Título: CrowdSim: A Crowdsourcing Simulation System

Autor: Alex Borés Ricart

Data: Març 2013

Director: Josep Larriba i Pey

Departament del director: Arquitectura de Computadors (AC)

Centre: Facultat d'Informàtica de Barcelona (FIB)

Universitat: Universitat Politècnica de Catalunya (UPC) Barcelona Tech
Table of Contents

1. INTRODUCTION ........................................................................................................................................3
   1.1. Crowdsourcing ......................................................................................................................................5
   1.2. The simulation .......................................................................................................................................8
   1.3. Motivation ...........................................................................................................................................9
   1.4. Limitations of the current localization process ..................................................................................10
   1.5. Building a Crowdsourcing Platform for Software Localization .....................................................12
   1.6. Advantages of using crowdsourcing on this example .....................................................................14

2. REQUIREMENTS .........................................................................................................................................15
   2.1. Functional Requirements ..................................................................................................................15
   2.2. Non-Functional Requirements .........................................................................................................16

3. DESIGN .....................................................................................................................................................17
   3.1. Classes ...............................................................................................................................................17
       Global ....................................................................................................................................................17
       MyThread ............................................................................................................................................19
       Worker ...............................................................................................................................................21
       Translation ..........................................................................................................................................30
       Task .....................................................................................................................................................32
       Model ..................................................................................................................................................34
       SourceOfTranslations .........................................................................................................................37
       SourceOfWorkers .................................................................................................................................38
       MainForm ..............................................................................................................................................39
   3.2. Existing classes and libraries ............................................................................................................43
       QWaitCondition and QMutex .............................................................................................................43

4. AVAILABLE TECHNOLOGY ..................................................................................................................44
   4.1. Python ...............................................................................................................................................44
   4.2. Qt .......................................................................................................................................................45
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>PySide</td>
<td>46</td>
</tr>
<tr>
<td>4.4</td>
<td>SimPy</td>
<td>46</td>
</tr>
<tr>
<td>4.5</td>
<td>matplotlib</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTATION</td>
<td>48</td>
</tr>
<tr>
<td>5.1</td>
<td>Threading with Python</td>
<td>48</td>
</tr>
<tr>
<td>5.2</td>
<td>The use of SimPy</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Processes</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>Defining a process</td>
<td>54</td>
</tr>
<tr>
<td>5.3</td>
<td>The use of PySide</td>
<td>57</td>
</tr>
<tr>
<td>6</td>
<td>HOW IT WORKS</td>
<td>58</td>
</tr>
<tr>
<td>6.1</td>
<td>General tab</td>
<td>58</td>
</tr>
<tr>
<td>6.2</td>
<td>Job tab</td>
<td>60</td>
</tr>
<tr>
<td>6.3</td>
<td>Population tab</td>
<td>61</td>
</tr>
<tr>
<td>6.4</td>
<td>Output tab</td>
<td>62</td>
</tr>
<tr>
<td>6.5</td>
<td>Results tab</td>
<td>63</td>
</tr>
<tr>
<td>6.6</td>
<td>Translations tab</td>
<td>64</td>
</tr>
<tr>
<td>6.7</td>
<td>Chunks tab</td>
<td>65</td>
</tr>
<tr>
<td>6.8</td>
<td>Workers tab</td>
<td>66</td>
</tr>
<tr>
<td>6.9</td>
<td>SSD Graph tab</td>
<td>67</td>
</tr>
<tr>
<td>7</td>
<td>EXECUTION</td>
<td>68</td>
</tr>
<tr>
<td>8</td>
<td>PROJECT PLANNING</td>
<td>70</td>
</tr>
<tr>
<td>8.1</td>
<td>Methodology</td>
<td>70</td>
</tr>
<tr>
<td>8.2</td>
<td>Stages</td>
<td>71</td>
</tr>
<tr>
<td>9</td>
<td>ECONOMICAL ANALYSIS</td>
<td>72</td>
</tr>
<tr>
<td>10</td>
<td>CONCLUSIONS</td>
<td>73</td>
</tr>
<tr>
<td>11</td>
<td>ACKNOWLEDGEMENTS</td>
<td>75</td>
</tr>
<tr>
<td>12</td>
<td>REFERENCES</td>
<td>76</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

This thesis describes the design and implementation process for a crowd sourcing simulation system.

The crowd sourcing system that we are going to simulate performs text translations and it belongs to CA Technologies. They generate certain information over time that needs to be translated to different languages. Moreover, they have become aware that using a crowd sourcing solution may reduce costs and improve results.

CA Technologies develops and uses machine translation technologies and tools to support software localization activities. Like most large software vendors, CA Technologies continuously improves its processes to reduce the cost of translations and increase the number of languages it supports. The most expensive and time-consuming phase of the localization process is the human post-editing after machine translation, especially when it has to be outsourced to translation service providers.

CA Technologies is one of the largest software corporations in the world. With a 4.4 billion US$ in revenue for fiscal year 2010, 150 offices in 45 different countries, CA Technologies provides work to more than 13000 employees, all of them allocated onto a varied range of software products that include anti viruses, mainframe applications, distributed computing, virtualization and cloud environments.

As a result of its size and its presence in a large variety of markets around the world, CA Technologies must create its software in a localized way.

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, and it is currently investing proximately around 10M $ per year to satisfy CA Technologies quality standards in translation.

CA Technologies has a localization team placed in Cornellà de Llobregat, which is in charge of translating both software and user guides, and is responsible for the localization process behind the commercialization of CA Technologies products in Europe. This team is composed of 20 professional translators and is in charge of translating both user interfaces for software applications and manuals for its products, from English to 4 different European languages: Spanish, French, Italian and German.
The current work methodology implies the use of the most recent Machine Translation technologies as well as an extensive post-editing of the MT output in order to reach the CA Technologies quality standards.
1.1. Crowdsourcing

For many years, we have seen a huge increase in the use of sophisticated systems that have enabled world-wide collaboration through our home PCs and, more recently, through ubiquitous hand-held devices. The purposes are as numerous as they are varied: content sharing, whether through blogs or many well-known peer-to-peer (P2P) applications; collaborative computation, starting from the early SETI@home project (setiathome.berkeley.edu) that was one of the first large-scale grid computing instances, and other examples.

People are gaining awareness of the power of collaboration through the network. We have recently seen national revolutions, like in Egypt, where people organized themselves using digital platforms. The crowd is becoming aware of its power, and the next natural step is to enhance the tools and modalities for collaborative computing. Powerful devices, like smartphones and tablets, are able to carry out an impressive amount and array of computation. P2P computing has shown to be feasible and efficient. We have some examples, such as Skype, that show that the model is valid and can challenge serious cloud-based competitors, such as Google Voice.

Not only machines but also real people are “connected” to the network combining their computing and thinking capacity. Trends point to this model as gaining the position to complement cloud computing: connecting people and machines in a single network. Nowadays, millions of people are asynchronously analysing, synthesizing, providing opinion and labelling and transcribing data that can be automatically mined, indexed and even learned. Therefore, there is not much difference between this and classical computing: the “crowd” is working online, taking digital data as input and yielding digital data as output. The main difference is that human brain-guided computation is able to perform tasks that computers can hardly do, at overwhelming speeds. A few examples of this are tagging a picture or a video based on their content or answering questions in natural language.

The term crowdsourcing was used for the first time by Jeff Howe in 2006, referring to the increasing practice of outsourcing tasks to internet as an open call over a variety of users. Since then, crowdsourcing has evolved and a large crowd of people remotely connected through the Internet are making use of it. For instance, recent work studies
different typologies and uses of crowdsourcing and proposes a possible taxonomy. They categorised crowdsourcing depending on different methodologies and processes.

As this idea increases in terms of popularity, several general purpose crowdsourcing platforms have appeared in the last few years. A good example is Amazon Mechanical Turk, a crowdsourcing marketplace that allows companies or individuals to use the human intelligence to perform simple tasks that are really complicated for computers. The requesters post tasks known as Human Intelligence Tasks (HITs) that can be viewed by workers. Other examples such as CrowdFlower (crowdflower.com) or ClickWorker (clickworker.com) extend Mechanical Turk capabilities by offering a wide variety of crowdsourcing services. They improve quality by using gold standard units, redundant reviewers of each data unit, etc. Their work-flow management system divides complex tasks into smaller units and distributes them among the crowd based on the profile of individuals.

Quality control is a key point in crowdsourcing and it changes depending on the nature of the task. On the one hand, Lease and Yilmaz\(^1\) show how the results obtained from the crowd are more inaccurate compared to laboratory participants. On the other hand, Yan et al.\(^2\) present CrowdSearch, a system that performs image search on mobile phones using the crowd. They show that workers are able to achieve over 95% precision. Other lines of research study the effect that different rewarding systems have on quality. For instance, Harris\(^3\) shows that financial incentives actually encourage quality. Along this line, some platforms like Kaggle (kaggle.com) guarantee quality by two combined strategies: an open competition (to derive the best predictive model for a given set of data) with an economic incentive for the winner.

Crowdsourcing markets are traditionally used for simple and independent tasks. As an example, labelling an image or finding relevance between search results. In \(^4\), the authors present a framework that enables solving complex and interdependent tasks using crowdsourcing markets. The authors follow an approach similar to MapReduce for breaking down a complex problem into a sequence of simpler subtasks. The subtasks are solved in parallel by the crowd and the results are combined to form the final solution.

In the use case presented on this thesis, we focus on the use of crowdsourcing for translation and, specifically, on software localization. Several research works are based on the use of Amazon Mechanical Turk for translation\(^5,6,7,8\). Zaidan et al. propose some factors to select a good translation among a set of different versions. These factors
include the workers country of residence, native language, etc. and each factor has its own weight. Experimental results show that some translations turn out to be very close in quality to the ones made by professional translators. Two other studies\textsuperscript{9,10} investigate the use of crowdsourcing to evaluate the output of machine translated natural language. Gao and Vogel\textsuperscript{11} present a case study of word alignment tasks performed by crowd on Mechanical Turk and Matteo et al.\textsuperscript{12} use crowdsourcing to create corpora to feed and enrich Statistical machine translation. Other studies focus on using crowdsourcing for post-editing tasks. Bernstein et al.\textsuperscript{13} propose the Find-Fix-Verify pattern to perform tasks like text shortening and proofreading. They split the tasks into a series of stages that utilise independent agreement and voting to produce reliable results.

However, these systems might not be suitable for implementing an industrial software localization process for two main reasons. Firstly, these systems are based on the resolution of very small tasks, mostly at sentence level, and their entire quality assurance methodology is based on this reduced amount of information. Secondly, many of them use automatic evaluation methods like BLEU and METEOR to evaluate the quality of translations. However, these methods rely on the existence of pre-computed golden translations. Golden translations are rarely available for unpopular languages and cannot be used for next texts. This requires human intervention to decide which translations are to be accepted and which are not, adding an additional managerial layer to the process.
1.2. The simulation

In our specific scenario, we will have to simulate a crowd that receives computer automated translations and improves them in order to get an appropriate result. In order to do this, we will divide translations into sets of sentences that can be easily managed. We will refer to each of these sets of sentences as a chunk. By means of this, workers will receive a chunk and improve its content by correcting the mistakes made by CA’s automated system.

Once this first objective is achieved, we will need a certain number of reviewers that will check whether the work done is correct or not. These workers will get a reviewed chunk and look for mistakes again. If they find any, our chunk will go back to its initial reviewer so she can decide if she wants to apply the suggested changes or not.

Ideally, after executing all this process, we will get a corrected translation and workers will be rewarded accordingly to the quality of their activity.

One of the main criticisms the use of crowd sourcing receives is at the same time one of its main advantages: everyone can cooperate. This particular feature requires a certain filtering process in order to decide whether a person is suitable or not for collaboration. A possible approach to this problem consists on testing people’s ability before allowing them to be members of the crowd.

In our concrete scenario, language examinations are previously made in order to evaluate the knowledge of our candidates. This way it is possible to construct an initial approximation of our worker’s real ability. This estimation will be essential when assigning tasks in our system (we will clarify this more in-depth later) and it will be periodically updated with the quality of the work made by our workers.

Therefore, measuring quality of work will be one of our main discussions. We will have to decide a proper way to do it in order to get an accurate approximation of our workers' real ability. By doing so, we will be able to assign tasks in a way that takes this information into account (and for example, we will assign fewer reviewers to our best translators).
1.3. Motivation

Our main motivation is analysing how the use of a crowd system can become a good solution to effectively carry out translations. The importance of this simulation resides on the fact that the obtained results will determine the actual implementation of the real application.

We wish to get an extensive view of the advantages and disadvantages of using a crowd system for acquiring quality translations. Of course, time will be one of our main concerns because our company has strict deadline requirements that need to be met. At the same time, the payment of workers’ salary will also have to be taken into account.

Up to our knowledge, there are no other applications that offer these same functionalities and we are very interested in discovering how useful it can be. As we said before, the use of crowd systems is becoming more and more popular and attractive for a lot of corporations because it has proven to be effective especially in terms of economic cost. For that reason, we believe that the implementation of a simulation system that allows an easy adjustment to different needs is a good idea.

Once the application is finished, we will be able to adapt it for other use cases and we believe it will suit a wide range of potential scenarios.
1.4. Limitations of the current localization process

Before describing the platform, we summarize the main motivations that lead to the proposal of a crowd-based system. The main limitations in the current localization process are:

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

- **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.

- **High cost of extending our market 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 economically expensive and thus unfeasible 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 very high.

- **High cost of software localization for languages we currently translate**: companies invest several million dollars in localization per year both in internal and outsourced localization.

In consequence, some objectives and scopes have been defined accordingly with CA Technologies in order to address and solve the over mentioned problems:

- **Reducing the cost of translation**: Part of the translation is made by CA Technologies internal team and another important part is outsourced to language vendors: approximately, 23% of this budget is spent in linguists and another 23% in translation services outsourcing. Outsourcing represents an important expense for CA technologies which should be reduced or even avoided.
• **Reducing the delay to market of the company’s products:** CA technologies wants to achieve simultaneous delivery for English and not English products and the current outsourcing partner is not able to provide this.

• **Obtaining a translation output acceptable under quality standards:** CA Technologies wants its quality standards to be reached and therefore quality management is a central point in the development of the project.

• **Increasing the capacity to translate to other languages:** CA Technologies is currently translating software from English to around 10 different languages. However, being able to translate to a larger number of languages could open the possibilities for the company to explore new markets.
1.5. Building a Crowdsourcing Platform for Software Localization

The objective of the translation platform is to overcome the above-mentioned issues. When localizing a user guide, the source texts go through the machine translation engine and a first automatic translation is produced. Usually, the original and the machine-translated texts are sent to human translators who post-edit the text written in the target language. After this, the text is ready to be published. When the amount of work exceeds the capacity of the translators, the localization has to be outsourced. With this proposal, the main goal is to crowdsource work instead of outsourcing it. This trustworthy crowdsourcing platform consists of:

- A model to divide a task into subtasks to be distributed among the crowd.
- A ranking function to evaluate the quality of the translations generated by a user.
- A model to organize tasks for its parallel execution.
- A quality-aware rewarding system to remunerate workers based on the quality of their work and other aspects that may range from their training to their location and native language.
- A quality-aware task sequence organization system that guarantees a minimum level of quality independently of the quality of translators.

We divide a task into smaller subtasks that can easily be solved by an individual worker. For example, a user guide containing roughly 100,000 words is divided into subtasks of 2,000 words each. This is different from other existing approaches that divide tasks into sentence level or paragraph level subtasks thus losing the context of the text. Our selected subtask size is both convenient for a worker to solve in reasonable time and also preserves the context. Workers work on different jobs like post-editing, fixing errors, verifying, etc. A quality control system is being developed, which will comprise one or two AV-Units, depending on the particular requirements of the translation, to provide quality in the final translation. An important aspect of this system is that, since the quality of their work directly affects the reward obtained after finishing a task, this motivates workers to focus on quality. Also, the quality control algorithm adapts to the availability of workers. If high ranked workers are available, a small number of them are assigned to a task. However, if only low ranked workers are available, this number is increased.
The platform will manage workers automatically. The software localization process will be carried out by professionals and non-professionals from around the world, while the quality of translations is being maintained by the system itself.
1.6. Advantages of using crowdsourcing on this example

Besides the obvious benefits of crowdsourcing, namely scalability, elasticity, etc., the use of this type of crowdsourced platform will reduce the cost of the software localization process significantly. It solves the problem of finding translators for less popular languages because it allows the participation of any remote translator around the world, and it improves quality. Specifically:

- **It leverages the capacity provided by the crowd to interact one-to-one with translators, increasing quality and agility.** Third-party systems are blackboxes, even if they are based on crowdsourcing strategies. When we work with localization service providers, quality issues usually arise. Many times this is due to the fact that translators are not familiar with CA’s products or even with the IT domain. When this happens, CA has to send back translations to the vendor and this is very time consuming. With this platform, they are able to make the process much more agile, since workers are exclusively trained through CA’s products and documents and the company can resubmit work directly to them, if necessary.

- **We eliminate middleman costs.** Existing translation service providers get a margin for their services.

- **We have full control of all the real costs, specifically of the per-word rate paid to translators.** Due to the socioeconomic situation of the countries speaking a certain target language, it is a common practice that the per-word rate varies from language to language. For example, the per-word cost for a translation into German is higher than the per-word rate for a translation into Russian.

- **We gain control in terms of confidentiality.** Since we are sending our information to the network, security is one of the issues of crowdsourcing. By using the proposed platform, it is possible to establish our own security measures. For instance, we decide how to partition the documents and who we send information to. With third-party services, we rely on security measures that are not under our control, but information is still distributed around the world.

By using this system costs will be reduced, CA Technologies will gain immediate capacity to translate to any language in the world and time-to-market of CA Technologies products will be significantly reduced.
2. REQUIREMENTS

2.1. Functional Requirements

- To develop a crowd systems simulator based on the action verification pattern. We will focus on a crowd system about text translation.
- To simulate the work arrival.
  - To adjust the input using:
    - Statistical distributions and their parameters.
    - Text files provided by the user.
  - To simulate the work pre-processing. For example, by dividing a task into subtasks that will be distributed between workers.
- To simulate the registration and desertion of workers.
- To simulate the execution of tasks.
  - To provide parameters to adjust the ability of workers. For example, the grammatical error detection probability.
  - To quantify the quality of a task. For example, by using the percentage of grammatical errors detected over the total number of sentences.
  - To classify workers according to their performance. For example, by maintaining an updated ranking with the performance of workers.
- To display graphic results of tasks during execution (time, cost, unfinished...).
- To evaluate the quality of the simulated product. In other words, we must determine whether our crowd system will be the optimal solution or not.
2.2. Non-Functional Requirements

- **Adaptability to other scenarios**: The application must be implemented in such a way that can be easily adapted to fit other possible use cases.
- **Documentation**: All software projects should be properly documented.
- **Platform compatibility**: The simulator must be compatible with Windows and Linux operating systems.
- **Usability**: The application has to be easy to use and to understand. Its design must be simple and the results must be properly displayed.
3. DESIGN

3.1. Classes

Global
This class includes all of the global variables definitions. Their initialisations can be found at the beginning of the MyThread class (globals function).

- *waitCondition* and *mutex*: These two variables are used to synchronize the execution of the main application and the thread. Basically, they operate when the Run / Pause button is pressed and they block the thread execution until the button is pressed once again.
- *refreshGUI (bool)*: We use this variable to indicate whether the GUI needs to be updated or not.
- *run (bool)*: This variable indicates whether the simulation is running or not.
- *ttc (int)*: Time to Completion (ttc) is the amount of time required to complete a task.
- *total_tasks (int)*: Its name is self-explanatory. It keeps the total number of tasks.
- *completed_trans (int)*: Again, a self-explanatory name. It keeps the total number of completed translations.
- *failed_trans (int)*: This is the total number of failed translations.
- *worker_id (int)*: This variable keeps the total number of workers.
- *translation_id (int)*: This variable keeps the total number of translations.
- *task_id (int)*: This variable keeps the total number of tasks.
- *translations (list)*: This is a list that contains all translations in our simulation.
- *workers (list)*: This list contains all of the workers in the simulation.
- *Tasks (list)*: This list contains all chunks in our system.
- *rsd (list of int)*: This list contains the different values for the standard deviation between ranking and ability obtained during the execution.
- *total_sentences (int)*: The total number of sentences in all the translations that have arrived to our system.
- *correct_sentences (int)*: The total number of accepted sentences.
- *min_ranking (int)*: The minimum ranking that we accept in order to be able to work in our system.
- **max_rounds (int):** The maximum number of rounds that have been needed to complete a chunk.
**MyThread**

This class includes the main functionalities. We decided to use Python’s threading capabilities in order to allow stepped simulations.

This class contains more classes: Worker, Translation, Task, Model, SourceOfTranslations and SourceOfWorkers.

On the one hand, this is the run function, which initialises the global variables, instantiates and runs a new Model, and destroys the thread when finished.

```python
def run(self):
    self.globals()
    self.myModel = self.Model()
    self.myModel.run()
    self.exit()
```

On the other hand, this is the globals functions, which does the initialisation for all the global variables we have described earlier on this section.

```python
def globals(self):
    Global.run = True
    Global.refreshGUI = False
    Global.errors = 0
    Global.corrected_errors = 0
    Global.ttc = 0
    Global.total_tasks = 0
    Global.completed_trans = 0
    Global.failed_trans = 0
    Global.worker_id = 0
    Global.translation_id = 0
    Global.task_id = 0
    Global.translations = []
    Global.workers = []
    Global.tasks = []
    Global.rsd = []
    Global.total_sentences = 0
    Global.correct_sentences = 0
    Global.min_ranking = myapp.ui.minRankingSpinBox.value()/100.0
    Global.max_rounds = 0
```
Worker

This class includes the definition for a worker in our crowd. It has a set of methods that allow a worker to perform post-edition and revision of a given task.

These are its main properties:

- Task_time_of_arrival (list of int):
- Task_added_errors: A dictionary that contains, for each task, the number of added errors.
- Added_errors (int):
- State (int):
- Reliability (int):
- Misses (int):
- Id (int): It keeps the worker’s id.
- Task_queue: This queue contains all the assigned tasks.
- Popname:
- Mistake_probability (float): It goes from 0 to 100 and it keeps the probability of adding errors to a sentence.
- Failed_tasks:
- Detection_ability (int): It goes from 0 to 100 and it contains the worker’s real detection ability.
- Commitment:
- Task: It is the current assigned task.
- Task_status:
- Completed_tasks:
- Task_marked_indexes: A dictionary that contains the number of marked indexes for each task.
- Hits:
- Task_errors_indexes:
- Task_corrected_errors: This dictionary contains, for each task, the number of corrections the worker has made.
- Task_time_required:
- Time_to_task_check: It is the amount of time that a worker spends between two consecutive logins.
- Average_speed:
- Group:
- Rv (int): This is the seed used for generating all random values for the worker.
- Ranking:
- Task_round:

Let's take a look at the main methods.

First of all, let's talk about the work function. This method holds the actual simulation process. We can see the yield statements that are used to block its execution for the time specified in its third parameter. This function consists on a loop that is continuously executed. On the one hand, the first condition is passed when a worker has no assigned tasks. In that case, we need to choose task. On the other hand, the else is met when the worker already has an assignment so the simulation needs to be performed.

```python
def work(self):
    """ Represents the work process. """
    while True:
        if self.task == 0:
            self.choose_next_task()
        if self.task == 0:
            yield hold, self, self.time_to_task_check
        elif self.task.round == 0:
            self.task_assignment()
        else:
            self.task_time_of_arrival[self.task.name] = self.sim.now()
            yield hold, self, self.task_time_required[self.task.name]
        if self.task != 0:
            self.perform_simulation()
        yield hold, self, self.time_to_task_check
```

After revising the work method, let's take a look at choose_next_task. This one consists on the selection of the following task. First of all, we check if there are queued tasks. These tasks belong to the worker, but they had to be queued because another user's intervention was needed. If this queue is empty, then we need to check whether there are available tasks in the global queue or not. If the global queue is not empty and the worker's ranking is not below the minimum, a task will be assigned. If ranking is below minimum, the worker is suspended.

```python
def choose_next_task(self):
    if self.task_queue != []:
        self.task = self.dequeue_task()
    if self.task == 0 and Global.tasks != []:
        if self.rv.randint(1, 1000) == 1:
```
self.state = "INACTIVE"
if self.ranking >= Global.min_ranking:
    self.task = self.choose_any_task()
else:
    self.state = "SUSPENDED"
return

And now let's take a look at the task_assignment method, which basically assigns a task for revision or translation depending on its status.

def task_assignment(self):
    if (self.task.status == MyThread.Task.available_for_revision or
        self.task.status == MyThread.Task.under_revision):
        self.revise()
    elif self.task.status == MyThread.Task.available_for_translation:
        self.translate()

The perform_simulation method performs the whole simulation. As we can see, we have implemented a special method for our first round. This is because the first round is different to the others. During this initial round, changes to ranking are calculated. Once again, we see that a worker can perform his job if and only if his ranking is above the minimum allowed. This minimum is specified on the general tab of the GUI.

def perform_simulation(self):
    if self.task.translation.status != MyThread.Translation.failed:
        if self.task.round == 0: self.first_round()
        elif self.task.status == MyThread.Task.available_for_translation:
            if self.ranking >= Global.min_ranking:
                self.simulate_post_edit_round()
                self.enqueue_task()
            else:
                self.remove_all_reviewers()
                self.worker_leaves_task()
        elif (self.task.status == MyThread.Task.available_for_revision or
              self.task.status == MyThread.Task.under_revision):
            self.simulate_revision_round()
            if (len(self.task_marked_indexes[self.task.name]) == 0 and
                len(self.task_errors_indexes[self.task.name]) == 0):
                self.remove_reviewer()
            else: self.enqueue_task()
    else: self.failed_tasks += 1
    self.task = 0
We use the `remove_all_reviewers` method when the post-editor’s ranking goes below the minimum threshold in order to tell the reviewers to leave the task.

```python
def remove_all_reviewers(self):
    for r in self.task.reviewers:
        if r.task_queue.count(self.task) == 1:
            r.task_queue.remove(self.task)
        if r.task == self.task:
            r.task = 0
    self.task.reviewers = []
    self.task.round = 0
```

And then we have the `first_round` method which is responsible for performing the first round simulation. The conditional statement is used to distinguish the case when the worker completes the assignment from the case when the worker leaves it unfinished.

```python
def first_round(self):
    if (self.reliability > self.rv.random() and
        self.ranking >= Global.min_ranking):
        self.worker_finishes_task()
    else:
        self.worker_leaves_task()
```

The `remove_reviewer` method is called when a reviewer finishes his task. This means that he or she has accepted all sentences as they are.

```python
def remove_reviewer(self):
    self.completed_tasks += 1
    self.task.assigned_reviewers -= 1
    self.task.completed_reviews -= 1
    self.task.reviewers.remove(self)
```

The `enqueue_task` function is called when a round finishes but the worker (post-editor or reviewer) needs to wait until next round to continue her task. In the post-edition case, the worker must always wait for the reviewers to see her work. In the revision case, the worker will call this method if she is still not satisfied with the post-edition.
def enqueue_task(self):
    self.task_status[self.task.name] = self.task.status
    self.task_round[self.task.name] = self.task.round
    self.task_queue.append(self.task)

The unqueue_task method is utilised when a worker needs to choose a task from his task queue. Basically, this function compares the status the task had when the worker left it and the present task status. This way, we can know if a task is ready for the worker to take it. It is important to see that we sort all tasks by their deadline before making the choice. This is because tasks with earlier deadline must be given the highest priority.

def unqueue_task(self):
    self.task_queue.sort(key=lambda task: task.translation.deadline, reverse=False)
    task = next((t for t in self.task_queue
                  if ((self.task_status[t.name]==MyThread.Task.available_for_revision and t.status==MyThread.Task.available_for_translation) or
                      (self.task_status[t.name]==MyThread.Task.under_revision and t.status==MyThread.Task.available_for_revision) or
                      (self.task_status[t.name]==MyThread.Task.under_revision and t.round>self.task_round[t.name]) or
                      (self.task_status[t.name]==MyThread.Task.available_for_translation and t.status==MyThread.Task.under_revision)),0)

    if task != 0:
        self.task_queue.remove(task)
        if task.status == MyThread.Task.available_for_translation:
            task.round += 1
            if task.round > Global.max_rounds: Global.max_rounds += 1

    return task
Here we have the revise method which is responsible for assigning a revision task to a worker. It updates the corresponding variables for that task and for that worker as well as it checks whether the task needs more reviewers or not.

One important detail to take into account is the way we consider how many reviewers we need to assign. We keep a variable called revision_status for each task that keeps the probability that we keep mistakes uncorrected. We update this probability each time a new reviewer is assigned and we only stop assigning reviewers once this number goes below 5%. This way we make sure that our translations will be up to our expectations.

```python
def revise(self):
    
    """
    Assigns a revision task to a worker.
    Updates the corresponding variables for that task and worker.
    Checks whether task needs more revisions or not.
    """
    self.task.assigned_reviewers += 1
    self.task.reviewers.append(self)
    self.task.status = MyThread.Task.under_revision

    rs = self.task.revision_status*(1-self.ranking)
    self.task.revision_status = rs
    if self.task.revision_status<=0.05: Global.tasks.remove(self.task)

    ttr = self.task.total_sentences*10/self.average_speed
    self.task_time_of_arrival[self.task.name] = self.sim.now()
    self.task_time_required[self.task.name] = ttr
    self.task_marked_indexes[self.task.name] = []
    self.task_errors_indexes[self.task.name] = []
```

The translate method is quite similar. It assigns a post-edition task to a worker and performs the corresponding updates to variables and removes the chunk from the global queue so that no other worker will take it.

```python
def translate(self):
    """
    Assigns a translation task to a worker.
    Updates the corresponding variables for that task and worker.
    Removes task from queue so that no other worker will take it.
    """
```
```python
    ttr = self.task.total_sentences*10/self.average_speed
    Global.tasks.remove(self.task)
    self.task_time_of_arrival[self.task.name] = self.sim.now()
    self.task_time_required[self.task.name] = ttr
    self.task.post_editor = self
    self.task_marked_indexes[self.task.name] = []
    self.task.status = MyThread.Task.under_translation
```

This method (choose_any_task) selects a task from the global queue. It considers the fact that workers cannot revise their own work and workers cannot revise those chunks they have failed to revise in the past. Chunks with earlier deadline are given the highest priority.

```python
def choose_any_task(self):
    """
    Chooses next task.
    Worker can not revise its own work.
    Worker can not revise his failed task.
    """
    Global.tasks.sort(key=lambda task: task.translation.deadline, reverse=False)
    task = next((t for t in Global.tasks if t.post_editor!=self and
t.reviewers.count(self)==0 and
t.failed_workers.count(self)==0 and
not (t.status == MyThread.Task.available_for_translation and
t.post_editor!=0)), 0)
    return task
```

```python
def worker_finishes_task(self):
    """
    Checks whether task was finished before deadline or not.
    """
    self.simulate_first_round()
    self.prepare_next_round()
```

prepare_next_round just does what its name says. It removes the reviewer if he or she has completed the task, or queues the task in case it is needed.

```python
def prepare_next_round(self):
    if (self.task.post_editor!=self and
```
len(self.task_marked_indexes[self.task.name])==0 and
len(self.task_errors_indexes[self.task.name])==0):
    self.remove_reviewer()
else:
    self.enqueue_task()

failed_task is called whenever a translation deadline is not met.

def failed_task(self):
    """ Prints task failure and updates its state. """
    self.task.status = MyThread.Task.failed
    myapp.ui.listWidget.addItem(
        "(0): {1} not completed by {2}. \\
        "Translation deadline was not met.".format(
            fullt(int(self.sim.now())),
            self.task.name,self.name))
    elem = []
    elem.append(fullt(int(self.sim.now())))
    elem.append(self.task.translation.id)
    elem.append(self.task.name[9:])
    elem.append(self.id)
    elem.append("NOT COMPLETED")
    elem.append("Translation deadline was not met.")
    row = myapp.ui.tableOfResults.rowCount()
    myapp.ui.tableOfResults.insertRow(row)
    self.print_table_line(elem,row,0)
MyThread_Worker

- _init__(self, name, sm, args, n, gnp, avgcpu, group, comm, instate, ran, tttt, file, popname)
- work(self)
- choose_next_task(self)
- task_assignment(self)
- next_task(self)
- perform_simulation(self)
- remove_all_reviewers(self)
- first_round(self)
- remove_reviewer(self)
- enqueue_task(self)
- dequeue_task(self)
- review(self)
- translate(self)
- choose_any_task(self)
- worker_finishes_task(self)
- prepare_next_round(self)
- failed_task(self)
- worker_leaves_task(self)
- simulate_post_round_rounding(self)
- error_post_round_rounding(self)
- error_addon(self, l)
- simulate_revision_rounding(self)
- error_revision(self)
- check_renovation_status(self)
- simulate_first_rounding(self)
- error_simulation(self)
- ml_sentences(self)
- error_counting(self)
- update_ranking(self)
- update_task(self)
- corrected_errors_sum(self)
- print_results(self)
- print_table_line(self, file, l)
- print(self, raw)

- task_time_of_arrival
- task_added_errors
- added_errors
- state
- reliability
- misses
- id
- task_queue
- popname
- mistake_probability
- failed_tasks
- detection_ability
- commitment
- task
- task_status
- completed_tasks
- task_marked_indexes
- hits
- task_errors_indexes
- task_corrected_errors
- task_time_required
- time_to_task_check
- average_good
- group
- RV
- ranking
- task_round
Translation

This class represents a translation in our simulation.

In the next snippet we can appreciate the update_status function, which is called once a task is finished. It checks whether the translation is completed or not. If all tasks are completed, then it updates the corresponding counting variables and prints the result both as a text line and as a table row.

```python
def update_status(self):
    
    Checks whether all tasks have been completed or not. And updates status and prints results.
    
    if all([task.status == MyThread.Task.completed
            for task in self.tasks]):
        self.status = MyThread.Translation.completed
        Global.completed_trans += 1
        time = niceTime(int(self.sim.now()) - self.time_of_arrival)
        for t in self.tasks:
            incorrect = t.errors - t.errors_corrected
            correct = t.total_sentences - incorrect
            self.correct_sentences += correct
            self.total_sentences += t.total_sentences
        Global.correct_sentences += self.correct_sentences
        Global.total_sentences += self.total_sentences
        myapp.ui.listWidget.addItem("{0}: {1} COMPLETED. \
        "Time To Completion = {2}. \
        "Correct sentences = {3} ({4}%).".format(
            fullt(int(self.sim.now())),
            self.name,
            time,
            self.correct_sentences,
            100 * self.correct_sentences / self.total_sentences))
        elem = []
        elem.append(fullt(int(self.sim.now())))
        elem.append(self.id)
        elem.append("-")
        elem.append("-")
        elem.append("COMPLETED")
        elem.append("Time To Completion = {0}. \
        "Correct sentences = {1} ({2}%).".format(
```
time,
self.correct_sentences,
100*self.correct_sentences/self.total_sentences))
row = myapp.ui.tableOfResults.rowCount()
myapp.ui.tableOfResults.insertRow(row)
self.print_table_line(elem,row,0)
Task

A task represents the actual work that a worker can perform. It has its own size, deadline, status and assigned workers. It has two deadlines: its own and its parent translation deadline. A worker has to go through it and detect the errors it contains. For simulation purposes, we assume that we know the actual number of errors that a certain task has before they are detected.

class Task(Process):
    (available_for_translation, under_translation,
     available_for_revision, under_revision,
     completed, failed) = range(6)
    def __init__(self, name, sim, translation, taskSizeMu, taskSizeSigma, seed):
        Process.__init__(self, name=name, sim=sim)
        self.id = Global.task_id
        self.rv = Random(seed*self.id)
        self.translation = translation
        self.total_sentences = \n            max(1, int(self.rv.normalvariate(taskSizeMu, taskSizeSigma)))
        self.sentences = []
        self.initial_sentences = 0
        self.corrected_sentences = []
        self.time_of_arrival = self.sim.now()
        self.time_remaining = 0
        self.errors = 0
        self.errors_corrected = 0
        self.revision_status = 0
        self.status = MyThread.Task.available_for_translation
        self.assigned_reviewers = 0
        self.completed_reviews = 0
        self.post_editor = 0
        self.reviewers = []
        self.failed_workers = []
        self.round = 0
        Global.task_id += 1
        self.str_status = [
            "AVAILABLE FOR TRANSLATION",
            "UNDER TRANSLATION",
            "AVAILABLE FOR REVISION",
            "UNDER REVISION",
            "COMPLETED",
            "FAILED"]
MyThread.Task

__init__(self, name, aim, translation, taskSizeMu, taskSizeSigma, seed)

- translation
- post_editor
- status
- time_of_arrival
- sentences
- round
- reviewers
- revision_status
- assigned_reviewers
- id
- total_sentences
- errors
- errors_corrected
- initial_sentences
- failed_workers
- corrected_sentences
- str_status
- rv
- completed_reviews
- time_remaining
- available_for_translation
- under_translation
- available_for_revision
- under_revision
- completed
- failed
Model

This class represents the simulation itself. It is a subclass of SimPy’s SimulationStep. It initialises all sources (of translations and workers) and waits until the simulation end time is reached. Then, it shows the simulation results.

The method that holds the simulation is called run and we can see it in the following code snippet. It calculates the simulation time by converting the number of weeks and months to hours. It start the stepping functionalities and it calls the simulate method by using a callback function as the first parameter. This function will be called each time a yield statement is achieved. Once the simulation finishes, it calls a set of print methods which will update the GUI properly.

Then we have the callbackUserControl method. This function stops the simulation when the pause button is pressed. It will also update the GUI at a certain time interval. The unlocking is controlled by the main application.

Finally, the sources method is in charge of activating the source of translations and the source of workers processes. The workers and translations methods are responsible for creating the initial workers and translations.

def run(self):
    myapp.ui.listWidget.clear()
    myapp.ui.lineEdit.clear()
    weeks = myapp.ui.weeksSpinBox.value()*168
    months = myapp.ui.monthsSpinBox.value() * 730.484
    self.until = weeks + months
    self.steps = 0
    self.start()
    self.startStepping()
    self.simulate(callback=self.callbackUserControl, until=weeks+months)
    self.printStats()
    self.printRanking()
    self.printTranslations()
    self.printChunks()
    self.showRSDPlot()

def callbackUserControl(self):
    if not Global.run:
        Global.mutex.lock()
        Global.waitCondition.wait(Global.mutex)
Global.mutex.unlock()
elif self.steps <= self.now():
    Global.refreshGUI = True
    Global.mutex.lock()
    Global.waitCondition.wait(Global.mutex)
    Global.mutex.unlock()
    Global.refreshGUI = False
    self.steps += myapp.ui.simulationSpeedSpinBox.value()*24*7

def start(self):
    self.initialize()
    self.sources()
    self.workers()
    self.translations()

def sources(self):
    sourceSeed = myapp.ui.sourceSeedSpinBox.value()
    for n in range(myapp.ui.tableOfJobDefinitions.rowCount()):
        sot = MyThread.SourceOfTranslations(sim=self,seed=sourceSeed,
                                             row=n)
        self.activate(sot,sot.generate(),0.0)
    for n in range(myapp.ui.tableOfGroups.rowCount()):
        sow = MyThread.SourceOfWorkers(sim=self,seed=sourceSeed,row=n)
        self.activate(sow,sow.generate(),1.0)
SourceOfTranslations

This class is in charge of simulating the arrival of translations over time. It has some parameters that adjust the time interval between two consecutive translations, the number of tasks per translation, their size and their deadline.

SourceOfTranslations is a subclass of SimPy’s Process. This way, we use a yield statement to wait a certain amount of time until a new translation is generated. As you can see in the following code snippet, we use activate to start the process. The first parameter is the translation itself and the second parameter is the method that will hold the process.

def generate(self):
    rv = Random(self.seed)
    while True:
        yield hold, self, rv.expovariate(1.0/self.interval)
    translation = MyThread.Translation(sim=self.sim,
                                       minDeadline=self.minDeadline, maxDeadline=self.maxDeadline,
                                       minTasks=self.minTasks, maxTasks=self.maxTasks,
                                       seed=self.seed, taskSizeMu=self.taskSizeMu,
                                       taskSizeSigma=self.taskSizeSigma)
    self.sim.activate(translation, translation.generate())
**SourceOfWorkers**

This class is in charge of simulating the arrival of new workers to our system. It has parameters to adjust the main features of workers.

SourceOfWorkers is also a subclass of SimPy's Process. Here we use a yield statement to wait a certain amount of time until a new worker arrives. As you can see in the following code snippet, we use activate to start the process. The first parameter is the worker himself and the second parameter is the method that will hold the process. We have talked about this method earlier in this chapter.

```python
def generate(self):
    rv = Random(self.seed)
    while True:
        yield hold, self, rv.expovariate(1.0/self.interval)
        if rv.randint(0,1000)<10:
            worker = MyThread.Worker(
                "Worker {0:02d}".format(Global.worker_id),
                sim = self.sim, mu = self.mu, sigma = self.sigma,
                avgspm = self.avgspeedm, avgsp = self.avgspeeds,
                seed = self.seed, group = self.row, comm = self.comm,
                mistake = self.mistake, ran = self.ran,
                tttc = self.tttc, fiab = self.fiab, popname = self.name)
            self.sim.activate(worker, worker.work())
```

![Class diagram for SourceOfWorkers](image)
MainForm

As we can see in the following diagram, MainForm is a subclass of QWidget. It displays the GUI and it controls the execution of our simulation thread.

Here we use the QMutex and QWaitCondition classes to control the execution of the thread. We stop the simulation when the pause button is presses and we resume it once it is pressed again. We will explain the use of these two classes more in-depth later on this chapter.

class MainForm(QtGui.QWidget):
    """ Represents the simulator gui and its events. """
    def __init__(self):
        super(MainForm, self).__init__()
        self.first = True
        self.state = "stopped"
        self.timer = QtCore.QTimer()
        self.timer.timeout.connect(self.refreshGUI)
        self.myThread = MyThread()
        self.myThread.started.connect(self.myThread_started)
        self.myThread.finished.connect(self.myThread_finished)
        self.ui = Ui_MainForm()
        self.ui.setupUi(self)
        self.ui.runButton.clicked.connect(self.runButton_clicked)
        self.ui.saveButton.clicked.connect(self.saveButton_clicked)
        self.ui.addGroupButton.clicked.connect(self.addGroupButton_clicked)
        self.ui.removeGroupButton.clicked.connect(self.removeGroupButton_clicked)
        self.ui.addTransButton.clicked.connect(self.addTransButton_clicked)
        self.ui.removeTransButton.clicked.connect(self.removeTransButton_clicked)
        self.ui.tableOfGroups.itemChanged.connect(self.tableOfGroups_validation)
        self.ui.tableOfJobDefinitions.itemChanged.connect(self.tableOfJobDefinitions_validation)
        self.addInitialGroup()
        self.addInitialTrans()
        self.ui.quitButton.clicked.connect(self.quitButton_clicked)
        self.ui.lineEdit.textChanged.connect(self.filter)
        self.ui.saveButton.setEnabled(False)
        self.tableOfWorkers_setResizeMode()
        self.tableOfChunks_setResizeMode()
self.tableOfTranslations_setResizeMode()
self.tableOfGroups_setResizeMode()
self.tableOfJobDefinitions_setResizeMode()
self.tableOfResults_setResizeMode()

weeks = self.ui.weeksSpinBox.value() * 168
months = self.ui.monthsSpinBox.value() * 730.484
self.ui.progressBar.setMaximum(weeks + months)

pyplot.figure()

# Global.refreshGUI.connect(self.refreshGUI

def myThread_started(self):
    self.ui.listWidget.setUpdatesEnabled(False)

def myThread_finished(self):
    self.state = "stopped"
    self.ui.runButton.setText("Simulate")
    self.timer.stop()
    self.ui.listWidget.scrollToTop()
    self.ui.listWidget.setUpdatesEnabled(True)
    self.ui.saveButton.setEnabled(True)
    self.ui.progressBar.setValue(self.myThread.myModel.now())
    myapp.ui.graphLabel.setPixmap(QtGui.QPixmap("temp.png"))

def runButton_clicked(self):
    if self.state == "stopped":
        Global.mutex.lock()
        self.timer.start(1000)
        self.ui.listWidget.setUpdatesEnabled(False)
        self.ui.saveButton.setEnabled(False)
        self.state = "started"
        self.ui.runButton.setText("Pause")
        self.myThread.start()
        weeks = self.ui.weeksSpinBox.value() * 168
        months = self.ui.monthsSpinBox.value() * 730.484
        self.ui.progressBar.setMaximum(weeks + months)
        Global.run = True
        Global.mutex.unlock()
    elif self.state == "started":
        Global.mutex.lock()
        self.timer.stop()
        self.ui.listWidget.scrollToBottom()
        self.ui.listWidget.setUpdatesEnabled(True)
        self.ui.saveButton.setEnabled(True)
        self.state = "paused"
self.ui.runButton.setText("Resume")
Global.run = False
Global.mutex.unlock()
elif self.state == "paused":
    Global.mutex.lock()
    self.timer.start(1000)
    self.ui.listWidget.setUpdatesEnabled(False)
    self.ui.saveButton.setEnabled(False)
    self.state = "started"
    self.ui.runButton.setText("Pause")
    Global.run = True
    Global.waitCondition.wakeAll()
    Global.mutex.unlock()
3.2. Existing classes and libraries

QWaitCondition and QMutex

The QWaitCondition class provides a condition variable for synchronizing threads. QWaitCondition allows a thread to tell other threads that some sort of condition has been met. One or many threads can block waiting for a QWaitCondition to set a condition with wakeOne() or wakeAll(). Use wakeOne() to wake one randomly selected condition or wakeAll() to wake them all.

The QMutex class provides access serialization between threads.

The purpose of a QMutex is to protect an object, data structure or section of code so that only one thread can access it at a time (this is similar to the Java synchronized keyword). It is usually best to use a mutex with a QMutexLocker since this makes it easy to ensure that locking and unlocking are performed consistently.
4. AVAILABLE TECHNOLOGY

4.1. Python

Python is a remarkably powerful dynamic programming language that is used in a wide variety of application domains. Fans of Python use the phrase “batteries included” to describe the standard library, which covers everything from asynchronous processing to zip files. The language itself is a flexible powerhouse that can handle practically any problem domain.

Python lets you write the code you need, quickly. And, thanks to a highly optimized byte compiler and support libraries, Python code runs more than fast enough for most applications.

Python is available for all major operating systems: Windows, Linux/Unix, OS/2, Mac, Amiga, among others. There are even versions that run on .NET and the Java virtual machine. The same source code will run unchanged across all implementations.

The Python newsgroup is known as one of the friendliest around. The avid developer and user community maintains a wiki, hosts international and local conferences, runs development sprints, and contributes to online code repositories.

Python also comes with complete documentation, both integrated into the language and as separate web pages. Online tutorials target both the seasoned programmer and the newcomer. All are designed to make you productive quickly. The availability of first-rate books completes the learning package.

The Python implementation is under an open source license that makes it freely usable and distributable, even for commercial use. The Python license is administered by the Python Software Foundation.
4.2. Qt

Qt ("cute") is a cross-platform application framework that is widely used for developing application software with a graphical user interface (GUI) (in which cases Qt is classified as a widget toolkit), and also used for developing non-GUI programs such as command-line tools and consoles for servers.

Qt uses standard C++ but makes extensive use of a special code generator together with several macros to enrich the language. Qt can also be used in several other programming languages via language bindings. It runs on the major desktop platforms and some of the mobile platforms. It has extensive internationalization support.

Qt is available under a commercial license, GPL v3 and LGPL v2. All editions support many compilers, including the GCC C++ compiler and the Visual Studio suite.

Qt is developed by Digia, who owns the Qt technology and trademark, and the Qt Project under open governance, involving individual developers and firms working to advance Qt.
4.3. **PySide**

PySide is an open source software project providing Python bindings for the Qt framework. Qt is a cross-platform application and UI framework, allowing the developers to write applications once and deploy them across many operating systems without rewriting the source code, while Python is a modern, dynamic programming language with a vivid developer community.

Combining the power of Qt and Python, PySide provides the wealth of Qt framework for developers writing software in Python and presents a first-class rapid application development platform available on all major operating systems.

4.4. **SimPy**

This library provides the modeller with components of a simulation model including processes, for active components like customers, messages, and vehicles, and resources, for passive components that form limited capacity congestion points like servers, checkout counters, and tunnels. It also provides monitor variables to aid in gathering statistics. Random variates are provided by the standard Python random module.

Many users claim that SimPy is one of the cleanest, easiest to use discrete event simulation packages.
4.5. **matplotlib**

matplotlib is a plotting library for the Python programming language and its NumPy numerical mathematics extension. It provides and object-oriented API which allows plots to be embedded into applications using generic GUI toolkits, like wxPython, Qt or GTK. There is also a procedural “pylab” interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB.

matplotlib was written and maintained primarily by John Hunter, and is distributed under a BSD-style license.

The `pylab` interface makes matplotlib easy to learn for experienced MATLAB users, resulting in a viable alternative for many MATLAB users as a teaching tool for numerical mathematics and signal processing.

Some of the advantages of the combination of Python, NumPy and matplotlib over MATLAB include:

- Based on Python, a full-featured modern object-oriented programming language suitable for large-scale software development.
- Free, open source, no license servers.
- Native SVG support.
5. IMPLEMENTATION

In this section we will discuss the main technological thrills we have faced.

5.1. Threading with Python

One of the most interesting features that CrowdSim provides is the possibility of performing a simulation by steps. The parameter we use to use this functionality is called Speed, and it is placed on the main tab. This parameter represents the number of weeks that are executed on each simulation step. Obviously, if we provide a value that is higher than the whole simulation time, then the execution will execute using just one step.

In order to implement stepping, we decided to make use of Python’s threading libraries because they fulfilled all of our requirements. One of them was related to the refresh of the GUI during the simulation. We wanted to provide a screen showing the output for each simulation step in real time (and not just at the end of the execution). As SimPy does not call the graphics library during its execution, we had to create a thread for the simulation. This way, the screen is refreshed with the last simulation results without modifying SimPy’s behaviour.
5.2. The use of SimPy

SimPy has been a fundamental tool for our project. This library includes all those functionalities required to implement a wide variety of simulations. Its documentation has proven to be very helpful, especially for the numerous examples it contains.

The two main classes we have utilised are called *SimulationStep* and *Process*. The most important statement is *yield hold* which makes a process to wait for a certain amount of time.

To use the SimPy simulation system you must import its Simulation module (or one of the alternatives):

```python
from SimPy.Simulation import *
```

All discrete-event simulation programs automatically maintain the current simulation time in a software clock. This cannot be changed by the user directly. In SimPy the current clock value is returned by the *now()* function.

At the start of the simulation the software clock is set to 0.0. While the simulation program runs, simulation time steps forward from one event to the next. An event occurs whenever the state of the simulated system changes. For example, an event might be the arrival or departure of a car from the gas station.

The following statement initializes global simulation variables and sets the software clock to zero. It must appear in the script before any SimPy process objects are activated.

```python
initialize()
```

This is followed by SimPy statements creating and activating process objects. Activation of process objects adds events to the simulation schedule. Execution of the simulation itself starts with the following statement:

```python
simulate(until=endtime)
```

The simulation starts, and SimPy seeks and executes the first scheduled event. Having executed that event, the simulation seeks and executes the next event, and so on.

Typically a simulation is terminated when endtime is reached but it can be stopped at any time by the command:
stopSimulation();

now() will then equal the time when this was called. The simulation will also stop if
there are no more events to execute (so now() equals the time the last scheduled event
occurred)

After the simulation has stopped, further statements can be executed. now() will retain
the time of stopping and data held in Monitors will be available for display or further
analysis.

The following fragment shows only the main block in a simulation program. Here Message
is a Process class and m is defined as an object of that class, that is, a
particular message. Activating m has the effect of scheduling at least one event by
starting the PEM of m (here called go). The simulate(until=1000.0) statement starts the
simulation itself, which immediately jumps to the first scheduled event. It will continue
until it runs out of events to execute or the simulation time reaches 1000.0.
In addition to SimPy.Simulation, SimPy provides four alternative simulation libraries which have the basic SimPy.Simulation capabilities, plus additional facilities:

- **SimPy.SimulationTrace** for program tracing:
  
  With from SimPy.SimulationTrace import *, any SimPy program automatically generates detailed event-by-event tracing output. This makes the library ideal for program development/testing and for teaching SimPy.

- **SimPy.SimulationRT** for real time synchronization:
  
  from SimPy.SimulationRT import * facilitates synchronizing simulation time and real (wall-clock) time. This capability can be used to implement, e.g., interactive game applications or to demonstrate a model's execution in real time.

- **SimPy.SimulationStep** for event-stepping through a simulation:
  
  The import from SimPy.SimulationStep import * provides an API for stepping through a simulation event by event. This can assist with debugging models, interacting with them on an event-by-event basis, getting event-by-event output from a model (e.g. for plotting purposes), etc.

- **SimPy.SimulationGUIDebug** for event-stepping through a simulation with a GUI:
  
  from SimPy.SimulationGUIDebug import * provides an API for stepping through a simulation event-by-event, with a GUI for user control. The event list, Process and Resource objects are shown in windows. This is useful for debugging models and for teaching discrete event simulation with SimPy.
Processes
The active objects for discrete-event simulation in SimPy are process objects – instances of some class that inherits from SimPy's Process class.

For example, if we are simulating a computing network we might model each message as an object of the class Message. When message objects arrive at the computing network they make transitions between nodes, wait for service at each one, are served for some time, and eventually leave the system. The Message class specifies all the actions of each message in its Process Execution Method (PEM). Individual message objects are created as the simulation runs, and their evolutions are directed by the Message class’s PEM.

Defining a process
Each Process class inherits from SimPy's Process class. For example the header of the definition of a new Message Process class would be:

```python
class Message(Process):
```

At least one Process Execution Method (PEM) must be defined in each Process class. A PEM may have arguments in addition to the required `self` argument that all methods must have. Naturally, other methods and, in particular, an `__init__` method, may be defined.

More than one can be defined but only one can be executed by any process object.

A Process Execution Method (PEM) defines the actions that are performed by its process objects. Each PEM must contain at least one of the yield statements, described later. This makes it a Python generator function so that it has resumable execution – it can be restarted again after the yield statement without losing its current state. A PEM may have any name of your choice. For example it may be called `execute()` or `run()`.

“The yield statements are simulation commands which affect an ongoing life-cycle of Process objects. These statements control the execution and synchronization of multiple processes. They can delay a process, put it to sleep, request a shared resource or provide a resource. They can add new events on the simulation event schedule, cancel existing ones, or cause processes to wait for a state change.”
For example, here is the Process Execution Method, go(self), for the Message class. Upon activation it prints out the current time, the message object’s identification number and the word “Starting”. After a simulated delay of 100.0 time units (in the yield hold statement) it announces that this message object has “Arrived”:

```python
def go(self):
    print now(), self.i, 'Starting'
    yield hold,self,100.0
    print now(), self.i, 'Arrived'
```

A process object’s PEM starts execution when the object is activated, provided the simulate(until= ...) statement has been executed.

`__init__(self, ...)`, where ... indicates method arguments. This method initializes the process object, setting values for some or all of its attributes. As for any sub-class in Python, the first line of this method must call the Process class’s `__init__( )` method in the form:

```python
Process.__init__(self)
```

You can then use additional commands to initialize attributes of the Process class’s objects. You can also override the standard name attribute of the object.

The `__init__( )` method is always called whenever you create a new process object. If you do not wish to provide for any attributes other than a name, the `__init__` method may be dispensed with. An example of an `__init__( )` method is shown in the example below.
5.3. The use of PySide

With PySide desktop applications, you must always start your file by importing PySide.QtCore and PySide.QtGui classes. These classes have the main functions for building PySide applications. For instance, PySide.QtGui contains functions for dealing with widgets while PySide.QtCore contains methods for handling signals and slots, and controlling the application.

After the imports, you create a QApplication which is the main Qt application. As Qt can receive arguments from command line, you must pass any arguments to the QApplication object. Usually, you do not need to pass any arguments so you can leave it as it is.

After the creation of the application object, we have created a QLabel object. A QLabel is a widget that can present text (simple or rich, like html), and images. Note that after the creation of the label, we are calling the method show which will present the label to the user.

Finally, we call app.exec_() which will enter the Qt main loop and start to execute the Qt code. In reality, it is only here that the label will be shown, but this can be ignored for now.
6. HOW IT WORKS

In this section, we will describe how to actually use the simulator by specifying the meaning of all parameters and options.

6.1. General tab

First of all, we need to execute the batch file `crowdsim.bat` to open the simulator. Once the program is open, we see the main tab with five general parameters. Let’s talk about these parameters:

- **Months** and **Weeks**: These two refer to the duration of the simulation.
- **Seed**: It is used internally as a random number generator.
- **Ranking**: It is the minimum ranking a worker must have. If a worker’s ranking goes below this percentage, he or she gets suspended.
- **Speed**: It is the simulation speed. It represents how many simulation weeks are included in each simulation step.
Apart from these parameters, we also find two buttons. The *Simulate* button starts the simulation. The *Save* button saves the simulation output to a text file.

Below the parameters and the buttons there is a square box and a progress bar. Once we start the simulation, the square box shows general results. The progress bar represents the percentage of simulation time already executed.
6.2. Job tab

Here we can specify all the incoming translations. By using the buttons “+” and “-” we can add and remove jobs. Each job is defined by the following parameters:

- **Initial No**: Number of initial translations.
- **Arrival Interval h**: In hours, it specifies the time spent between the arrivals of two consecutive translations.
- **Min deadline w**: In weeks, it represents the minimum deadline for a translation.
- **Max deadline w**: In weeks, it represents the maximum deadline for a translation.
- **Min chunks**: Minimum number of translation chunks.
- **Max chunks**: Maximum number of translation chunks.
- **Sentences mu**: Average sentence size.
- **Sentences sigma**: Standard deviation for sentence size.
6.3. Population tab

In these tab we can define different worker groups with their own particular properties. There is a group added by default. We have the following attributes:

- **Name**: The group’s name.
- **Initial No**: Number of initial workers.
- **Arrival Interval h**: In hours, it specifies the time spend between the arrivals of two new workers.
- **Ability mu**: Average ability percentage.
- **Ability sigma**: Standard deviation for the ability percentage.
- **Speed mu**: Average translation speed.
- **Speed sigma**: Standard deviation for translation speed.
- **Commitment %**: Probability for a worker to change a sentence.
- **Mistakes %**: Probability for a worker to add errors.
- **Random Rank**: If set to 0, rank is calculated by using the worker’s ability. If set to 1, ranking is generated randomly.
- **TtTC**: Time spend between two consecutives task checks.
- **Reliability %**: Probability for a worker to finish a task.
6.4. Output tab

Once the simulation starts, its output will be shown on this tab.

An output line starts with the simulation time. After that, there is a description with the details of the event undergone. This is an example where Good marks and Bad marks refer to the marked errors:

00w01d04h: Chunks 02.03 reviewed by Worker 57. Good marks = 0. Bad marks = 3. Time To Completion = 03h. Round = 1.
6.5. Results tab

It shows the same information as the Output tab but here it is embedded in a table.

We present the following columns: Time, Translation, Chunk, Worker, Procedure and Details. Cells are filled with a hyphen when not needed.
6.6. Translations tab

It shows different information for all the translations and it has the following columns:

- **ID**: The translations’ ID.
- **Chunks**: The number of chunks in which the translation is divided.
- **Arrival**: The time of arrival.
- **Deadline**: The translation’s deadline.
- **Status**: This can be IN PROCESS, COMPLETED or FAILED.
- **Correct**: The number of correct sentences.
- **Total**: The total number of sentences.
6.7. Chunks tab

It shows different information for all the chunks and it has the following columns, which are self-explanatory:

- **Name**
- **Size**
- **Arrival**
- **Errors**
- **Corrected**
- **Status**
- **Round**
- **Post-editor**

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Arrival</th>
<th>Errors</th>
<th>Corrected</th>
<th>Status</th>
<th>Round</th>
<th>Post-editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chunk 00.00</td>
<td>98</td>
<td>00h00m00d00h</td>
<td>87</td>
<td>86</td>
<td>COMPLETED</td>
<td>4</td>
<td>Worker 10</td>
</tr>
<tr>
<td>Chunk 00.01</td>
<td>100</td>
<td>00h00m00d00h</td>
<td>92</td>
<td>90</td>
<td>COMPLETED</td>
<td>4</td>
<td>Worker 11</td>
</tr>
<tr>
<td>Chunk 00.02</td>
<td>121</td>
<td>00h00m00d00h</td>
<td>105</td>
<td>105</td>
<td>COMPLETED</td>
<td>4</td>
<td>Worker 12</td>
</tr>
<tr>
<td>Chunk 00.03</td>
<td>90</td>
<td>00h00m00d00h</td>
<td>81</td>
<td>81</td>
<td>COMPLETED</td>
<td>3</td>
<td>Worker 13</td>
</tr>
<tr>
<td>Chunk 00.04</td>
<td>94</td>
<td>00h00m00d00h</td>
<td>81</td>
<td>81</td>
<td>COMPLETED</td>
<td>3</td>
<td>Worker 14</td>
</tr>
<tr>
<td>Chunk 00.05</td>
<td>108</td>
<td>00h00m00d00h</td>
<td>88</td>
<td>88</td>
<td>COMPLETED</td>
<td>2</td>
<td>Worker 15</td>
</tr>
<tr>
<td>Chunk 00.06</td>
<td>99</td>
<td>00h00m00d00h</td>
<td>89</td>
<td>84</td>
<td>COMPLETED</td>
<td>3</td>
<td>Worker 16</td>
</tr>
<tr>
<td>Chunk 00.07</td>
<td>96</td>
<td>00h00m00d00h</td>
<td>70</td>
<td>70</td>
<td>COMPLETED</td>
<td>3</td>
<td>Worker 17</td>
</tr>
<tr>
<td>Chunk 00.08</td>
<td>78</td>
<td>00h00m00d00h</td>
<td>69</td>
<td>69</td>
<td>COMPLETED</td>
<td>4</td>
<td>Worker 18</td>
</tr>
<tr>
<td>Chunk 01.00</td>
<td>90</td>
<td>00h00m00d00h</td>
<td>70</td>
<td>77</td>
<td>COMPLETED</td>
<td>3</td>
<td>Worker 00</td>
</tr>
<tr>
<td>Chunk 01.01</td>
<td>102</td>
<td>00h00m00d00h</td>
<td>81</td>
<td>89</td>
<td>COMPLETED</td>
<td>3</td>
<td>Worker 01</td>
</tr>
<tr>
<td>Chunk 01.02</td>
<td>98</td>
<td>00h00m00d00h</td>
<td>78</td>
<td>76</td>
<td>COMPLETED</td>
<td>3</td>
<td>Worker 02</td>
</tr>
<tr>
<td>Chunk 01.03</td>
<td>98</td>
<td>00h00m00d00h</td>
<td>73</td>
<td>72</td>
<td>COMPLETED</td>
<td>4</td>
<td>Worker 03</td>
</tr>
<tr>
<td>Chunk 01.04</td>
<td>86</td>
<td>00h00m00d00h</td>
<td>70</td>
<td>74</td>
<td>COMPLETED</td>
<td>3</td>
<td>Worker 04</td>
</tr>
<tr>
<td>Chunk 01.05</td>
<td>94</td>
<td>00h00m00d00h</td>
<td>70</td>
<td>75</td>
<td>COMPLETED</td>
<td>2</td>
<td>Worker 05</td>
</tr>
<tr>
<td>Chunk 01.06</td>
<td>104</td>
<td>00h00m00d00h</td>
<td>80</td>
<td>88</td>
<td>COMPLETED</td>
<td>3</td>
<td>Worker 06</td>
</tr>
<tr>
<td>Chunk 01.07</td>
<td>95</td>
<td>00h00m00d00h</td>
<td>82</td>
<td>80</td>
<td>COMPLETED</td>
<td>4</td>
<td>Worker 07</td>
</tr>
<tr>
<td>Chunk 01.08</td>
<td>101</td>
<td>00h00m00d00h</td>
<td>85</td>
<td>84</td>
<td>COMPLETED</td>
<td>4</td>
<td>Worker 08</td>
</tr>
</tbody>
</table>
6.8. Workers tab

It shows different information for all workers.
6.9. SSD Graph tab

It shows a graph that displays the standard deviation evolution for the difference between the worker’s rankings and their abilities.

\[ \frac{\sum (\text{ranking} - \text{ability})^2}{\text{number of workers}} \]

Ideally, this graph should decrease because we are seeking a ranking that converges with the real abilities.
7. EXECUTION

In this section, we would like to explain our implementation by explaining a single CrowdSim execution. We will undertake this experiment by using the default values we can find the first time we execute our program. We will use the exact same output given by our application.

First of all, we see the arrival of our three initial translations. Translation 00 contains 9 chunks and its deadline is 2 weeks, 5 days and 22 hours. Translation 01 has 10 chunks and its deadline is 2 weeks and 22 hours. Finally, Translation 02 also has 10 chunks and its deadline is 2 weeks, 6 days and 17 hours.

After that, we see how workers translate the corresponding chunks. As we can see in line 4, Worker 13 translates Chunk 00.03 (which is Chunk 03 in Translation 00). This chunk contains 90 sentences, and this worker corrects 47 out of 81 errors and she doesn’t add any. She spends 2 hours in completing the task. Her ranking is 0.46 and her ability is 46.86.

Let’s take a look at line 18. In this case, a worker hasn’t completed the task because deadline hasn’t been met. When this happens, another worker will have to finish this task later.

1. 00w00d00h: Translation 00 arrives. 9 chunks. Deadline = 02w05d22h.
2. 00w00d00h: Translation 01 arrives. 10 chunks. Deadline = 02w22h.
3. 00w00d00h: Translation 02 arrives. 10 chunks. Deadline = 02w06d17h.
4. 00w00d03h: Chunk 00.03 translated by Worker 13. Size = 90. Corrections = 47 out of 81. Added errors = 0. Time To Completion = 02h. Round = 0. Ranking = 0.46. Ability = 46.86.
5. 00w00d04h: Chunk 02.02 translated by Worker 21. Size = 81. Corrections = 26 out of 68. Added errors = 1. Time To Completion = 03h. Round = 0. Ranking = 0.39. Ability = 41.47.
6. 00w00d04h: Chunk 00.08 translated by Worker 18. Size = 78. Corrections = 28 out of 69. Added errors = 0. Time To Completion = 03h. Round = 0. Ranking = 0.44. Ability = 43.85.
7. 00w00d04h: Chunk 01.09 translated by Worker 09. Size = 83. Corrections = 42 out of 73. Added errors = 2. Time To Completion = 03h. Round = 0. Ranking = 0.50. Ability = 58.99.
8. 00w00d04h: Chunk 02.06 not completed post-edition by Worker 26. Task deadline was not met.
9. 00w00d04h: Chunk 00.04 translated by Worker 14. Size = 94. Corrections = 44 out of 81. Added errors = 0. Time To Completion = 03h. Round = 0. Ranking = 0.50. Ability = 54.72.
10.00w00d04h: Chunk 00.07 translated by Worker 17. Size = 96. Corrections = 54 out of 79. Added errors = 1. Time To Completion = 03h. Round = 0. Ranking = 0.50. Ability = 61.95.
11.00w00d04h: Chunk 01.04 translated by Worker 04. Size = 86. Corrections = 38 out of 75. Added errors = 4. Time To Completion = 03h. Round = 0. Ranking = 0.50. Ability = 54.95.
12.00w00d04h: Chunk 02.03 translated by Worker 22. Size = 93. Corrections = 45 out of 80. Added errors = 3. Time To Completion = 03h. Round = 0. Ranking = 0.50. Ability = 53.84.
13.00w00d04h: Chunk 00.05 translated by Worker 15. Size = 108. Corrections = 55 out of 88. Added errors = 0. Time To Completion = 03h. Round = 0. Ranking = 0.50. Ability = 64.81.
14.00w00d04h: Chunk 01.07 translated by Worker 07. Size = 95. Corrections = 50 out of 82. Added errors = 0. Time To Completion = 03h. Round = 0. Ranking = 0.50. Ability = 64.81.
15.00w00d04h: Chunk 01.02 translated by Worker 02. Size = 98. Corrections = 66 out of 77. Added errors = 0. Time To Completion = 03h. Round = 0. Ranking = 0.50. Ability = 81.80.
16.00w00d04h: Chunk 02.09 not completed post-edition by Worker 29. Task deadline was not met.
17.00w00d04h: Chunk 00.00 translated by Worker 10. Size = 98. Corrections = 46 out of 86. Added errors = 0. Time To Completion = 03h. Round = 0. Ranking = 0.50. Ability = 62.14.
18.00w00d04h: Chunk 01.00 translated by Worker 00. Size = 98. Corrections = 54 out of 79. Added errors = 2. Time To Completion = 03h. Round = 0. Ranking = 0.50. Ability = 58.16.
19.00w00d04h: Chunk 01.00 translated by Worker 23. Size = 102. Corrections = 59 out of 89. Added errors = 0. Time To Completion = 04h. Round = 0. Ranking = 0.50. Ability = 62.74.
8. PROJECT PLANNING

8.1. Methodology

We have decided to apply the Agile methodology for the development of our project. This methodology allows adapting the way of working to the conditions of our project. Agile’s life cycle is broken in small iterations. This way we can develop our application incrementally by introducing small changes step by step.

As we said, the use of Agile divides the implementation in small phases and it has proven to be protective against potential risks. This is because at the end of iteration we obtain a functional version (that, of course, doesn’t fulfil all our requirements).

Concretely, we have been using a simplification of SCRUM’s agile methodology which is defined by the following statements:

- All tasks are sorted by priority and they have weights. The more weight a task has, the longer it takes to complete it.
- To control the realisation of tasks by maintaining a list with their state (to do, in progress, waiting or done).
- Sprints consists on periods of time (between one and four weeks) during which all TODO tasks must be completed.

The next illustration shows how this methodology should be implemented.
8.2. Stages

*Introduction*
To start, we have defined the main objectives and the requirements. We have made a list of what the application has to do and we have given a different priority to each of our tasks. We have also made a list with possible extensions for our software.

We have also needed to learn how to use the main programming language we have decided to use (Python) and the main tools needed to develop our software.

*Planning*
A good planning is needed in order to finish our project on time. During this phase we have made a list of tasks with the amount of time each of these tasks requires.

*Design and Specs*
We have specified the use cases and the application’s design. We have introduced all the functionalities that must be available for our users and how the users will interact with our system.

*Development*
During this phase, we made the actual implementation for all the functionalities described before. We have followed the priority order for our different tasks and we have started to write the documentation.

*Closure*
This is the final phase, where the application has been tested and it is working properly. We have finished the documentation and we have done the project presentation.

<table>
<thead>
<tr>
<th>Stages</th>
<th>Days</th>
<th>Hours per day</th>
<th>Hours per stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>18</td>
<td>8</td>
<td>144</td>
</tr>
<tr>
<td>Planning</td>
<td>7</td>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>Design and Specs</td>
<td>29</td>
<td>8</td>
<td>232</td>
</tr>
<tr>
<td>Development</td>
<td>130</td>
<td>6</td>
<td>780</td>
</tr>
<tr>
<td>Closure</td>
<td>5</td>
<td>6</td>
<td>30</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>189</strong></td>
<td></td>
<td><strong>1242</strong></td>
</tr>
</tbody>
</table>
9. ECONOMICAL ANALYSIS

Now we have discussed the time required to perform this project, we are able to analyse its economic costs.

We can see an approximation for the salaries required in the following table:

<table>
<thead>
<tr>
<th>Role</th>
<th>Salary</th>
<th>Hours</th>
<th>Total €</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Manager</td>
<td>65 €/h</td>
<td>226,0</td>
<td>14690,00</td>
</tr>
<tr>
<td>Analyst / Designer</td>
<td>45 €/h</td>
<td>147,50</td>
<td>6637,50</td>
</tr>
<tr>
<td>Programmer</td>
<td>30 €/h</td>
<td>890,50</td>
<td>26715,00</td>
</tr>
<tr>
<td>Tester</td>
<td>20 €/h</td>
<td>77,0</td>
<td>1540,00</td>
</tr>
</tbody>
</table>

Obviously, this project is a special one, as it has been developed in a very particular context, which is the implementation of a final thesis. For that reason, only two people have been working on the development of the application and this table is just an approximation. If this project was performed under normal circumstances, this numbers would be close to the real ones.
10. CONCLUSIONS

Once we have explained the whole project, it is time to go back and see what we have achieved. Our main objective was to develop a crowd sourcing simulator which provided us with the information we needed to deploy our system. We can affirm that the simulator is currently operative and it fulfils all our expectations. We can configure it so we can obtain very precious data that is going to be very useful to complete the final implementation of CA's crowdsourcing platform.

We will now provide some of the most relevant aspects concerning the function of this platform. First of all, the most important aspect we must take into account is that the system has to be attractive for workers to join. As we all know, motivation is essential if we want the translation process to be fast, effective and fluent. In order to achieve this objective, we need a proper rewarding system. In this view, the way our workers are paid becomes one of the most important features on the design process.

There might be many different reasons for people to participate in a crowd-based process, ranging from their willingness to participate in a collaborative process to build something new, their motivation to help the community or their interest to be economically rewarded. Most industrial applications pursue lucrative objectives. As a result, these industrial applications based on the crowd are much more constrained in terms of motivating the crowd and tend to reward workers economically. We may classify the different crowd systems depending on the rewarding model they use:

- **Best-gets-paid systems**: Usually, in this type of system, only the best workers get rewarded. In general, the system provides the tools to present ideas or solutions to a specific problem and a voting system for the crowd to decide the best proposals. This philosophy usually allows costs to be drastically reduced and obtain very good quality, although it is in general unfair for workers, given that most of them work and are not rewarded, potentially becoming a source for lack of motivation.

- **Pay-per-Work systems**: In this case, workers are in general rewarded by the amount of work done. This is for instance the philosophy of Amazon’s Mechanical Turk, where workers execute Human Intelligence Tasks (HITs) and get a predefined amount of money for it.
However, these two systems do not take quality into account. We propose to build crowdcomputing systems based on a variant of a Pay-per-Work system. We call them Pay-per-Quality systems. The fundamental idea is that workers get paid for their work, but the amount that each worker receives depends on their profile. In other words, the rewarding system depends on the rankings of the workers. In this way, a trustworthy worker will be better rewarded than an inexperienced worker or a worker with lower skills in general.

It has been a great pleasure to learn about the existence of this brand new technology. I had never heard of crowdsourcing and all its implications before, so to me it has been an excellent opportunity to work with this kind of systems.

I have also been able to put in practice all the skills I have been learning throughout the whole Informatics Degree, especially in terms of programming and software design. If I had to choose the most useful subjects I have needed to fulfil this project, I would obviously list all the programming ones: P1, PRED, PRAP, PROP, as well as the software engineering ones: ES1, ES2 and PESBD. I have realised the importance of a correct software design with the aim of optimising the time it takes to implement an application. I had seen several concepts previously and I have been able to observe their actual concretion during the construction of our simulator.

At the same time, it was a good decision to use Python. Its simplicity has been fundamental. I have enjoyed programming a lot and I have learned the use of a programming language that I did not know.

And most of all, I have learned how to face a technological problem by using all the tools provided at University for the last few years. From my point of view, this is the most important and valuable experience I get from all the work that I have done.
11. ACKNOWLEDGEMENTS

I appreciate a lot the opportunity that CA Technologies and UPC Barcelona Tech have brought to me by allowing my participation in such an exciting and innovative project. After working during the last six months on this simulator, I am definitely convinced that the implementation of the crowdsourcing platform is a brilliant idea and it will reduce translation costs for the company as well as improve the time needed for localization purposes.

Therefore, thanks to Marc Solé Simó for all his help. This project could not have been properly completed without his inestimable support and collaboration. And also, thanks to Josep Lluís Larriba Pey and Victor Muntés i Mulero for relying on me and allowing me to add my contribution on the construction of this magnificent project.

Again, thanks a lot to the three of them.
12. REFERENCES


3 Harris, C., You’re Hired! An Examination of Crowdsourcing Incentive Models in Human Resource Tasks, in Proceedings of the Workshop on Crowdsourcing for Search and Data Mining (CSDM) at the Fourth ACM International Conference on Web Search and Data Mining (WSDM), eds. Lease, M., Carvalho, V., and Yilmaz, E. (Hong Kong, China, 2011), pp. 15-18.


6 Gao, Q. and Vogel, S., Consensus versus expertise: a case study of word alignment with mechanical turk, in Proceedings of the NAACL HLT 2010 Workshop on Creating


12 Negri, M., Bentivogli, L., Mehdad, Y., Giampiccolo, D., and Marchetti, A., Divide and conquer: crowdsourcing the creation of cross-lingual textual entailment corpora, in