr>
<tr>
<td>Typographical error</td>
<td>The word does not comply with specific language regarding spelling. This error category includes typographical errors or misprints: unneeded spaces between words, omission of letters or order of the letters wrong, etc.</td>
</tr>
</tbody>
</table>

3.1.2 Review AV-Unit

In the second phase the post-edited output of a chunk from Postedit AV-Unit is assigned to a reviewer. The reviewer who is the native speaker of the target language checks for grammatical, structural and fluency errors and corrects them. The reviewed text is assigned to two or more verifiers in the second phase. They check for errors in the reviewed text and provide comments for improving it. The text with highlighted errors and comments is sent back to the reviewer of first phase for revision. After revision we have the final version of the text. The errors for review
3 Quality Control and Scalability

phase are the same as post edit phase except mistranslation and omission/addition errors.

3.2 Worker selection algorithm

3.2.1 Posteditor/Reviewer selection

Post editor or reviewer is selected for a job after an auction. There are different ways in which the auction can be implemented. We discuss each of them along with the pros and cons.

1. **Handpick the best workers:** In this policy we keep the list of available workers in descending order based on rank. When a job is posted we send offers to the top ranked workers and wait for their acceptance. This policy is very good for getting high quality work from the best workers but it conflicts with the usual way in which auctions works. It also takes a longer time for sending proposals and waiting for acceptance. Another drawback is that it causes a lot of incoming traffic for the workers because they are notified whenever a new job is available.

2. **Perform an auction and select the best workers:** In this policy we give an open call when a new job is available, and the workers apply for it. After receiving the requests, we select the top-ranked workers for the job and leave the others. It is faster and gives high quality results but the low-ranked workers suffer from starvation and are never assigned a job.

3. **Perform and auction and select workers on FCFS basis:** In this policy we give an open call for a new job and the workers who apply for it are selected on first-come, first-served basis, irrespective of their rank. This policy results in a fair distribution of work among all workers. However, we do not select the best workers for the job. The quality control mechanism takes care of this and ensures high quality. We have adopted this strategy for the selection of post editors and reviewers.

3.2.2 Verifiers selection

The auction for all subtasks is performed and the workers are queued on FCFS basis. Since there are more than one verifier for a post editor or reviewer, we select the number of verifiers based on the available budget for the task. The algorithm works like this. As an example, we allocate a budget of 2 cents per word for and AV-Unit. The system will start selecting workers from the queue and add their cost to the total cost. When the total cost equals the allocated budget, the selection is stopped. If at any time during this process the cost accedes the budget while selecting a worker, that worker is skipped until a next worker with lower cost is found that satisfies the budget. The pseudocode is shown below:
3.3 Ranking Algorithm and Trustworthiness

The worker cost is retrieved from Table 3.5 and Table 3.6. This cost is based on the rank of the worker. The above process is repeated for each subtask from the beginning of the queue. Therefore, the workers skipped during previous subtask, get a chance to get selected for the next partition.

3.3 Ranking Algorithm and Trustworthiness

We have developed two ranking algorithms: one for post editors and reviewers, and one for verifiers. Our ranking module updates the rank of each worker after the completion of every job by that worker. A high quality output by a worker results in an increase in rank and a low quality output results in a decrease in rank. The purpose of using such a ranking system is to establish trustworthiness. A worker who constantly delivers high quality output will keep going higher in rank and vice versa. Thus, the rank of a user represents the trustworthiness and it directly affects the amount of reward the worker will get for a job. Therefore, we have carefully developed the ranking algorithms, which are explained next.

3.3.1 Post editor/Reviewer ranking

The post editor and reviewer are given points after each job completion. The number of points earned, depend on the number of mistakes done during the job. These mistakes are found by the verifiers. To ensure a minimum level of quality we set the Permitted Error Rate (PER) as 15%. The Worker Error Allowance (WEA) is the maximum number of errors allowed to a worker to achieve a passing score, and is calculated by the following formula.

\[ WEA = PER \times wordCount \]

**Example:** A worker receives a job of 1000 words. PER is 15%, which gives us WEA=150. So, the worker can make a maximum of 150 mistakes in 1000 words.
The work of a post editor or reviewer having error rates greater than PER is rejected. We have set 100 as maximum score and 40 as passing score for a job. These values have been decided in the light of the current policies implemented in CA Technologies. We can calculate the Translation Quality Index (TQI) of a job by the following formula.

\[
TQI = 100 - \left( 60 \times \frac{TotalError}{WEA} \right)
\]

**Example:** A worker submits a job of 100 words. The verifiers find a total of 15 errors. WEA is also 15.

\[
TQI = 100 - \left( 60 \times \frac{15}{15} \right) = 40
\]

The worker has used all of the allowed error allowance, therefore TQI has the minimum value of 40. In another scenario the verifiers find only 5 errors and WEA is 15.

\[
TQI = 100 - \left( 60 \times \frac{5}{15} \right) = 80
\]

The TQI represents the rank of a worker for a particular job and it relates to the quality levels in Table 3.2.

<table>
<thead>
<tr>
<th>TQI value</th>
<th>Quality level</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;90</td>
<td>Excellent</td>
</tr>
<tr>
<td>80 to 90</td>
<td>Very Good</td>
</tr>
<tr>
<td>70 to 80</td>
<td>Good</td>
</tr>
<tr>
<td>60 to 70</td>
<td>Acceptable</td>
</tr>
<tr>
<td>50 to 60</td>
<td>Borderline</td>
</tr>
<tr>
<td>40 to 50</td>
<td>Poor</td>
</tr>
<tr>
<td>&lt;40</td>
<td>Reject</td>
</tr>
</tbody>
</table>

Table 3.3 gives a mapping of error rate to the rank of the worker. E.g. a worker can achieve a rank greater than 90 by making a maximum of 2% mistakes.

### 3.3.2 Verifier ranking

The verifiers earn points based on the mistakes that they find in the text. When a verifier finds a mistake in a statement, that statement is flagged as 'possibly erroneous' with weight 'w' equal to the rank of the verifier. Similarly, if another
### 3.3 Ranking Algorithm and Trustworthiness

Table 3.3: Maximum error rates for different ranks

<table>
<thead>
<tr>
<th>Post Editor/Reviewer Rank</th>
<th>Error Rate %</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;90</td>
<td>&lt;=2</td>
</tr>
<tr>
<td>80 to 90</td>
<td>3 to 4</td>
</tr>
<tr>
<td>70 to 80</td>
<td>6 to 7</td>
</tr>
<tr>
<td>60 to 70</td>
<td>8 to 9</td>
</tr>
<tr>
<td>50 to 60</td>
<td>10 to 12</td>
</tr>
<tr>
<td>40 to 50</td>
<td>12 to 15</td>
</tr>
<tr>
<td>&lt;40</td>
<td>&gt;15</td>
</tr>
</tbody>
</table>

verifier also finds the same mistake, the rank of this verifier is also added to the previous weight of the error. An error needs to have a weight \( w \geq 50\% \) of the sum of ranks of all verifiers working on the job, in order to be considered as "definitely erroneous".

**Example:** Suppose we have verifiers A, B and C with ranks 90, 50, and 40 respectively. The sum of ranks is 180 and 50% of the sum is 90. If verifier A finds an error then its weight will be 90 and hence, it will always be considered as "definitely erroneous". However, if verifiers B or C find different errors, they will only have a weight of 50 and 40, both of which are less than 90, therefore they will only be considered as "possibly erroneous". However, if B and C find the same error then it will have a weight of 50+40 = 90 and hence it will be considered as "definitely erroneous". This is shown in Table 3.4.

Table 3.4: Error selection mechanism

<table>
<thead>
<tr>
<th>Error number</th>
<th>Marked by Verifier</th>
<th>Total weight</th>
<th>Flag</th>
<th>Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
<td>90</td>
<td>Definitely Erroneous</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td></td>
<td>50</td>
<td>Possibly Erroneous</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>X</td>
<td>40</td>
<td>Possibly Erroneous</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td>X</td>
<td>90</td>
<td>Definitely Erroneous</td>
</tr>
</tbody>
</table>

In this way the workers with higher ranks have greater weight for their vote as compared to the workers with lower ranks. Verifiers get points according to their contribution to the total number of 'definitely erroneous' statements found in the text, according to the following formula:

\[
VR_{score} = \left( \frac{\text{maxpoints}}{\text{errorRate}} \right) \times \text{contribution} + \left( \frac{1}{\text{errorRate}^3} \right) \times 10^3
\]
3 Quality Control and Scalability

The maximum available points for any task are maxpoints=100. If the verifier points are greater than this value, they are capped at 100; i.e. the verifier gets full points. In order to keep a check on the malicious workers who randomly mark statements as erroneous, we allow a maximum deviation value of 3%. After this value the verifier faces a penalty of VR Penalty (VRP) of 3 points for each 1% of excess deviation (ED). The deviation value is the number of statements marked by a verifier as "possibly erroneous" that do not qualify in the end as "definitely erroneous". If a text has 100 statements and a verifier finds mistakes in 5 statements that are not in agreement with majority then he gets penalized for 2% i.e. 6 points. Now the formula becomes:

\[
VRscore = \left[ \left( \frac{\text{maxpoints}}{\text{errorRate}} \right) \times \text{contribution} \right] + \left[ \left( \frac{1}{\text{errorRate}^3} \right) \times 10^3 \right] - (ED \times VRP)
\]

**Example:** Suppose we have a text with 100 statements. A verifier finds 4 mistakes in agreement with majority and 4 with disagreement, and the total mistakes in the text are 6.

- maxpoints = 100
- errorRate = 6%
- contribution = 4%
- deviation = 4%

\[
VRscore = \left[ \left( \frac{100}{6} \right) \times 4 \right] + \left[ \left( \frac{1}{6^3} \right) \times 10^3 \right] - (1 \times 3) = 66.66 + 4.62 - 3 = 68.2
\]

3.3.3 Exponential Histogram for ranking

A straightforward method of calculating overall rank is to take average of all the jobs done by the worker in her lifetime. With this method the worker gets equal weight for each job she has completed. An alternative method is to use exponential histogram to calculate average rank. The purpose of using exponential histogram to calculate the overall rank of the worker is to give more weight to her recent jobs and less to the oldest ones.

We put the job scores in \( k \) buckets, each of size \( 2^i \). A bucket has a timestamp of the newest element and the sum of the number of elements in it. Initially each bucket has size of 1. When the number of buckets increases from \( k \), the oldest 2 buckets with smallest size are compressed into 1 bucket.

To calculate the rank of a worker we calculate the average of all buckets.
3.4 Rewarding mechanism

for all buckets k
    sum += bucket.value

averageRank = sum/k

3.3.4 Trial period

When a new worker joins the system, the first 10 jobs done by her are in trial mode. In ranking gained by a user in trial mode in not carried forward. This allows workers to get used to the system and they get a chance to get rid of the bad rank they got due to lack of proper understanding of the system in the beginning. The system starts the real ranking of the worker once the trial mode is over. If a worker falls below the rank of 40 at any time after trial mode, she is flagged as a malicious worker and is not assigned any task.

3.4 Rewarding mechanism

Crowd-based systems can be broadly classified into two categories based on the rewarding model they use.

Best-gets-paid model: In this model, only the best workers get rewarded. The requester uploads a job and the crowd provides solutions to it. The requester selects the best result pays the worker who provided that solution. The best solution may also be selected through voting from the crowd. This model usually allows to reduce costs as well as obtain good quality. However, it is unfair in a sense that most of workers so the job and are not rewarded. Many crowd-based systems ranging from logo designing, t-shirt designing to R&D solutions use this model.

Pay-per-work model: In this model, all the workers get rewarded according to the amount of work done by them. This is the most common model for crowd-based system with a large number of independent and small tasks. Amazon Mechanical Turk also uses this model where each task has a predefined award. The workers perform these tasks and get the reward in case of successful completion.

Pay-per-quality model: We propose a new rewarding mechanism where workers get paid according to the quality of work they do. The tasks have a predefined minimum amount that a worker will get plus a bonus amount that is directly proportional to the rank of the worker. The higher the rank, the greater the bonus. It means that the same task done by two workers having different ranks will have different payments. This model motivates the workers to provide good quality work, which will result in an increase in rank and in turn increase in the amount of payment. This claim is explained in [20] where the authors show that financial incentives actually encourage quality.

The amount of payment per word for different ranks and post editors and their verifiers are shown in Table 3.5 whereas Table 3.6 displays these amounts for reviewers.
and their verifiers. These values have been decided in such a way that they are lower than the amounts usually paid to professional translators in companies, and higher than the amounts paid to crowd in systems like Amazon MTurk. It is important to note that these values are very general and they are not based on extensive survey of the current market.

The amounts vary depending on the level of complexity of the job. In this case the most crucial step is post editing. It involves two languages and the post editor has to correctly rephrase the machine-translated text. Post editor verification job is relatively simpler. Review and reviewer verification have a low level of complexity as they have to read the text only in target language and they don’t have to deal with translation issues.

<table>
<thead>
<tr>
<th>Post Editor Rank</th>
<th>Pay Per word (cents)</th>
<th>Verifier Rank</th>
<th>Pay Per word (cents)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;90</td>
<td>2</td>
<td>&gt;90</td>
<td>1</td>
</tr>
<tr>
<td>80 to 90</td>
<td>1.5</td>
<td>80 to 90</td>
<td>0.75</td>
</tr>
<tr>
<td>70 to 80</td>
<td>1.25</td>
<td>70 to 80</td>
<td>0.625</td>
</tr>
<tr>
<td>60 to 70</td>
<td>1.0</td>
<td>60 to 70</td>
<td>0.5</td>
</tr>
<tr>
<td>50 to 60</td>
<td>0.75</td>
<td>50 to 60</td>
<td>0.375</td>
</tr>
<tr>
<td>40 to 50</td>
<td>0.5</td>
<td>40 to 50</td>
<td>0.25</td>
</tr>
<tr>
<td>&lt;40</td>
<td>-</td>
<td>&lt;40</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reviewer Rank</th>
<th>Pay Per word (cents)</th>
<th>Verifier Rank</th>
<th>Pay Per word (cents)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;90</td>
<td>1</td>
<td>&gt;90</td>
<td>0.75</td>
</tr>
<tr>
<td>80 to 90</td>
<td>0.75</td>
<td>80 to 90</td>
<td>0.625</td>
</tr>
<tr>
<td>70 to 80</td>
<td>0.625</td>
<td>70 to 80</td>
<td>0.5</td>
</tr>
<tr>
<td>60 to 70</td>
<td>0.5</td>
<td>60 to 70</td>
<td>0.375</td>
</tr>
<tr>
<td>50 to 60</td>
<td>0.375</td>
<td>50 to 60</td>
<td>0.25</td>
</tr>
<tr>
<td>40 to 50</td>
<td>0.25</td>
<td>40 to 50</td>
<td>0.1875</td>
</tr>
<tr>
<td>&lt;40</td>
<td>-</td>
<td>&lt;40</td>
<td>-</td>
</tr>
</tbody>
</table>

### 3.5 Cost Reduction

Taking mean values from Table 3.5 and 3.6 for all four phases gives us the total cost of roughly 3.5 cents per word. This is only an assumption for average case. At this rate, a 300 page user manual containing around 150 thousand words will be translated at a cost of $5250. This reduces the current cost of localization by roughly 50%.
3.6 Scalability

One of the key properties of crowd computing is elasticity. It allows access to very large human resource. Therefore, a crowd-based system needs to be able scale well and cope with large problems. To design a scalable crowd-based system we adopt concepts from distributed computing. First we define our workload and then find an appropriate technique to solve it. We divide each task (user manual) typically containing around 150,000 words into smaller chunks of 1000 to 2000 words. Our workload falls into the category of embarrassingly parallel problems because there is almost no dependency between the chunks.

MapReduce [21] is, perhaps the most popular framework for processing embarrassingly parallel problems across huge datasets using a large number of computers. A problem is initially divided into smaller sub-problems. In the Map step the sub-problems are distributed among the worker nodes, which process the sub-problems and send results to Reduce step. In the Reduce step the results of sub-problems are combined on some way to form the final solution to the original problem.

We follow the MapReduce framework concepts to achieve distributed computation using crowd. We distribute each chunk to a Map step. In the Map step two AV-Units are involved: Postedit AV-Unit and Review AV-Unit. Each Map step has a Map Controller. It has the following responsibilities:

- Assign a chunk to a posteditor
- Collect results from posteditor
- Assign verifiers to evaluate the results
- Send feedback to posteditor.
- Receive revised results from posteditor
- Assign chunk to reviewer
- Collect results from reviewer
- Assign verifiers to evaluate the results
- Send feedback to reviewer.
- Receive revised results from reviewer

After each Map step the result of one chunk is ready. In the Reduce step the results of each Map step (related to a certain task) are combined together to create the final version of the task (user manual) in target language. It is important to note here that Map and Reduce are not distributed computers, rather they are processes managing distributed workers. The MapReduce model for crowdsourcing is shown in Figure 3.3.
3 Quality Control and Scalability

Figure 3.3: MapReduce model for crowdsourcing
Chapter 4
System Design

Our proposed system introduces a crowd-based localization process to replace (in house or outsourced) professional translation teams. The crowd-based localization is illustrated in Figure 4.1.

We divide a translation task into smaller chunks and distribute them among the crowd. Each worker in the crowd has a particular role and a single chunk goes through many workers with different roles and finally gives a high quality output. Defining roles for the workers and using multiple steps of verification are essential to ensure good quality of translation.

Figure 4.1: Crowd-based localization process illustration
Figure 4.2: Crowdsourcing framework architecture
4.1 Architecture

The system consists the following high level modules:

1) **Task controller (TC)**
We define task as the biggest unit of data to be translated. For example, a user manual, set of user interface of a software etc. Whenever we want to translate a task, a TC is invoked. A TC is responsible for completing the whole translation process of one task.

2) **Task partitioner**
TC calls the task partitioner, which divides the task into multiple partitions based on predefined rules. E.g. a user manual may be partitioned by topics, a user interface by menu etc. The size of partitions is typically 1000 to 2000 words.

3) **Translation memory module**
The TC sends the partitions to translation memory module, which matches the text in the partitions with a database of previously translated texts. If an exact match of a string of text is found, it is translated and marked as read-only so that it cannot be modified. When the whole translation process is finished, the final product is fed into the translation memory for future use. With the passage of time, the translation memory grows bigger and bigger. As a result we get more matching strings of text in future and less amount of text to be translated.

4) **Machine translation module**
After the matching process, the partitions are returned to TC, which sends the partitions to machine translation module. The machine translation engine translates all the text in the partitions, except the read-only strings. Since this machine-translated text is not of high quality, we need humans to post edit it.

5) **Map controller (MC)**
The machine-translated texts are refined into the final product by employing MapReduce model using the crowd. One Map process is initiated for each partition of the task and one Reduce process is initiated for the entire task. A Map Controller (MC) is associated with each Map process that is responsible for the successful completion of the translation process of the partition. MCs request the Inter-Task Controller for workers. MC assigns partitions to workers and collects results from them.

6) **Inter-task controller (ITC)**
ITC initiates auctions for the jobs and selects workers based on first-come, first-served basis. Afterwards, ITC assigns the selected workers to MCs and updates the list of available workers. The ITC is responsible for performing auctions for jobs, allocating appropriate number of workers to jobs, and maintaining worker ranks. It maintains the lists of the workers along with their ranks. We have two lists of workers. First list is of the bilingual speakers who perform post editing and its verification. Second list is of the native speakers of the target language who review the post-edited text. The job of post editors and reviewers is to provide the correct translation and the job of verifiers is to find out errors in the jobs performed by the post editors or reviewers.
7) Worker ranking module (WRM)
The WRM updates the rank of a worker after every job completion by her. It receives input from MC and ITC about the performance of current job and the current rank of the worker respectively. There are several factors impact the ranking that are explained in section 3.5

8) Discussion forum
Discussion forum enables various workers working in the system to discuss and clarify the issues they face while performing a job.

9) Billing module
The billing module is connected to ITC and the worker and billing database. ITC notifies the billing module about a job completion by a worker and the billing module computes the payment amount based on the job award and worker rank.

The framework is shown in Figure 4.2.

4.2 Database Design

4.2.1 Entities and Attributes

The first step in designing a database is to identify the entities. The entities are represented by tables in the database. We have identified the entities in our system shown in Table 4.1.

After listing all the entities, the second step is to identify the attributes of each entity. The attributes represent the columns/fields of the tables. The third step is to form relationships between entities. The attributes of each entity and the relationship between the entities are explained in the Entity Relationship Diagram (ERD) of the system, which is shown in Figure 4.3.

4.2.2 Explanation of the entities and relationships

Entities - Task and Chunk
Relationship - One to many
Explanation - One task comprises many chunks. A task may be a big user manual with hundreds of pages of text, which is divided into smaller chunks.

Entities - Chunk and Unit
Relationship - One to many
Explanation - A Chunk can have four Units. A Unit can be one of the four types: PE_Unit, PEVE_Unit, RE_Unit or REVE_Unit. Their common attributes have been grouped into a super type. There can be no other type of Unit. This is indicated by the double line. Since the instance can only be one of these entities, the "d"
### 4.2 Database Design

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Task is an independent entity that has to be translated. E.g a software user manual.</td>
</tr>
<tr>
<td>Chunk</td>
<td>A task is divided into smaller chunks.</td>
</tr>
<tr>
<td>PE_Unit</td>
<td>Post Edit Unit contains two post-edited versions of the machine-translated unit. An initial version and an improved version.</td>
</tr>
<tr>
<td>RE Unit</td>
<td>Review Unit contains two reviewed versions of the post-edited unit. An initial version and an improved version.</td>
</tr>
<tr>
<td>PEVE_Unit</td>
<td>Post Edit Verify Unit consists of the errors found by verifiers in the job done by a post editor.</td>
</tr>
<tr>
<td>REVE_Unit</td>
<td>Review Verify Unit consists of the errors found by verifiers in the job done by a reviewer.</td>
</tr>
<tr>
<td>MT_Text</td>
<td>MT_Text contains the sentences of the original text in source language and machine-translated text of the Task.</td>
</tr>
<tr>
<td>PE_Text</td>
<td>PE_Text contains the sentences of the post-edited text of a PE_Unit.</td>
</tr>
<tr>
<td>RE_Text</td>
<td>RE_Text contains the sentences of the reviewed text of a RE_Unit.</td>
</tr>
<tr>
<td>PE_Errors</td>
<td>PE_Errors contains the sentences of post-edited text having errors, along with the error type and comments.</td>
</tr>
<tr>
<td>RE_Errors</td>
<td>RE_Errors contains the sentences of reviewed text having errors, along with the error type and comments.</td>
</tr>
<tr>
<td>Registration_Test</td>
<td>Registration_Test contains the answers of the workers for the initial test taken by them and the score of the test.</td>
</tr>
<tr>
<td>Payment</td>
<td>Payment consists of the amount of money payable to a worker for a job, along with the status of the payment.</td>
</tr>
</tbody>
</table>

(disjoint) is present in the bubble.

Entities - Unit and Worker  
Relationship - One to many  
Explanation - One unit can be assigned to many workers. There are multiple verifiers working on the same unit submitted by a post editor or a reviewer.

Entities - Worker and Payment  
Relationship - One to many  
Explanation - Since one worker can work on many jobs in the system, she can have many payments.
Entities - Worker and Registration_Test
Relationship - One to one
Explanation - A worker takes the registration test only once and gets an initial score based on the performance in the test.

Entities - Unit and Payment
Relationship - One to one
Explanation - One unit can have only one payment. A unit is the smallest object that a worker can work on, and has a fixed payment.

Entities - Worker and PE_Text
Relationship - One to many
Explanation - A worker works on many lines of text stored in PE_Text.

Entities - Worker and RE_Text
Relationship - One to many
Explanation - A worker works on many lines of text stored in RE_Text.

Entities - Worker and PE_Errors
Relationship - Many to many
Explanation - A worker may find many errors or may not find any error in a post edited text. On the other hand, the same error may be found by many workers.

Entities - Worker and RE_Errors
Relationship - Many to many
Explanation - A worker may find many errors or may not find any error in a reviewed text. On the other hand, the same error may be found by many workers.

Entities - Task and Text
Relationship - One to many
Explanation - A task consists of many lines of text.

Entities - Errors and Text
Relationship - Many to many
Explanation - A line of text may have many errors and one error may be in many lines of text.
Figure 4.3: Entity Relationship Diagram
4.2.3 Tabular form of entities

The tabular form of the entities is given below.

Worker Table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>worker_id</td>
<td>int(11)</td>
<td></td>
</tr>
<tr>
<td>f_name</td>
<td>varchar(500)</td>
<td>None</td>
</tr>
<tr>
<td>dob</td>
<td>varchar(15)</td>
<td>None</td>
</tr>
<tr>
<td>email</td>
<td>varchar(40)</td>
<td>None</td>
</tr>
<tr>
<td>password</td>
<td>varchar(100)</td>
<td>None</td>
</tr>
<tr>
<td>target_lang</td>
<td>varchar(25)</td>
<td>None</td>
</tr>
<tr>
<td>english_level</td>
<td>varchar(7)</td>
<td>None</td>
</tr>
<tr>
<td>category</td>
<td>varchar(20)</td>
<td>None</td>
</tr>
<tr>
<td>rank</td>
<td>int(11)</td>
<td>None</td>
</tr>
</tbody>
</table>

Chunk Table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>chunk_id</td>
<td>varchar(10)</td>
<td>Primary</td>
</tr>
<tr>
<td>task_id</td>
<td>int(11)</td>
<td>None</td>
</tr>
<tr>
<td>start_time</td>
<td>datetime</td>
<td>None</td>
</tr>
<tr>
<td>end_time</td>
<td>datetime</td>
<td>None</td>
</tr>
<tr>
<td>status</td>
<td>varchar(15)</td>
<td>None</td>
</tr>
<tr>
<td>binaryfile</td>
<td>blob</td>
<td>None</td>
</tr>
<tr>
<td>filename</td>
<td>varchar(50)</td>
<td>None</td>
</tr>
</tbody>
</table>

Unit Table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit_id</td>
<td>varchar(50)</td>
<td>Primary</td>
</tr>
<tr>
<td>task_id</td>
<td>varchar(12)</td>
<td>None</td>
</tr>
<tr>
<td>worker_id</td>
<td>int(11)</td>
<td>None</td>
</tr>
<tr>
<td>binaryfile</td>
<td>blob</td>
<td>None</td>
</tr>
<tr>
<td>filename</td>
<td>int(11)</td>
<td>None</td>
</tr>
<tr>
<td>start_time</td>
<td>datetime</td>
<td>None</td>
</tr>
<tr>
<td>end_time</td>
<td>datetime</td>
<td>None</td>
</tr>
<tr>
<td>status</td>
<td>varchar(15)</td>
<td>None</td>
</tr>
<tr>
<td>score</td>
<td>int(3)</td>
<td>None</td>
</tr>
</tbody>
</table>

PE_Text Table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>pe_text</td>
<td>int(11)</td>
<td>Primary</td>
</tr>
<tr>
<td>experimentid</td>
<td>varchar(46)</td>
<td>None</td>
</tr>
<tr>
<td>pe_workerid</td>
<td>int(11)</td>
<td>None</td>
</tr>
<tr>
<td>sentenceid</td>
<td>int(11)</td>
<td>None</td>
</tr>
<tr>
<td>version1</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>version2</td>
<td>text</td>
<td>None</td>
</tr>
</tbody>
</table>

MT_Text Table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>idmt_text</td>
<td>int(11)</td>
<td>Primary</td>
</tr>
<tr>
<td>experimentid</td>
<td>varchar(46)</td>
<td>None</td>
</tr>
<tr>
<td>sentenceid</td>
<td>varchar(46)</td>
<td>None</td>
</tr>
<tr>
<td>origmttext</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>mtext</td>
<td>text</td>
<td>None</td>
</tr>
<tr>
<td>task_id</td>
<td>tinyint(4)</td>
<td>None</td>
</tr>
</tbody>
</table>
### 4.2 Database Design

**PE_Errors Table**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_peerrors</td>
<td>int(11)</td>
<td>Primary</td>
</tr>
<tr>
<td>vp_workoid</td>
<td>int(11)</td>
<td>None</td>
</tr>
<tr>
<td>vp_errortype1</td>
<td>bit(1)</td>
<td>None</td>
</tr>
<tr>
<td>vp_errortype2</td>
<td>bit(1)</td>
<td>None</td>
</tr>
<tr>
<td>vp_errortype3</td>
<td>bit(1)</td>
<td>None</td>
</tr>
<tr>
<td>vp_errortype4</td>
<td>bit(1)</td>
<td>None</td>
</tr>
<tr>
<td>vp_errortype5</td>
<td>bit(1)</td>
<td>None</td>
</tr>
<tr>
<td>vp_errortype6</td>
<td>bit(1)</td>
<td>None</td>
</tr>
<tr>
<td>vp_errortype7</td>
<td>bit(1)</td>
<td>None</td>
</tr>
<tr>
<td>vp_errortype8</td>
<td>bit(1)</td>
<td>None</td>
</tr>
<tr>
<td>vp_errortype9</td>
<td>bit(1)</td>
<td>None</td>
</tr>
<tr>
<td>vp_comment</td>
<td>tex1</td>
<td>None</td>
</tr>
<tr>
<td>ldeo_text</td>
<td>int(11)</td>
<td>None</td>
</tr>
<tr>
<td>ps_workoid</td>
<td>int(11)</td>
<td>None</td>
</tr>
</tbody>
</table>

**Payment**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>payment_id</td>
<td>int(11)</td>
<td>Primary</td>
</tr>
<tr>
<td>chunk_id</td>
<td>int(11)</td>
<td>None</td>
</tr>
<tr>
<td>worker_id</td>
<td>int(11)</td>
<td>Indexed</td>
</tr>
<tr>
<td>status</td>
<td>varchar(15)</td>
<td>None</td>
</tr>
<tr>
<td>amount</td>
<td>float</td>
<td>None</td>
</tr>
</tbody>
</table>
Chapter 5
Implementation

5.1 Website

The web application has been developed using HTML, CSS, Javascript and Java EE technologies. The layout has been designed using Adobe Photoshop. The website has a green and blue theme that matches with the logo of both CA Technologies and UPC. The website content is displayed in five different languages namely: English, Catalan, Spanish, German and Chinese. We selected these languages for our initial prototype implementation. The content was translated by CA Technologies’ localization teams. The home page in English language is shown in Figure 5.1. The website can be accessed at http://crowd.pc.ac.upc.edu/

5.2 MVC architecture

The Model-View-Controller (MVC) architecture (or design pattern) has been around for a long time and is widely being used for the development of enterprise applications. Our system also follows the MVC architecture. As the name suggests, the system components are divided into three different parts, the model, the view and the controller. In Java EE, JavaBeans serve as the model, JSPs form the view and Servlets represent the controller.

Model

The model represents the backend data of the application and provides logic for accessing and manipulating this data. The model provides public methods that are utilized by the controller for accessing or updating the state of the model. Our application contains various manager classes in the model. The UnitManager class provides access to the PE_Unit, PEVE_Unit, RE_Unit and REVE_Unit tables. It is used to read or update the contents of these tables. The WorkerManager class is used to add new users to the worker table, access user data for operations and authentication, or updating user information. The TranslationManager class is used
to access and update the translations done by worker at different stages.

**View**
The view is responsible for the presentation layer of the application. It forwards the user input to the controller and also displays the state of the model. The JSP pages of our application comprise the view. The registration.jsp page has a form that takes details of the user and passes them to the controller, which in turn communicates with the model to store the user information in the database. The index.jsp page forwards the username and password to the controller for user authentication. There are several JSP pages that are used by the administrator to monitor the application and the tasks done by various users. For the worker-training phase we have several JSP pages that present questions to the user with multiple answers. In post editing, reviewing and verification phases of the system, several JSP pages are involved that show the appropriate interface to the user to work on.
5.3 Flow of the web application

Controller
The controller encapsulates the main logic of the application. It takes input from the view and performs various actions on it. The Registration Servlet in our application takes user information from the view and executes an insert query on the worker table. The Login Servlet authenticates the user, and redirects her to the profile page in case of successful login. There are several servlets used by the administrator to assign tasks to post editors, reviewers and verifiers by executing several complex queries on worker, text and unit tables. Various phases of the application are controlled by PE Servlet, PEVE Servlet, PE2 Servlet, RE Servlet, REVE Servlet and RE2 Servlet. Each of these servlets serves the appropriate content to the user to work on. The design of the MVC architecture of our application is shown in Figure 5.2.

![Model View Controller Architecture](image)

**Figure 5.2: Model View Controller Architecture**

5.3 Flow of the web application

There are three main flows of the web application. First is the flow of the user registration and initial assessment test. The users first register into the system using the registration page. Afterwards, they login to the system from the index page and are redirected to the interface where they can see an overview of the kind of tasks they have to perform in the system and some guidelines. After this step the users go through some pages and take a multiple-choice test related to translation. After completing the test, the users are assigned a category of A, B or C based on the performance in test. This flow is shown in Figure 5.3. Second flow is that of the administration modules. The administrator logs in to the
system from index page and after authentication she is redirected to the admin page. From this page the administrator can start a new translation task by uploading the text to be translated from an XML file. The text is automatically assigned to post editors. The administrator has four dashboards for monitoring the users in each of the four phases of the system. Dashboard 1 shows the list post editors and the status of their work. Similarly dashboard 2, 3 and 4 display the lists of post editors’ verifiers, reviewers and reviewers’ verifiers and the status of their work. From these interfaces the administrator also assigns jobs to each type of worker through various servlets. The admin flow is shown in Figure 5.4.

The third flow is that of the main application modules used by the workers participating in the system. After successful login, each worker is redirected to her profile page where she can see the pending jobs. The worker can click on a job to start working on it. Each worker is redirected to either post editing, post editor verification, reviewing or reviewer verification page based on the type of worker. Each of these pages calls a servlet that is responsible for displaying the right content and storing the work done by the workers in the database. This is shown in Figure 5.5.

5.4 Modules

5.4.1 Registration module

The registration module consists of a registration form in five different languages namely: English, Catalan, Spanish, German and Chinese. On submitting the form a new user is created. The values from the registration form including personal information like name, email, password as well as information related to the project like native language, English proficiency, translation experience, field of study etc, are handled by RegistrationServlet and stored in the database through RegistrationManager Java bean. Before inserting the new record, the database is first queried to check if a record with the specified username already exists. In that case an error message is displayed to the user.

5.4.2 Login and Logout modules

A user logs in to the system using email and password. The Login Servlet receives username and password as parameters from the index page. The database is queried to extract the record containing this username. In the database the passwords are not stored as plain text. They are encrypted with SHA1 encryption algorithm and the resulting hashcode is stored in the database. Therefore, we encrypt the password received from index page with SHA1 and then compare the hashcode with the one retrieved from the database. If the hashcodes match, we allow the user to login and start a new session. In case if the hashcodes do not match or there is no record
Figure 5.3: Web flow of initial assessment
Figure 5.4: Web flow of admin modules
Figure 5.5: Web flow of main application
with the specified username, the user is redirected to the index page with the error message that the username or password is incorrect. In the logout module, all session variables are deleted and the session is invalidated in the Logout Servlet and the user is redirected to the index page.

5.4.3 User dashboard

In the user dashboard a user can see the pending jobs, accept to work on a specific job and submit jobs after completion.

5.4.3.1 Profile module

The Profile Servlet displays the profile page to the user with the pending tasks. The servlet first checks the type of the user, which can be PE, PEVE, RE or REVE. Afterwards, based on the type of user it queries PEUnit, PEVEUnit, REUnit or REVEUnit table to retrieve the pending tasks of the user. With each task there is an accept button. On clicking the button the user is redirected to one of the six servlets responsible for post editing, post editing verification, post editing revision, review, review verification or review revision task.

5.4.3.2 Post Edit (PE) module

The PE Servlet displays the original text and the machine translated text to the post editor in two columns after reading from the database table MT_Text. The post editor can modify and improve the machine translation. One post editing task consists of roughly 50 statements. The post editor can save the work, sign out and continue working from the same position next time. The interface for post editing is shown in Figure 5.6. When the entire task is finished the post editor can submit it and it gets stored in the PE_Text table. This task goes to multiple verifiers who find mistakes in translation and provide comments. PE2 Servlet displays the verifiers comments to the post editor. After receiving the feedback the post editor improves the statements having mistakes and uploads a revised version of the task, which is again stored in PE_Text table in another column. This interface is shown in Figure 5.7.
Figure 5.6: Post editing interface
5 Implementation

Figure 5.7: Post editing interface with verifiers’ feedback

Figure 5.8: Verification interface
5.4 Modules

5.4.3.3 Post Editor Verification (PEVE) module

The PEVE module displays the work done by a post editor to multiple verifiers as shown in Figure 5.8. The interface consists of two columns. The text in source language is loaded in first column from MT_Text table and the post-edited text in target language is loaded in the second column from PE_Text table where worker_id is the id of the post editor. Under each pair of statements there are nine check boxes, each with an error type. If there is a mistake in the translation done by post editor, the verifiers have to select one or more of the error types and also provide a comment. On submitting the work the selected errors and comments are stored in PE_Error table.

5.4.3.4 Review (RE) module

In this module the reviewer who is the native speaker of the target language finds grammatical or structural mistakes in the revised tasks by post editors. RE Servlet loads the post-edited text from PE_Text table and displays it in two columns. The reviewer can modify the statements that have linguistic mistakes and submit the improved version, which is stored in RE_Text table. After submission, the verifiers evaluate the work. RE2 Servlet displays the verifiers’ feedback to the reviewer, who improves the work and submits a revised version, which is stored in RE_Text table in another column.

5.4.3.5 Reviewer Verification (REVE) module

The REVE module is very similar to PEVE module. The difference between the two is the error types. Each verifier finds errors in the text and submits them, which are stored in RE_Error table.

5.4.4 Administrator dashboard

The administration dashboard is used by the system administrator for controlling the experiment. The administrator has various dashboards where she can assign job users and monitor the status of different jobs. There are separate servlets for these functions.

5.4.4.1 PE Assign module

The PE Assign Servlet is used by the administrator to assign tasks to the post editors. This is done by creating a new record in PE_Unit table with the worker_id of the post editor. The job status is initially set to pending.
5.4.4.2 PEVE Assign module

This module is used to assign verifiers to a post editor after she has finished and submitted the job. The "Assign PEVE" button appears in front of those post editors only, who have submitted their job. The PEVE Assign Servlet queries the database to find the available verifiers and assigns them to the post editor.

5.4.4.3 PE feedback module

The administrator can see the list of verifiers working on the task of a specific post editor in one row in the admin dashboard. At the end of each row is 'Send' button. When the administrator sees that all verifiers have finished the job, she clicks on the button, which calls PE Error Check Servlet. In this Servlet the comments of all the verifiers are accumulated and sent to the post editor. It also updates the status of the post editor, and sends a notification that her work has been verified. When the post editor logs in to the system she can see the errors pointed out by the verifiers and their comments and can upload a revised version of the task.

5.4.4.4 RE Assign module

The RE Assign module is used to assign tasks to the reviewers when the first two phases have been successfully passed. RE Assign Servlet creates a new record in RE_Unit table with the worker_id of the reviewer. The job status is initially set to pending.

5.4.4.5 REVE Assign module

This module is used to assign verifiers to a reviewer after she has finished and submitted the job. The logic of this module is similar to PEVE Assign module.

5.4.4.6 RE feedback module

RE Error Check Servlet accumulates the comments of all the verifiers, sends them to the post editor. The logic of this module is similar to PE feedback module.
5.4.5 Email module

The Email module consists of an Email Servlet that is used to send confirmation or invitation emails to users. A confirmation email is sent when the user is successfully registered or the user successfully submits a job. Invitation emails are sent when there is a new task available for the user to be performed. There are setter methods to set email body, subject and receiver for each email based on the language of the user and purpose of email. Code snippets from Email Servlet are shown here.

```java
String msgText = "";
String to = "";
String from = "support.dama@ac.upc.edu";
String host = "relay.upc.es";
String subject = "";
MimeMessage msg = new MimeMessage(session);
    msg.setFrom(new InternetAddress(from));
    InternetAddress[] address = {new InternetAddress(to)};
    msg.setRecipients(Message.RecipientType.TO, address);
    msg.setSubject(subject);
    msg.setSentDate(new Date());
    msg.setText(msgText, "UTF-8");
Transport.send(msg);
```

5.4.6 XML Parser module

This module is used to parse an XML file containing the strings in source language and their corresponding machine-translated text strings. XML Parser Servlet reads the file and stores the string in MT_Text table in two separate columns. A sample XML tag is shown below:

```xml
<Tu MatchPercent='0'>
<Tuv Lang='EN-US'>Metering Gateway Installation and Configuration</Tuv>
<Tuv Lang='CA-01'>Instal·lació de la Passarel·la de mesura (MGT) I Configuracio</Tuv>
</Tu>
```

Acknowledgement

The user dashboard and its associated modules have been developed by my group member Andrea Gritti. They have been included here to give a complete picture of the entire system.
Chapter 6

Evaluation

We performed an experiment on the initial prototype system to test how well our algorithms work. The experiment results also helped us in selecting the right people for the right phase. We selected two languages for the initial experiment, namely: Catalan and Spanish. We spread the word about the experiment through flyers, online social networks like Facebook, Google+ and Twitter etc and also through friends and acquaintances.

6.1 Motivation for participation in experiment

To motivate the people to participate in the experiment we provided several benefits and incentives. First of all the participant was are helping a public university to try a new technology that leverages the potential of all the users of the Internet, so it was a philanthropic reason for some of them to contribute. For the translation students or the people who would consider translation as a career in future, we provided them with the advantage that they will be among the first group of people to be contacted when the commercial version of the crowdsourcing platform for translation is built. On this platform, participants would be compensated for the translations that they perform. Finally, to motivate the general public to participate in the experiment we provided an incentive of giving away an iPad to one of the participants through a lucky draw.

6.2 Training and assessment test

We provided the registered users with style guides and glossary to help them get familiar with the translation process. We performed a small test to assess their competency level. The test consisted to two parts. The first part consisted of a statement in English and multiple translations in the target language. The user had to select the correct translation. A screenshot of part 1 of the test is shown in Figure 6.1.
The second part of the test consisted of questions where a statement in source language and its translation in target language was given and the user had to find out errors in translation. Nine different types of errors were provided as checkboxes. A statement could have more than one error also. A screenshot of part 2 of the test is shown in Figure 6.2. In part 3 the users were given a set of statements in target language only and they had to find grammar, style, punctuation or typographical errors. This section was specifically meant for monolingual users. A screenshot of this part is shown in Figure 6.3.

6.3 Results and discussion

The test was performed in two languages: Spanish and Catalan. There were two different tests for each language, one for monolingual speakers and one for bilingual speakers (English+Spanish or English+Catalan). The monolingual test consisted of 21 questions and the bilingual test consisted of 26 questions. The results were calculated based on the following scheme.

Scoring scheme for part 1

In part 1 each question had only one correct answer. The user was given 1 point for
6.3 Results and discussion

selecting the correct answer and 0 for a wrong answer.

<table>
<thead>
<tr>
<th>Error de traducció</th>
<th>Omissió-Addició</th>
<th>Opcions de software</th>
<th>Gramàtica</th>
<th>Estil</th>
<th>Puntuació</th>
<th>Error ortogràfic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.2: Assessment test part 2

<table>
<thead>
<tr>
<th>Error de traducció</th>
<th>Omissió-Addició</th>
<th>Opcions de software</th>
<th>Gramàtica</th>
<th>Estil</th>
<th>Puntuació</th>
<th>Error ortogràfic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.3: Assessment test part 3
6 Evaluation

Scoring scheme for part 2
In part 2 each question had zero or more correct answers. The weight of each correct answer was based on the total number of correct answers of that question. E.g. if question 1 has four correct answers then the user gets 0.25 points for selecting each correct answer and loses 0.25 points for selecting each wrong answer. The overall score for any question can’t be less than 0.

Scoring scheme for part 3
The scoring scheme of part 3 was similar to that of part 2.

Figure 6.4: Individual scores of monolingual users

Figure 6.5: Individual scores of bilingual users
6.3 Results and discussion

Figure 6.6: Histogram of users' scores

Table 6.1: Detailed score of all users

<table>
<thead>
<tr>
<th>Test Part</th>
<th>Mean Score</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.39</td>
<td>1.60</td>
</tr>
<tr>
<td>2</td>
<td>3.81</td>
<td>1.36</td>
</tr>
<tr>
<td>3</td>
<td>3.88</td>
<td>1.49</td>
</tr>
<tr>
<td>all</td>
<td>13.09</td>
<td>3.07</td>
</tr>
</tbody>
</table>

The results were calculated based on the above scoring schemes and are shown in the graph in Figure 6.4 and Figure 6.5 for monolingual and bilingual users respectively. In Figure 6.4 we have 18 monolingual speakers. The x-axis shows the ids of users and the y-axis shows the percentage of score achieved in the test. Similarly, in Figure 6.5 we have 118 bilingual speakers and their score is shown. The histogram in Figure 6.6 shows the distribution of all users. The x-axis shows the score blocks and the y-axis shows the number of users in each score block. The histogram follows a normal distribution. The highest number of users falls in the middle block with a score of 50. The number of users gradually decreases from this value on both sides. Almost 78% of users have scores in the range of 0.40 and 0.65. There is also one outlier scoring 90. Table 6.1 shows the detailed score of all users in section 1, 2 and 3 of the test.

More interesting results can be seen when we analyze the score by separating the
users into different categories. During the registration process each user selects her translation experience. We provide the following three categories and a user can select one of them.

1. Professional - This category is for the people who have prior experience of translation as a profession in a company or as a freelancer.
2. Student - This category is for students of translation academies who study translation to adopt it as a profession in future.
3. Amateur - This category is for the general bilingual speakers who have not performed translations professionally but they could possibly do it because they know the source and target language.

<table>
<thead>
<tr>
<th>Test Part</th>
<th>Mean Score</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.00</td>
<td>1.32</td>
</tr>
<tr>
<td>2</td>
<td>4.30</td>
<td>1.16</td>
</tr>
<tr>
<td>3</td>
<td>4.32</td>
<td>1.50</td>
</tr>
<tr>
<td>all</td>
<td>14.62</td>
<td>3.13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Part</th>
<th>Mean Score</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.72</td>
<td>1.51</td>
</tr>
<tr>
<td>2</td>
<td>3.55</td>
<td>1.17</td>
</tr>
<tr>
<td>3</td>
<td>3.70</td>
<td>1.53</td>
</tr>
<tr>
<td>all</td>
<td>12.97</td>
<td>2.79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Part</th>
<th>Mean Score</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.98</td>
<td>1.65</td>
</tr>
<tr>
<td>2</td>
<td>3.68</td>
<td>1.46</td>
</tr>
<tr>
<td>3</td>
<td>3.74</td>
<td>1.46</td>
</tr>
<tr>
<td>all</td>
<td>12.41</td>
<td>3.5</td>
</tr>
</tbody>
</table>

It was expected that professional translators will perform and best and amateurs the worst. The results in Table 6.2, 6.3 and 6.4 show the expected trend. However, we can see that on average the difference between amateurs and professionals is not
6.3 Results and discussion

The Tables show the average score and the standard deviation. As can be seen that amateurs are more widely spread as compared to the professionals, while the students have the lowest spread. This trend can also be seen in the histogram in Figure 6.7. We have the percentage of score on x-axis and the number of users achieving that score on y-axis. As expected, majority of the professionals is on the right half of the histogram between the range of 0.60 and 0.70. Majority of the students is in the range of 0.50 and 0.70 and amateurs in the range of 0.45 and 0.70. The test was made quite hard and the highest score achieved was 0.70 (except one outlier). Taking this into account the performance of amateurs was very good. Almost 92% of them scored between 0.35 and 0.65, which roughly represents the range of 0.45 to 0.85 in a moderately hard translation task.
7 Conclusion and Future Work

7.1 Conclusion

In this work we have developed a prototype of a crowdsourcing framework for software localization to overcome the limitations currently faced by CA Technologies Inc. We have addressed some of the challenges of crowdsourcing systems that act as a barrier for the adoption of this technology at industrial level. The main challenge is quality, which we have addressed by proposing the Action-Verification Unit (AV-Unit) and quality-based rewarding system. We have developed algorithms for selection of workers for participation in different phases, and ranking of workers. The preliminary results show that on average the quality of work delivered by the crowd is of acceptable level. We can further improve the quality by employing our quality control mechanisms.

7.2 Future Work

We have designed a more extensive experiment to verify the effectiveness of the quality control mechanism. The results of the experiment will also enable us to fit the best combination of workers in each of the four phases, which gives the desired quality of output with maximum cost efficiency. We assign the category A, B or C to the users based on their performance in the initial assessment test explained in Section 5.3. After completing the categorization we assign the workers to the four phases of the translation process. After each phase we check the quality of the work by using automatic evaluation tools like BLEU [22] and METEOR [22]. The distribution of workers can be seen in Table 7.1.
Conclusion and Future Work

Table 7.1: Distribution of workers in each phase

<table>
<thead>
<tr>
<th>Worker Type</th>
<th>No. of workers</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post Editor</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>PE Verifier</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Reviewer</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>RE Verifier</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Workers of each category are given hypothetical ranks as shown in Table 7.2.

Table 7.2: Hypothetical ranks of each category

<table>
<thead>
<tr>
<th>PE PE Verifier</th>
<th>RE RE Verifier</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>X</td>
<td>100</td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
<td>70</td>
</tr>
<tr>
<td>C</td>
<td>Z</td>
<td>50</td>
</tr>
</tbody>
</table>

Based on the above ranks the PE Verifiers are divided into four groups. The sum of the ranks of verifiers range between 300 and 150 as shown in Table 7.3. After phase 1 gives 18 distinct results, one by each post editor. The quality of each output is evaluated using BLEU and METEOR. These results are assigned to a group of verifiers for verification.

Table 7.3: Groups of verifiers

<table>
<thead>
<tr>
<th>Group no.</th>
<th>Combination</th>
<th>Total rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AAA</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>BBB</td>
<td>210</td>
</tr>
<tr>
<td>3</td>
<td>CCC</td>
<td>150</td>
</tr>
<tr>
<td>4</td>
<td>ABC</td>
<td>220</td>
</tr>
</tbody>
</table>

Each group verifies the work of one category A, one category B and one category C post editor. They find translation errors and the feedback is sent to the respective post editors. The post editors upload the revised version of their work. The quality is evaluated again to see the improvement from first version and the impact of using different groups of verifiers. The revised versions of these 18 outputs are sent to one X, one Y and one Z category reviewer. The reviewers find structural and fluency mistakes in the next from the point of view of native speakers. In this way, after phase 3 we had 36 outputs. The quality of each of these outputs is evaluated to see the how the review phase impacts quality. It also reveals if there is a significant difference in quality of texts reviewed by X, Y and Z category workers.
7.2 Future Work

Each of these 36 outputs is then sent to a group of verifiers who find errors in the works of reviewers and send the feedback to the respective reviewer. The reviewers upload the revised version of their work in the light of the feedback. The quality of the revised work is again evaluated to see the impact of verification phase as well as the use of different groups for verification.

The design of this experiment is shown in Figure 7.1. After this experiment we will have an insight into the impact of each phase and the use of different combinations of workers in these phases, on the quality of translation.
We will assign the total cost of translation to each flow from phase 1 to phase 4 based on the costs in Table 3.5 and Table 3.6. The cost of each flow will vary depending on the combination of workers in that flow. We will sort all flows in descending order by cost. Afterwards, we can select the optimal flow that gives us the desired quality and lower cost, and implement it in the future release of our crowdsourcing system. We can also skip some of the next phases on runtime if the desired quality is achieved after phase 1, 2 or 3. This will also result in further cost saving.
Bibliography


