POLITECNICO DI MILANO
DEPARTMENT OF ELECTRONICS INFORMATICS AND BIOENGINEERING
DOCTORAL PROGRAMME IN COMPUTER SCIENCE AND ENGINEERING

METHODOLOGIES FOR THE DEVELOPMENT OF CROWD AND SOCIAL-BASED APPLICATIONS

Doctoral Dissertation of:
Andrea Mauri

Supervisor:
Prof. Marco Brambilla

Tutor:
Prof. Stefano Ceri

The Chair of the Doctoral Program:
Prof. Fiorini Carlo Ettore

2015 – XXVIII

Acknowledgements

Here is the hardest part for me to write, since I have a lot of people to thank and so few ways to express my gratitude toward them.

I want to thanks my advisor Marco Brambilla and Stefano Ceri for introducing me to the research world and guiding me during my study period.

I thank my parents for their endless support, starting back from the beginning of the bachelor to the end of the PhD.

Even if we worked together only in the beginning I'm very grateful to Alessandro Bozzon, always ready to give advices whenever we met around the world.

Thanks to Daniele for being always ready to invest time in side projects that, up to now, always succeeded.

A very special thank to my friends and colleagues Chiara and Riccardo, who shared with me the joy of success and the sadness of failure, and made all these years more enjoyable. Without them the PhD period would have been harder and dull.

Finally I want to give a big thank all my friends Luca Galli, Luca Losa, Davide (Ser), Davide (Gadio), Davide (Mani), Pier, Andrea Pozzetti, Andrea Vaccarella, Marco Balduini, Marco (Gas), Daniele and Lorenzo[1] Cossa.

[1]for his strength

Abstract

Even though search systems are very efficient in retrieving world-wide information, they cannot capture some peculiar aspects of user needs, such as subjective opinions, or information that require local or domain specific expertise. In these scenarios the knowledge of an expert or a friend's advice can be more useful than any information retrieved by a search system. This way of exploiting human knowledge for information seeking and computational task is called Crowdsourcing. The main objective of this work is to develop methodologies for the creation of applications based on Crowdsourcing and social interaction. The outcome is a framework based on model-driven approach that allow end user to develop their own application with a fraction of the effort required by the traditional approaches. It guarantees a strong control of the execution of the crowdsourcing task by mean of a declarative specification of objectives and quality measures. A prototype has been developed that allow the creation and execution of task on various platforms. Validation of the approach has been carried out with quantitative and qualitative analysis of results and performance of the system upon some sample scenarios, where real users from social networks and crowdsourcing platforms have been involved.

Riassunto

Nonostante i sistemi di ricerca siano molto efficienti nel recupero delle informazioni in tutto il mondo, non possono catturare alcuni aspetti peculiari di esigenze degli utenti, come opinioni soggettive, o informazioni che richiedono competenze specifiche di dominio. In questi scenari l'abilitá di una persona, la conoscenza di un esperto o il consiglio di un amico puó essere piú utile di qualsiasi informazione fornitas da un sistema di ricerca. Questo modo di sfruttare la conoscenza umana per ricerca di informazioni e attivitá di calcolo si chiama Crowdsourcing. L'obiettivo principale di questo lavoro é quello di sviluppare metodologie per la creazione di applicazioni basate su Crowdsourcing e di interazione sociale. Il risultato é un framework basato su approccio orientato al modello che permette all'utente finale di sviluppare una propria applicazione con una frazione dello sforzo richiesto dai metodi tradizionali. Garantisce un forte controllo del lavoro svolto dalla crowd per mezzo di una specifica dichiarativa degli obiettivi e di metriche di qualitá. E' stato sviluppato un prototipo che consente di poter creare e di far eseguire lavori alla crowd su varie piattaforme. La validazione dell'approccio é stata effettuata tramite un'analisi quantitativa e qualitativa dei risultati e le prestazioni del sistema sono state valutate su alcuni scenari di esempio, in cui sono stati coinvolti gli utenti reali da social network e piattaforme di crowdsourcing.

Contents

CHAPTER 1

Introduction and Problem Statement

1.1 Introduction

A new class of software applications, called *crowd-based applications*, is emerging. These applications use crowds, engaged through a variety of platforms, for performing tasks; the most typical application scenarios include fact checking, opinion mining, localized information gathering, marketing campaigns, expert response gathering, image recognition and commenting, multimedia decoding and tagging, and so on. Crowds have been used also for more creative tasks, including scientific discovery[1] exploiting game-with-a-purpose (GWAP) approaches. All these applications delegate to people the activities that are better performed by humans than by computers.

The common aspect of these applications is the interaction between the requestor (who poses a task), the system (which organizes the computation by mixing conventional and crowd-based modules), and a potentially wide set of performers (who are in charge of performing crowd tasks and are typically unknown to the requestor). The system may take multiple forms: in addition to crowdsourcing platforms (such as Amazon Turk

[1]E.g., FoldIt, a GWAP for protein folding `http://fold.it`.

or CrowdFlower) or question-answering systems (such as Quora or Yahoo!Answers), a recent trend is to use social networks (such as Facebook, Twitter or LinkedIn) as sources of human labor to be integrated in software applications, thanks to the availability of their programming interfaces.

Crowd-based computations undergo a new set of design principles and phases, as dealing with crowds introduces many peculiar aspects:

- Performers should be selected, possibly on the basis of their expertise on the task that they should perform.
- They must be reached, typically through an invitation system.
- They must be motivated through incentives, that include non-monetary ones, such as fun, self-esteem, altruism, visibility and reputation.
- Their work should be controlled, and in particular a task should be deemed as complete when it meets certain quality criteria, which take into account elapsed time, cost, and quality of result.
- Performers should be controlled, in particular by detecting malicious and/or incompetent performers, who should be banned from the computation.

Crowd-based computations can be part of a complex workflow and can be intertwined with conventional software computations; it is important to provide them with clear interfaces to the rest of the software, so that it becomes possible to discuss their requirements, design, testing, deployment, control, and maintenance, both from a local perspective and from a global, system-oriented perspective.

1.2 Problem Statement

In this situation different issues arise: each type of scenario is characterized by a peculiar set of needs and requirements, that need to be mapped to the particular platform,that usually is not flexible, as it do not support a high-level, fine-tuned control upon posting and retracting tasks.
For instance if the requester wants to post and control a crowdsourcing task on Amazon Mechanical Turk he has to code the implementation with imperative and low-level programming language or using a framework like Turkit [53]. If he wants to exploit the relations between people, he may want to use a social network as crowdsourcing platform. In this case the requester has to directly use the API provided by the social network.
Thus the research questions that lead this work are:

1 What are the main features of a crowdsourcing application ?

2 How can I abstract all these characteristics in a agnostic metamodel?

3 How can this model be used to facilitate the development of a crowd or social based application maximizing its performance?

1.3 Contribution

The objective of this work is to develop methodologies for creating applications that leverage the knowledge of the crowd or social communities. The approach developed should be platform agnostic and allow the requester to create his application without having strong technical knowledge.
This approach defines a design methodology, a specification paradigm and a reactive execution control environment for designing, deploying, and monitoring crowd-based modules. It is focused on data-centric applications, which apply structured human processing to large datasets; these applications represents the largest and currently most important class of crowd-based applications, as they include big data preparation, cleaning, and consistency checking. I disregard crowd-based applications which are selectively addressed to few performers (possibly a single one), such as interaction with experts or localized question-answering.

1.4 Thesis organization

The remainder of this thesis is structured as follow:

- Chapter 2 provides a detailed overview of the state of the art regarding crowdsourcing methodologies.
- Chapter 3 describes the proposed model for building crowdsourcing application and the related design process.
- Chapter 4 deals with the problem of controlling the work of the crowd, so that cost, quality and time constraints are satisfied.
- Chapter 5 analyses the problem of building interoperable crowdsourcing applications, where the execution of the task is carried out upon different execution environments.

- Chapter 6 presents a systematic approach to the design and deployment of crowd-based applications as arbitrarily complex workflows of elementary tasks
- Chapter 7 addresses the problem of understanding how incentives (non monetary) influence the engagement of people performing a crowdsourcing task.
- Chapter 8 describes the prototype (CrowdSearcher) that has been built in order to perform the experiments needed for the validation of this work.
- Chapter 9 summarizes the results obtained and proposes possible future research directions.

CHAPTER 2

State of the art

2.1 Introduction

In this chapter I describe the state of the art of Crowdsourcing application, in particular I focus on four main aspects:

- Approaches for programming the Crowd (Section 2.2): I focus on previous research studies and tools that aim to facilitate the development and the control of Crowdsourcing applications. These works address the problem in different ways: they provide an imperative or declarative approach for defining the model of the application and the rules for the control.

- Crowdsourcing Application Design (Section 2.3): I focus on studies that focus on more theoretical aspects of modeling, proposing pattern that should enhance the quality of the result of a crowdsourcing task or studying how different design dimension impact on the performance.

- Optimization of Crowdsourcing Task (Section 2.4): I focus on previous work that try to address the problem of finding the optimal design for a crowdsourcing task. Usually they focus on a single single aspect of the crowdsourcing application (e.g. results aggregation, performer selections, etc..) and build a mathematical model of the problem.

- User Engagement (Section 2.5): I focus on previous work regarding methods for boosting the participation for users in Crowdsourcing activity, both in term of quality of the results and number of workers. Usually this is achieved using either monetary of non monetary rewards. Studies about the former focus on how much to pay and how to regulate financial bonuses in order to enhance the quality of the results, while the second studies how to leverage factors such as motivation and entertainment (i.e. serious games)

2.2 Approaches for programming the crowd

In this section I describe various programming approaches proposed in these years in order to facilitate the development of application that leverage the crowd. Most approaches rely on an **imperative programming model** to specify the interaction with crowdsourcing services.

Turkit [55] offers a Java/Javascript API for programming iterative tasks on Amazon Mechanical Turk. A developer that uses Turkit has access to all the Javascript features and to an abstraction layer of the Amazon Mechanical Turk API, allowing to create complex set of HITs as steps of an algorithm. They propose a new programming model called *crash-and-rerun* in order to address the problems of errors handling and fault detection since it allows program to be re-executed without repeating the costly parts. This is achieved by caching the results of costly operations so that, if the program crashes, their outcome are easily retrieved. This is very useful in the context of crowdsourcing because operations cost also money. This also allows the programmer to tune the program between executions, given that the order of important operations is not altered. The authors also provide a web based IDE for developing and running TurkIt scripts, that also allows to keep track of the execution steps. Finally the authors evaluate their work with different tasks involving also people outside the project. They noticed that TurkIt is useful for writing simple scripts, but often people get confused with the *crash-and-rerun* programming model. Furthermore it favors usability over efficiency, for instance if a script need to be rerun, all the loops present need to be re-executed. Another benefit is the experiment replication, since it’s enough to share the source code and the related database.

RABJ [49] is a proprietary human computation engine developed by Metaweb to enhance the data processing pipelines of Freebase with human judgments. The data model is very simple, it's composed by three entities: Question, Judgment and Queue. Question is the smallest unit of work, typically it ask the crowd to evaluate some piece of information or to input some data. The Judgment is the response provided by the worker. The Queue is a set of Questions bound to an application. The model is content agnostic, so it can cover very different use cases, the downside of this is that complex control is delegated to the applications. The authors evaluated this approach with three application: Typewriter, an application built for connecting entity to a class (e.g. is George Clooney an actor?), Geographer, for associating a location with its geographic coordinates, and Genderizer, for determining the gender of an entity by looking only to its picture or reading a description. Finally they report the lesson learned: the relation between requester and workers is important, the authors didn't apply complex anti spam methods, but they invested time in training the workers and in setting up special communication channel for receiving feedbacks. They noticed that binary questions are more effective than their multi selection counterpart, less options lead to less conflict, moreover is important to estimate the correct amount of response needed. RABJ allows to dynamically tune the number of judgments needed, but the actual control must be performed by the client application. Finally incentives for volunteer are very different from the ones for paid worker: a volunteer wants to be recognized for his work and enjoy to compete against the others.

Jabberwocky [3] it's a framework for human computation composed by three component: *Dormouse*, a virtual machine that provides cross platform social computation capabilities by abstracting the implementation details of the underlying platforms; *ManReduce*, a parallel programming framework inspired by MapReduce; and *Dog*, a high level procedural programming language that allow to program crowdsourcing tasks. *Dormouse* sits on top of the crowdsourcing platforms and it hides their implementation details, offering a cross-platforms programming environment. It also allow the developer to build custom social structure and enrich the information related to the workers. *ManReduce*, based on MapReduce, allows the system to be ready for data intensive application. Moreover it allows both machine and human to perform both the map and reduce steps. Since the *ManReduce* is too low level for some application, the authors propose also *Dog*, a high level programming language that sit on top of *ManReduce*. It offers a set of libraries that implement the most common crowdsourcing task type such as Vote, Compare, Classify and machine functions such as Histogram, Median, etc.. The language is also extensible by the developers that can write both human and machine libraries function by accessing directly to *ManReduce*.

Other works propose approaches for human computation which are based on **high level abstractions, sometimes of declarative nature**.

In [63], authors describe a language that interleaves human-computable functions, standard relational operators and algorithmic computation in a declarative fashion. Then they extend their model in order to take into the account uncertainty, cost, scheduling and spam. This is a very abstract study, since they do not study the interaction with a real crowdsourcing environment. They propose a Datalog like model for specifying crowdsourcing task and they focus on the optimization of the query processing in presence of uncertainty.

Qurk [57] is a query system for human computation workflows that exploits a relational data model, SQL to express queries, and a UDF-like approach to specify human tasks. In particular, the user writes SQL queries to require information, and, UDF statement to involve the crowd in the process. Each UDF statement is a template for a HIT on Amazon Mechanical Turk. They propose different optimizations for the query execution. For instance it can modify the price of a HIT on Amazon Mechanical Turk at runtime; in case of a big dataset it can sample the input in order to cover uniformly the input space, or it can group the inputs according to some feature in order to reduce the number of HITs; training a model in order to substitute the crowd as they provide answers (for example in a image recognition task).

CrowdDB [37] also adopts a declarative approach by using CrowdSQL (an extension of SQL) both as a language for modeling data and to ask queries; human tasks are modelled as crowd operators in query plan, from which it is possible to semi-automatically derive task execution interfaces. They apply the same query optimization and processing as in a traditional SQL database, such as predicate push-down, stopafter push-down, join-ordering and determining if the plan is bounded. They evaluated their approach on Amazon Mechanical Turk. At first they performed a micro benchmark where they asked the crowd to fill out a form with missing data. The workers on AMT had to find the phone number and the address of the company given its name. The authors made three variation of the experiment in order to measure different metrics. In the first case the authors published the task grouped in HITGroups having different size (from 10 to 400 HITs) and they measured the response time. They noticed that a trade off between throughput and completion is present: the best throughput is obtained with the largest HITGroup while the best completion with HITGroup size of 50 or 100 HITs. In the second the authors published the HITs varying the amount of money paid to the crowd (from 0.01$ to 0.04$) and they measured the response times. As expected, higher is the reward, faster the crowd answer. At last they studied the quality of the results. The ground truth was generated by applying the majority voting on the five answers for phone numbers. They analyzed the workers that answered to the task of the experiment, and they noticed that their distribution was skewed (there were few workers executing many HITs). The authors expected an increase of precision (as the workers learn to perform the task) but they found that the error rate remained constant. Finally they tried to change the reward (from 0.01$ to 0.04$) but they noticed no correlation with the quality of the results (at least within this range of payments). Then they experimented with three more complex queries. For the first query the authors used a dataset composed by companies (name, address). They submitted to the crowd the query `SELECT name FROM company WHERE name ="[a non-uniform name of the company]"`. The workers on Amazon Mechanical Turk had to compare 10 company names to the non uniform one. They measured the time and quality, in term of number of correct name found. In the second query the authors asked the crowd to rank of pictures in different subject areas. In particular the workers had to compare four pairs of pictures. The authors measured the time and the quality by comparing the final result with a groundtruth built by six experts. In the last experiment the authors sent the query:

```
SELECT p.name, p.email, d.name, d.phone
FROM Professor p, Department d
WHERE p.department = d.name AND
p.university = d.university AND
p.name = "[name of a professor]"
```

They executed it following two different plans: in the first they asked for the Professor information and the department the professor is associated with. Then in a second step, they asked to the crowd for the remaining information of the departments. In the second plan they asked in a single step for all the information. They measured completion time, quality and costs.

DeCo [64] system allows SQL queries to be executed on a crowd-enriched datasource; however, human tasks are defined as *fetch* and *resolution* rules programmed in a scripting language (Python) and defined in the schema of the data source. Fetch rules tell the system how to retrieve the data from the crowd while resolution rules are used to clean the data. As the authors in [37] they focus on the query processing and optimization, exploring different fetch rules and query plan. In general the found that fetch rules that ask for multiple information perform better than the simple one.

2.3 Crowdsourcing Application Design

In this section I describe works that proposed different models for modeling crowdsourcing task and studied how different design choices influence the performance of the crowd.

The authors in [47] performed a broad study on the influence that the various dimensions have on the quality of the result obtained by the crowd. They experimented with different levels of payment (0.10$ and 0.25$), task difficulties (5 or 10 objects per HIT) and qualifying criteria (present or not present). They found that increasing the pay increases also the quality of the result, but also attracts unethical worker, so a more refined quality control is required. Modifying the difficulty of the task has two consequences: it leads to less accurate results but it attracts well performing workers. Finally applying qualifying criteria increase directly the overall quality of the results.

In [56] the authors focused on order and join task types, and they experimented with different types of implementation. Regarding the *join* they propose: *SimpleJoin*, *NaiveBatching*, *SmartBatching*. *SimpleJoin* consists in showing to the worker two objects and ask him to tell if they are related or not, *NaiveBatching* shows n pairs of objects per HIT and finally *SmartBatching* show the objects belonging to the sets to join in two different columns, and the worker has to choose which ones can be joined. They ran these implementation on Amazon Mechanical Turk and observed their impact in term of quality, time and cost. They found that batching has a little effect on quality and time, but it decrease the cost. Regarding the *sort* problem they propose a *Comparison* based or *Rating* based algorithm, and a *Hybrid* approach. The *Comparison* methods ask directly to the worker to rank the objects, while the *Rating* approach ask the worker to rate the object according a numerical scale. The *Hybrid* approach, instead, starts from the evaluation gathered with the *Rating* approach and iteratively refine it using the *Comparison* method. They compared the performance of the various approaches and the found that the *Rating* method achieved a lower accuracy than *Comparison* even though its cheaper. The *Hybrid* approach achieved the same result as *Comparison* but at one third of the cost.

In [54] the authors experiments with parallel and iterative human computation processes, in order to understand the tradeoff of each approach. The processes are defined using the concept of *Creation Task*, when the worker create new content, and *Decision Task*, when he either select the object through comparison or by giving a score. In the experiments the authors defined iterative and parallel process in order to ask the crowd to solve the same problem. The question asked to the crowd were: writing image description, brainstorming and blurry text recognition. In the first experiment the crowd was given the work to write the description of a image, and as expected the iterative process achieved the best result. In the second the crowd had to come up with a company name given its description. In this case the parallel process came up with the name that received the highest rating (the rating is also performed by the crowd), however the average score is higher for the iterative one. In the last experiment the workers needed to transcribe text from a blurry image. In this case the authors did not find significant difference between the two type of process, probably because in the iterative process, one iteration can be influenced by a poor guess performed by a previous worker.

In [11] the authors propose a task design pattern called *Find - Fix - Verify*, in which the work performed by the crowd is split in three phases. They apply this pattern in order to aid the writing process by integrating the worker in Microsoft World plugin that is able to: shorten the text, finding errors and formatting or finding figure. In general the *Find* phase corresponds to the task of highlight which part of the text can be addressed, the *Fix* phase consists in to actually perform the task (correcting the error or summarizing the sentence) and, finally, in the *Verify* phase workers evaluate the work performed by the others. The authors were able to achieve satisfying results in all the three cases: shortening text from 10% to 22%, correcting the 67% of typos (versus the 15% of the tool provided by Word). The last scenario consisted in sending open ended instruction to the crowd through the tool (like changing the text of the document to past tense to present tense). In this case the 30% of the worker result contained errors, most of the task were done correctly but they made some mistake in the details.

In [15] the authors propose a model for integrating the contribution of the crowd with the results of a search engine. They define Query Language as a mapping from an Input Model, including a dataset of object and the operations the crowd need to perform, to an Output Model, which is obtained by modifying the dataset and by adding the answers to structured queries.

The mapping defines how the output model will be created started from the input. It contains information about the crowdsourcing platform that will be used, the group of worker that will be involved, the execution constraints, how the work should be split and assigned and the template to be used to render the UI of the task. Moreover they focus on the deployment on social networks supporting three types of interactions: using directly the interface of the platforms (e.g the Facebook likes), embedding the interface in the platform (e.g. A Facebook application), or using an external standalone application. They also build a prototype that allows the creation of crowdsourcing task and the deployment on Facebook and Doodle. In the experiments the both asked the users to answer and create their own crowdsourcing tasks. They found that the participation is higher if friends are involved in the execution of a query. Facebook is most effective in retrieve answers in the first few hours, but Doodle in the end retrieve more results.

2.4 Optimization of Crowdsourcing Task Design

In this section I describe works that propose methods for optimizing the results of a crowdsourcing application.

Usually they focus on a single aspect. For example, regarding result aggregation, in [52] the authors propose probabilistic model and a EM (Expectation Maximization) algorithm in order to infer the correct answer of a task that can have infinite number of possible answers. Whether or not worker get the correct answer, depends on his error parameter and the difficulty of the task. Then the EM algorithm jointly learns maximum-likelihood estimates of the difficulty of the task, the error parameter of the worker while inferring the correct value of the task. The author perform both simulated and real experiments on Amazon Mechanical Turk. The synthetic experiments had the purpose to tune the algorithm and to see his resistance to noisy workers, while the real one show that their methods outperform the classic majority votes.

In [5] the authors propose a method based on active learning for inferring the correct solution given a set of answers. In their model each task is characterized by a difficulty d and a set of possible answer R (where only one is correct), the workers have a certain level of skill a. Given the set of worker answers, the model is able to infer the correct solution. They performed synthetic experiment on the TREC 2011 Crowdsourcing Track dataset [51] and show that they slightly outperform the majority vote. An increment of performance is obtained if the worker are first tested on a set of questions in which the true answer is known.

The authors in [46] show how to use the probabilistic matrix factorization approach in order to aggregate results of a labeling task. The idea is that methods like majority voting do not take into the account that usually a single worker label only a small set of objects, so only few workers influence the final outcome of a task. The authors address this problem by proposing a collaborative filtering approach. This method can infer the worker's missing label, so that all the workers contribute to reach the consensus. The authors evaluate their approach using the 2010 TREC Relevance Feedback Track [6] and show that their methods outperforms majority voting and general EM approaches.

In [73] the authors propose a Bayesian model for aggregating the results of a labeling task taking into the account the quality of the workers. The novelty of this approach is that, instead of estimating the reliability of a single worker, they classify the workers in community according their confusion matrix. Then the accuracy of the worker is given by the accuracy of the community. The author evaluate their approach on two different dataset (from CrowdFlower and ClickWorker) with respect to majority vote, EM and a single worker Bayesian model. Their approach performs better with small sets of labels by exploiting community patterns between workers and transferring learning of community knowledge across community members. In large sets, it performs comparably or slightly better than the other methods. In average it performs better than the other baselines.

Regarding task decomposition and assignment in [80] the authors propose a probabilistic framework for choosing which HIT is better to send to the crowd from a set of candidates based on the informativeness of the HIT. They consider only HITs in which the task is answering a yes/no question (they reduce a labeling problem to a "Is object X a l?" question). Given all the possible outcome for a given object, the uncertainty is modeled as the Shannon Entropy, then the utility of a HIT is modeled by its expected uncertainty reduction. The authors evaluate their approach in three different scenario, showing that their approach succeeds in reducing the uncertainty and increase the accuracy with respect to random selection.

Furthermore [45] proposes a model for decomposing a task into simpler sub-task, focusing on the quality of the final results. In this work the authors formally define and compare two models for task decomposition (vertical and horizontal) in order to give explicit guidelines to task requester. In vertical decomposition the task is split according to the operation that need to be performed (an example is the Find-Fix-Verify approach [11]) while in the horizontal method, the task is split according the number of objects. In general in the first case the obtained subtasks are interdependent, while in the second they are independent. Then the authors define the concept of quality and difficulty for the subtasks. Then they run a simulation for both the models and find that decomposing a task into subtasks is worse than assigning the whole task to one worker, but in this case it can be difficult to find a worker willing to perform the task. In general, vertical task decomposition strategy outperforms the horizontal one in improving the quality of the final solution

Most of these works focus on a single type of operation or scenario and address only the problem of aggregating the results of the single evaluations given by each performer. Moreover they often fail to guarantee a consistency check of the effectiveness of the optimal solution in terms of real-life application execution; also, no mathematical model can cover the variety of optimization dimensions and constraints, as each model typically ad- dresses a small set of decisions for a specific crowdsourcing ask; even under such limitations, many mathematical models are hardly tractable, as the underlying problems fall into exponential of NP-hard classes.

2.5 User Engagement

Works about user engagement fall into two main categories: they study either social or monetary rewards.

Even though in this thesis I experimented with purely external and immaterial incentives, i.e., visibility and reputation gains obtained by successfully performing the given task [65], I present previous works related both monetary and non monetary incentives.

Several studies has been done on how monetary incentives influences the performance on worker in performing Crowdsourcing task, often showing contradicting results.

For instance Harris [42] experimented with both positive (i.e. giving a bonus if the worker successfully performed the task) and negative (punishing the performer by reducing his payment if the results were not good enough) incentives and found out that in both cases the workers performed better than not applying any bonus.

Both [69] and [77] experimented with different incentives schemes and sizes, finding that they don't have any impact on the quality of the results. The authors in [69] experimented with very small bonuses, so a possibility that it's too small to justify an increase of effort. Instead in [77] the authors used offered large bonuses (with respect to the base payment), so probably even the smallest bonus was enough to obtain the maximum effort from the workers.

On the other hand Yin et al. [78] studied performance base incentives in the context of switch task (i.e. where workers switch back and forward between different types of task), and found out that incentives do increase the performance, in particular when the type of task change occurs.

In [58] the authors shown that financial incentives increase the quantity of work performed by the crowd but not its quality. They show that more important is how the incentives are organized: a good incentives scheme can lead to higher quality work paid less. Furthermore they suggest that social rewards could be as effective as financial ones.

These findings are in line with other works [25]. The authors in [41] observed this phenomena also on a different platform such as oDesk[1]. Furthermore [66] also show that intrinsic (i.e. motivation) incentives lead to better result.

If the worker is paid with social rewards we are in the context of *gamification*, [32], where games with a purpose are used for obtaining a specific objective in terms of user behavior or information collection.

Gamification has been successfully applied in different scenario, such as: mining data from images [40], collecting urban data [26,27], predicting proteins structure [48]. Reward and reputation systems are at the core of gamified applications, which builds upon incentive centered design.

These techniques are studied in persuasive technology [36], where games are seen as potential means to shape user behavior [1, 61], or to instill desired values or messages [9].

My work on user engagement is founded the well known fact that psychological factors that influence human behavior are fundamentally of two types, namely, incentive and cost [59, 67]. The former increases and the latter decreases the motivation to complete an action. People get incentives when they are awarded with material or immaterial value, spanning from money or gifts, to points, or simply visibility or popularity.

An important aspect that drives the worker in case of non monetary incentives is the motivation. Psychology distinguishes also between external and internal motivations [10, 14], i.e., socially–derived motivations (e.g., public recognition) and individual–based motivations (e.g., enjoyment in games [71]).

Besides the general classification though, researchers agree that motivations are very heterogeneous, and in particular they vary a lot depending on uniqueness of each specific scenario [75] and incentives can be dynamically tuned [28]. The incentive problem has been widely studied within

[1]www.odesk.com

organizations [68]: for instance, single incentive measure are known to induce unwanted responses, and therefore multiple incentives are usually combined to counteract the such behaviors.

Various works studied the incentives and conditions that favor participation in social networks. Yogo et al. study incentives that stimulate activities in social networking services [79]. The work [72] proposes a user behavior model considering link strength changes, communication time and participation time; based on that, they examine the conditions that encourage users to participate more intensively in social networking. Ermecke et al. [35] study agents on Facebook in order to understand who contributes to diffusion of the information. They found out that user behavior has strong influence on peers. So in order to boos user engagement also the peer-to-peer relations need to be exploited.

The work [33] studied the tradeoff between the *cost* represented by the concern about personal data privacy and the *incentive* of sharing and showing personal facts, records or content to social connections.

Incentives are also studied in other vertical scenarios, for instance in [29] the authors studied the incentives in the context of production of semantic context. In particular they propose a developing process for building an incentive mechanism that better fit your scenario.

CHAPTER 3

Models for Crowdsourcing Application Design

Part of the work described in this chapter was published in [17]. Furthermore the work described in Section 3.6 was published in [24],

3.1 Introduction

In general a Crowdsourcing application can be modeled with three entities: a requester, a system and the crowd. The **requester** is the one who has a **problem** he want to solve with the **crowd**. In order to do so he has to utilize the **system** in order to interact with the crowd.
The problem can comprehend a wide range of use cases: translating a text, transcribing an audio file, annotating images, and so on. Currently crowdsourcing has been applied in very different scenarios like: databases, information retrieval, artificial intelligence and social science.
The crowd can be very heterogeneous and can range from the anonymous worker on Amazon Mechanical Turk to your friend on Facebook or an expert on a Question Answering platform like StackOverflow. Usually the type of crowd the requester want to involve depends on the type of problem

need to be solved.
The system should be able to easily translate the problem in a format that the crowd can solve and allow the correct match between requester's needs and crowd type.
In order to do so I defined three models that together allow to design the human computation application in all its aspects, furthermore they will enable the definition of strategy for creating particulars patterns to be applied to a task (for example limiting the number of object to be evaluated for each task) and rules to control its execution. These model are:

- Crowd Model: it describes the characteristics of the considered crowd.
- Problem Model: it represents the problem that is going to be solved with a human computation task, for example the type of operations, the type of data (structured, unstructured) and peculiar configurations.
- Task Model: represents the structure of a task. It will contain general information (title, description, etc..) and the objects interested by the crowdsourcing campaign.

3.2 Crowd Model

Figure 3.1 shows the crowd model. A crowd is composed by a set of Performers, that are the individual that are going to participate to the crowdsourcing task. A performer can belong to different Communities, a set of people that share common interests (e.g., football club fans, opera amateurs, ...), have some common feature (e.g., living in the same country or city, or holding the same degree title) or belong to a common recognized entity (e.g., employee in an office, work-group or employer; students in a university; professionals in a professional association; ...).
A Platform represents the concrete place where the crowdsourcing computation is carried out, for instance it can be a crowdsourcing market place as Amazon Mechanical Turk, a social network as Facebook or LinkedIn.
A Performer can exist in different Platform at the same time, hence it has associated different Profiles. Finally a Performer can have Relations with others, meaning that it has some kind of link with them on a Platform (e.g friends on Facebook or followers on Twitter).

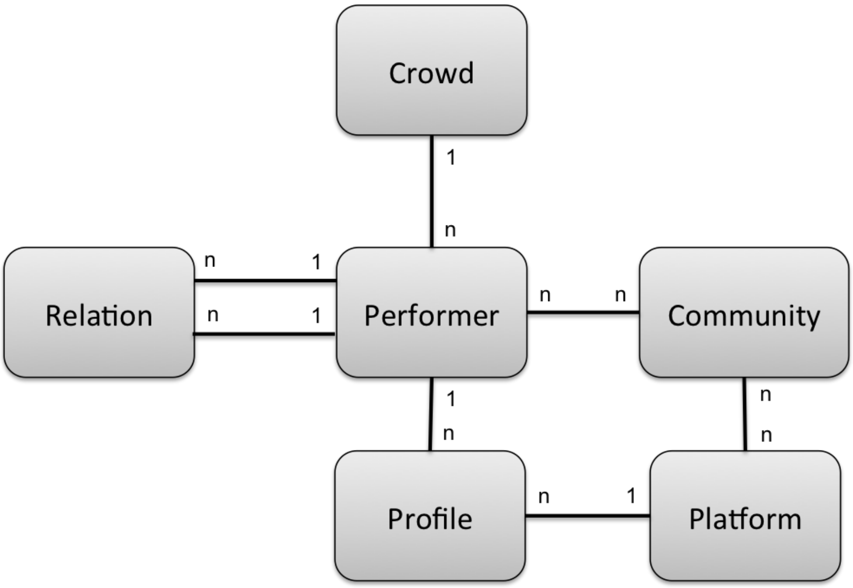


Figure 3.1: *Model describing the crowd.*

3.3 Problem Model

Figure 3.2 shows the general model of a problem that need to be solved via crowdsourcing.
Usually a Problem consists in one Operation that need to be performed on a set of Objects in order to obtain a result. Notice that the operation here, is more an high level description of the outcome of the crowdsourcing campaign than the operation that the crowd will perform on the objects.
Different problems can be addressed with crowdsourcing, but recently Gadiraju et al. proposed the following taxonomy [39]:

- Information Finding: consists in asking the crowd to provide information regarding some topics.
- Verification and Validation: consists in asking the crowd to verify some facts.
- Interpretation and Analysis: consists in asking the crowd to evaluate some objects.
- Content Creation: consists in asking the crowd to produce new content for documents or website. The difference between the Informa-

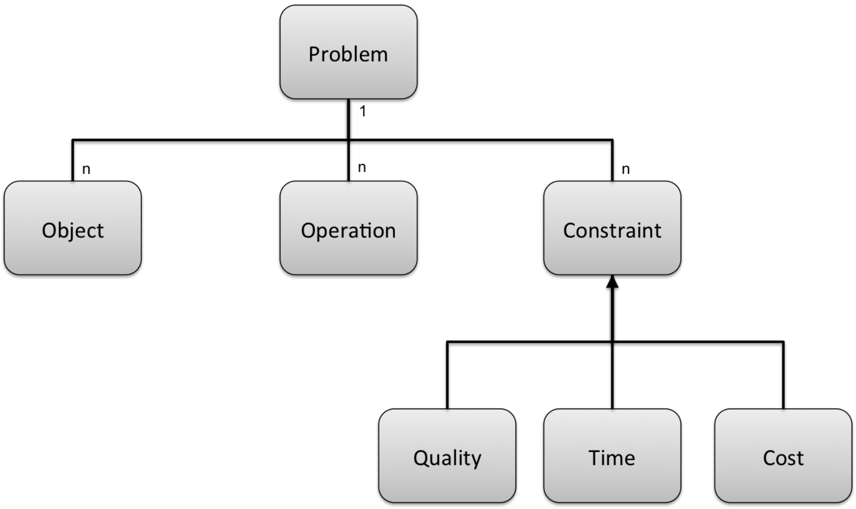


Figure 3.2: *Model describing a problem that needs to be crowdsourced*

tion Finding category is that in that case the content is already present on the web.

- Surveys: consists in asking the crowd to fill forms.
- Content Access: consists into asking the crowd to access to some content.

A given Problem can be solved in many different way, for instance an Interpretation and Analysis problem like building a rank can be addressed by asking the crowd to make pairwise comparison or to give a score to each objects.

In the following Section I propose a model for the Task to be performed by the crowd that can be mapped to any Problem and Crowd.

3.4 Task Model

Figure 3.3 shows the abstract model, built after a careful analysis of the systems for human task executions and of many applications and case studies.

A *Task* receives as input a list of *Objects* (e.g., photos, texts, but also arbitrarily complex elements) whose schema is defined by the *Object Type*. Then a task ask the crowd to perform one or more *Operation* upon the objects. The possible operations are:

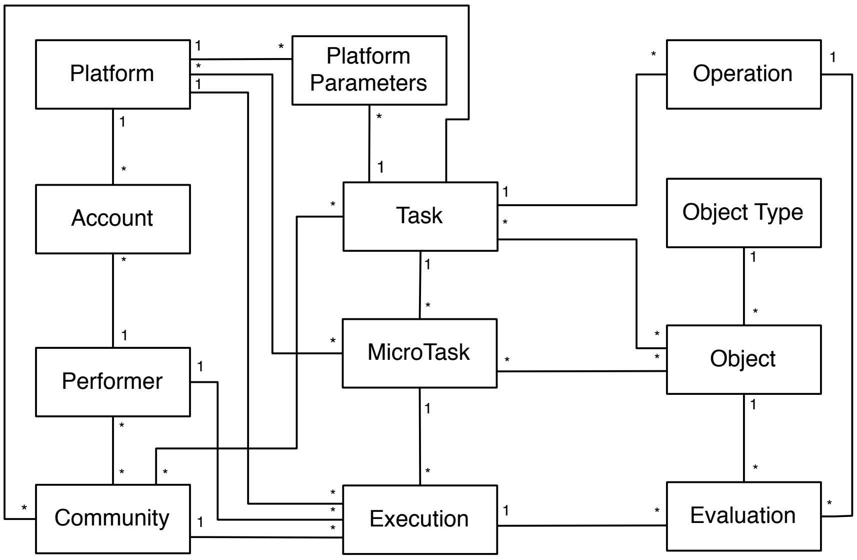


Figure 3.3: *Metamodel of task properties.*

- Classify: The performer assigns each object to one or more classes
- Comment: The performer write a comment about an object.
- Like: The performer adds a like to some objects.
- Score: The performer assigns a score (in the 1..*n* interval) to some objects.
- Tag: The performer annotates some objects with tags
- Group: The performer clusters the objects into (at most *n*) distinct groups
- Insert: The performer inserts up to *n* objects.
- Delete: The performer deletes up to *n* objects in the list.
- Modify: The performer changes the values of attributes of some objects in the list.
- Order: The performer reorders the objects in the input list.

A task can be executed on different *Platforms*, that can be both crowdsourcing marketplaces and social networks, according to some *Platform*

Parameters. The *Execution* element represent the fact that a *Performer*, belonging to a particular *Community*, produces an *Evaluation* on a specific *Object*. The structure of an *Evaluation* element depends on the *Operation*, in particular the schema are the following:

- Classify: the list of selected categories.
- Comment: the text written by the performer.
- Like: an empty evaluation.
- Score: the number representing the score.
- Tag: the list of tags provided by the performer.
- Group: the group identifier.
- Insert: an annotation that show that the object was created as a response.
- Delete: an annotation that show that the object was marker for deletion.
- Modify: the fields new values.
- Order: the position of the object in the list.

Moreover the *Execution* element contains statistics regarding the activity of the performer, for instance the how much time he dedicated to performing the operation. A performer does not execute directly the entire *Task*, but operate on a subset of objects called *MicroTask*. Finally a *Performer* can have *Accounts* on different *Platforms* and belongs to different *Communities* (these concepts will be further explained in Chapter 5).

The model can be extended either by adding custom operation types or by adding custom parameters and output variables to the given types; however, the operation types of the model are supported by automatic model transformations, discussed next, for generating a crowdsourcing application, while custom elements require manual refinement of either the transformations or the constructed models.

In the following section I describe the design process that is used for creating a crowdsourcing application.

3.5 Crowdsourcing design process

The design of each task in a crowd-based application consists of a progression of six phases reported in Figure 3.4, namely ***1*) Operation design**, i.e. deciding which operation the crowd is going to perform on the objects; ***2*) Object design**, i.e. defining the **Object Type** and preparing the actual set of objects to be evaluated, which may be extracted from different data sources; ***3*) Performer design**, i.e. selecting the performers that will be asked to perform the task; ***4*) Workplan design**, i.e. defining how each task is split into micro-tasks, and how micro-tasks are assigned to objects; ***5*) Platform selection**, i.e. defining the *invitation platforms*, where performers are recruited, and the *execution platforms*, where performers execute tasks; many different platforms may be involved in either roles; ***6*) UI design**, i.e. defining the front end aspects of the task execution. We next define each of these phases.

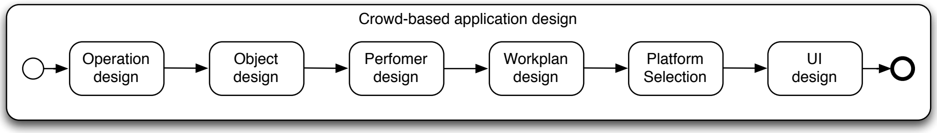


Figure 3.4: *Development process of crowdsourcing tasks.*

3.5.1 Operation Design

Operation design consists of deciding the operations of the task, it is conducted by instantiating part of the the meta-model in Fig. 3.3, in particular the *Task* (containing some additional informations such as a name and a description) and the *Operation*, by selecting the type and the following configuration.

3.5.2 Object Design

Object design consists of defining the dataset which is subject to the analysis. In particular, object design entails: *1*) the definition of the schema of the objects to be analysed; *2*) the extraction or collection of the instances (e.g., from web sources, proprietary databases, plain text, or multimedia repositories); and *3*) the cleaning of the extracted objects, so as to make them conforming to the defined schema. By the end of this phase, the *Object Type* and all the *Objects* elements will be instantiated. Notice that in case of a *Insert* operation, the initial list of objects may be empty.

3.5.3 Performer Selection

In crowdsourcing, strategies for task assignment can be roughly classified into two main categories: *push* and *pull* assignment. With *pull* assignments, tasks are published on an open board, and performers select them. This strategy is used by most crowdsourcing platforms, and is well-suited for simple repetitive tasks. With *push* assignments, tasks are routed either to individuals or to communities based upon trust, knowledge, or expertise. Performers are either pre-selected or dynamically assigned to executions, depending on the task content.

Social platforms, such as Facebook, Twitter and LinkedIn, provide their members with several hundreds of known contacts, with variable expertise about various topics, and with varying availability and responsiveness.

3.5.4 Workplan Design

Workplan design consists of creating micro-tasks for each task and of mapping each micro-task to specific performers (if previously selected) and objects. Task design includes task splitting, that should be performed on different task types according to different algorithms. Several articles are dedicated to task splitting algorithms for specific operations, e.g., [74]; I don't discuss task splitting further (although I support it in my framework and use it in experiments).

Task planning is performed according to *planning directives*, that indicates:

- Cardinality: the number of objects associated to each micro-task.
- Replication: the number of copies of each object that should be assigned to micro-tasks.
- Initiation: the number of micro-tasks which should be created when the application starts (more micro-tasks can be dynamically created during the execution).
- Grouping: how the objects should be grouped in micro-task.

This phase generates a number of *MicroTasks*; each micro-task instance is associated to one task, a set of performers, and one or more objects. Depending on the underlying execution platform, tasks can be *pushed* to specific performers or performers can *pull* the tasks that they like; in the push model micro-task mapping to performers is static, while in the pull model it is dynamically assigned.

3.5.5 Platform Selection

Plat.Type	Examples	Inv.	Exec.
Social Network	Facebook, Twitter, G+, LinkedIn	YES	Limited
Question & Answer	Quora, Doodle	Limited	YES
Crowd-sourcing	AMT, CrowdFlower	YES	YES
Proprietary UI	Custom developed application	Limited	YES
Email or messaging	Mailing lists, personal email, phone messages	YES	Limited

Figure 3.5: *Taxonomy of platforms and use in invitation and execution.*

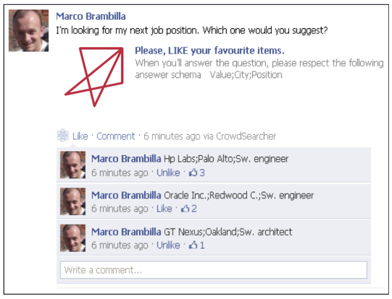

Marco Brambilla
I'm looking for my next job position. Which one would you suggest?
Please, LIKE your favourite items.
When you'll answer the question, please respect the following ansewer schema Value;City;Position
Like · Comment · 6 minutes ago via CrowdSearcher
Marco Brambilla Hp Labs;Palo Alto;Sw. engineer
6 minutes ago · Unlike · 3
Marco Brambilla Oracle Inc.;Redwood C.;Sw. engineer
6 minutes ago · Like · 2
Marco Brambilla GT Nexus;Oakland;Sw. architect
6 minutes ago · Unlike · 1
Write a comment...

Figure 3.6: *Native Facebook* "like" *operation.*

At this stage, after a platform-independent design, deployment platforms must be chosen. A variety of systems are offered, and it is crucial to understand how they can be crafted to reflect the application needs. I distinguish *invitation* from *execution*, the former process is concerned with inviting people to perform tasks, the latter is concerned with executing tasks. In general it is possible to use different systems for invitation and execution. Figure 3.5 shows how the different kinds of platform support them.

Social Networks provide powerful interfaces for crowd selection and invitation, and limited support for execution. Note that deployment on social network can be of two kinds [16]:

- *Native implementations* use the features of a specific social network for task execution.

- *Embedded implementations* use the execution of user-defined code from within the social network.

For instance, Figure 3.6 shows how Facebook can be natively used for implementing *like* operations: data objects are posted on a wall and users simply click the Facebook *Like* button. Certain operations, such as liking and tagging, are best supported by native interfaces.

Question-Answering Systems normally cannot support invitation (or provide invitation mechanisms that require to provide the list of invitees from a personal contact list), while they can support execution, although some of them only support free text responses.

Crowdsourcing Systems support both invitation and execution, but they do it within the context of a given platform, which acts as a market place, where invitations include the monetary reward for each task, and after execution each performer is credited. Invitations from other platforms are typically not allowed.

Email and other messaging systems support the invitation phase, that can be routed to specific groups, mailing lists, or enumerated list of targets. Execution is typically delegated to other platforms.

The most typical form of deployment consists in delivering a UI crafted around the features of the specific objects to be shown to performers, as discussed in the next section.

3.5.6 UI Design

UI design plays an important role in crowdsourcing, as it produces the user interfaces that permit the actual execution of tasks by performers. UIs can be designed in three main ways: *1*) By exploiting default, basic UIs made available by crowdsourcing platforms (e.g., AMT); *2*) By exploiting the conceptual task model for generating a simple custom UI; *3*) By manually implementing an ad-hoc user interface most suited to the task.

Finally, no UI is needed if tasks are natively performed on social platforms. For instance, this is the case when a task is performed by directly exploiting the *like* mechanism in Facebook.

3.6 An Explorative Method for Crowdsoucing Task design

In this section I describe an alternative task design process based on an empirical exploration of the design space, where small prototypes of the task are executed in order to discover which one will produce the best results.

This approach refers to a fragment (shown in Figure 3.7) of the model described in the previous sections, which describes how each elementary execution of a crowdsourcing step, called *Execution*, is referred to an underlying operation (e.g. classifying, tagging, labeling, liking, commenting) called *Task*, to a specific *Platform* (that can be either a crowdsourcing marketplace or a social network), to a specific *Object* of a given collection, and to a specific *Performer* who executes.

These concepts, in turn, are characterized by a set of properties, whose ranges of values define the design space. Typically, they should be assigned by the application designer in order to configure the crowdsourcing tasks, either by interacting with a design tool, or by using scripting languages

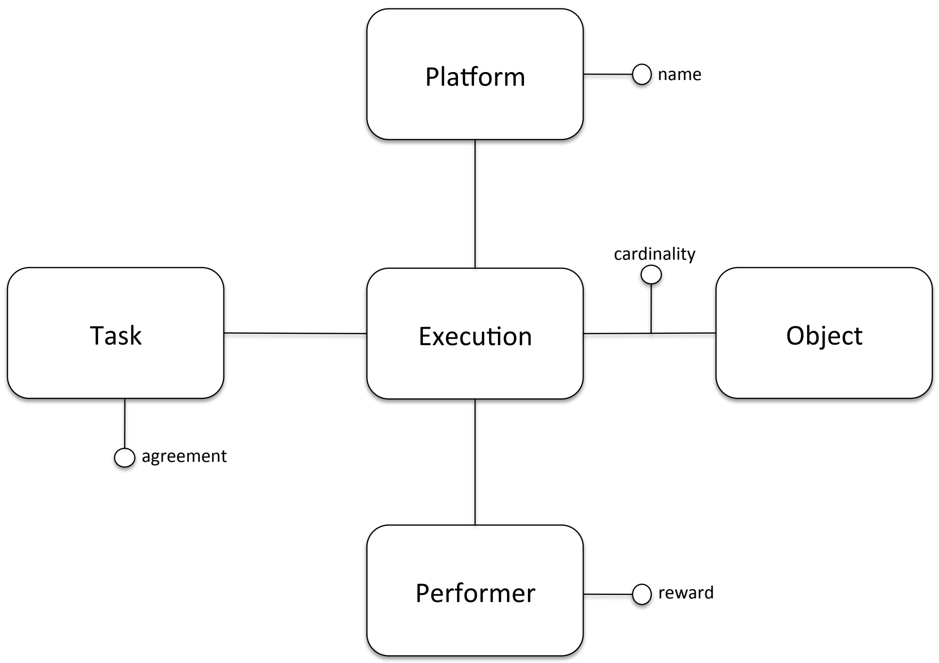


Figure 3.7: *Concept model and parameters used by our approach*

(which in turn invokes an API where they appear as parameters), or for directly configuring the application. For instance, figure 3.7 illustrates a setting where we define four properties; the method is agnostic to the specific choices of properties but assumes that they can be referred to the concept model and, of course, that they can be used for configuring the execution task. They are:

- **Platform**: where the task will be executed. This is a very important dimension because each platform targets different crowds which have different skills.
- **Cardinality**, i.e. the number of object shown to the performer: this parameter controls the amount of work that a performer has to face each time. It influences the cost and time required by the task.
- **Reward**, i.e. the cost of a HIT on Amazon Mechanical Turk.
- **Agreement**: i.e., with a majority based decision for each objects, it indicates the amount of agreement needed in order to consider an object as evaluated. A high level of agreement should correspond a

better quality of the results while negatively impacting on the time and cost.

This list can be extended in order to satisfy specific user needs, for instance adding a spam detection strategy, whose modeling would lead to adding a **Spam** flag on the performer, set to 0 or 1 to indicate its inclusion or exclusion.

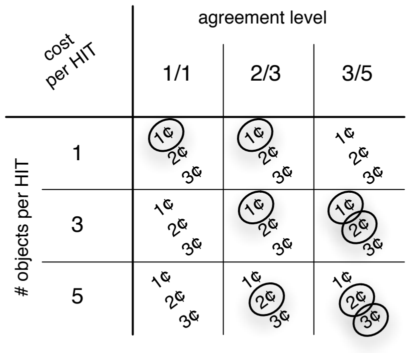
Each candidate execution is thus represented by a vector $S = \{s_1, s_2, \ldots, s_n\}$ in an $n-$dimensional space, where n is the number of considered parameters; for instance, an execution on Amazon Mechanical Turk showing 3 objects per HIT, requiring a 2 workers over 3 to agree on the evaluation and paying each worker 0.01$ is represented as:

$$S = ["AMT", 3, 2/3, 0.01] \tag{3.1}$$

Once the design space is well defined, the designer should then choose some of the possible strategies (represented as a collection of vectors.) It is not possible to consider all possible combinations due to the cost and the required time for conducting all the small-scale experiments. It is important to choose few strategies by including interesting points in the solution space and by using as criteria parameter diversification, at the same time by avoiding to include any two solutions when one of them *dominates* the other. This notion is not easily formalizable, but it takes into account the correlation between parameters. For instance, it makes little sense to include two solutions such as one has a higher cost and a lower object cardinality than another one (i.e., a simpler task which is better paid).

The execution of strategies, both in the small and in the large, can be evaluated by using a set of quality measures that are computed at the end of the process by inspecting how each object has been managed by the crowd (e.g., its classification, tagging, liking and commenting). I use the following quality measures:

- **Cohen's kappa coefficient**, a statistical measure of inter-annotator agreement for categorical annotation tasks. When several performers evaluate the same objects, kappa measures the agreement among them.
- **Precision of responses**, that can be computed only when the ground truth is available; it corresponds to the percent of correct responses over the total and can be aggregated at the level of object, performer, platform, or whole task.
- **Execution time**, the elapsed time needed to complete the whole task.



Configurations	Images per HIT	Required agreement	Reward per HIT
1	1	-	$0.01
2	1	2 over 3	$0.01
3	3	2 over 3	$0.01
4	3	3 over 5	$0.01
5	3	3 over 5	$0.02
6	5	2 over 3	$0.02
7	5	3 over 5	$0.02
8	5	3 over 5	$0.03

(a) (b)

Figure 3.8: *(a) Multi-dimensional matrix of parameters which are compared in the running example, and selection of representative combinations (encircled); and (b) Quality measures of the selected combinations after their execution over a subset of the objects.*

- **Monetary cost**, the total amount of money spent for rewarding the crowd in order to complete the whole task.

This is only a small set of possible performance measures, and can be extended with more complex (as the ones shown in [44]) or application specific metrics.

Finally, this approach requires the splitting of the dataset of the objects into two subsets *small* and *large*, with $|small| << |large|$, such that the selection of S is not biased.
Then, all the strategies $\{S_1, S_2, \ldots, S_m\}$ are run on the *small* set (*in the small* phase) and the quality measures are collected; by analysing them, the strategy S_{best} which is associated with the *best quality measures* is selected.

Eventually S_{best} is run on the remaining objects of the *large* dataset and its results are composed with the ones obtained with the *small* set.

3.6.1 Experiment

I designed an image labeling crowdsourcing task in which I ask the crowd to classify pictures related to actors, telling if it represents the actor himself in a portrait, if it is a scene taken from a movie, or if it is not relevant (exclusive options); Amazon Mechanical Turk (AMT) has been used as execution platform.
Using this approach, I identified the following design dimensions: number of images shown in each user task, agreement level for each picture clas-

sification, and cost of each AMT hit. Then I selected 8 different strategies (as shown in Figure 3.8) and I ran them on both the small and large dataset.

The experiment had the purpose of assessing the two main assumptions of this method:

A1 The outcome of the experiment *in the small* is correlated with the result of the experiment *in the large*;

A2 The cost of performing all experiments *in the small* followed by one experiment *in the large* is affordable and the extra-effort is well compensated by the possibility of choosing the experiment with the best quality measures in the small.

I determined an experiment of limited size but sufficient to perform such an assessment. The dataset consists of 900 images related to 90 actors, retrieved from Google Images; 90 images were selected for the phase in the small (i.e. 10 images for 9 actors, including both men and women), so that the comparison of small vs large involves an order of magnitude, which is enough to illustrate the difference between small and large cases. This setting hints to the quality of the method also when the difference between small and large size reaches two or three orders of magnitudes, as in a typical big data scenario.

Then the experiment was run eight times, both in the small and in the large, so as to assess the similarity of small and large size experiments; we paid a total of $227 for the sixteen experiments (of course, introducing two or more orders of magnitude of difference between the small and large cases would require a corresponding, proportional increase of the total cost.)

Table 3.1 summarizes the results of the experiment, by reporting the four quality measures (kappa, precision, cost and duration).

Regarding the assumption A1, I calculated the Pearson correlation coefficient, configuration-wise, between the experiments in the small and in the large. As one can see, correlation is almost one for the cost, that can be obtained just by considering the scale factor between *small* and *large*; but correlation is quite good also for duration, performer agreement and precision. Note that durations are longer for the *small* experiments than for the *long* ones. This reflects a known behavior of the crowd, which tends to select tasks with higher number of executions to perform (also due to the bias introduced by crowd platforms, which show the biggest tasks first).

Then, for verifying the assumption A2, I compared the strategy by looking for a trade-off between *precision* and *cost*. In particular, based upon the small-scale experiment, I selected Strategy 6, which appears to have

Table 3.1: *Quality measures and Pearson correlation of experiments* in the small *and* in the large.

	Agreement kappa		**Precision**		**Cost ($)**		**Duration (s)**	
Config.	***small***	***large***	***small***	***large***	***small***	***large***	***small***	***large***
1	N/A	N/A	0.733	0.799	1.35	13.68	14885	8832
2	0.692	0.607	0.778	0.855	4.05	43.97	11788	20346
3	0.596	0.612	0.811	0.838	1.40	14.15	52219	30032
4	0.579	0.578	0.822	0.857	2.25	23.10	114186	63963
5	0.442	0.569	0.856	0.858	4.77	46.35	120983	53162
6	0.499	0.540	0.811	0.864	1.92	16.86	110535	65178
7	0.580	0.606	0.800	0.871	2.70	28.05	121945	67676
8	0.533	0.555	0.833	0.838	4.05	41.67	78086	23745
Correlation	**0.707**		**0.619**		**0.999**		**0.915**	

enough precision (0.864) associated with a low cost (1.92), yielding a good price/performance ratio.The choice of Strategy 6 completes the decision making.

The designer's choice is anyway driven by cost-benefit analysis, that however is performed *in the small*, e.g. the designer will be able to decide if a difference in precision from .811 of case 3 to .856 of case 5 is justified by an increase in costs from 1.40 to 4.77.

Note that \$22.49 were spent for computing all the strategies in the small and \$16.86 for executing the strategy number 6, for a total cost of \$39.35; these two numbers are comparable, but the difference between the cost of experiments in the small and in the large increases a lot with big input data. When the task is very large, an incremental tuning is also possible, e.g. using datasets of increasing sizes for computing the quality measures of a restricted number of candidates. The case *in the large* of Table 3.1 can be considered an intermediate-size experiment if one has to process a dataset of millions of photos; in such case, the eight cases in the large would result from a selection starting from a larger number of experiments in the small.

One could note that case 7 is associated with a slightly higher cost of 2.70 compared to case 6 (that was selected by considering quality measures in the small), but it also exhibits a better precision in the large of 0.871 compared to case 6; such better precision is not predicted by the experiment in the small and comes as a surprise. Indeed the method incurs some unexpected differences between tests in the small and in the large due to the intrinsic statistical variability of our study; greater sizes in both small and large cases would yield to less variability.

3.7 Conclusion

In this chapter I modeled the main aspects of a Crowdsourcing application. I defined the Crowd model and the Problem model to be used as guide for building the Task model. I defined a six steps design process for defining the Task model. Finally I proposed an alternative design process based on empirical method in which the "best" instance of the Task model is selected from various candidates through the execution of small experiment.

The concepts described in this chapter are only the starting point of my thesis, in the following chapters I will extend the model in order to add feature of control (Chapter 4), interoperability (Chapter 5) and complex flows of tasks (Chapter 6)

CHAPTER 4

Controlling the Crowd

The work described in this chapter was published in [17].

4.1 Introduction

Most crowdsourcing systems are not flexible, as they do not support a high level, fine-tuned control upon posting and retracting tasks. The AMT platform offers an API for controlling the posting and evolution of tasks, but the implementation of applications must be done through imperative languages, either with low-level programming or through frameworks like TurkIt [53]. Since crowdsourcing can be performed upon social networking platforms too, one can exploit social networks APIs for programming applications which use them. Some academic works provide control capabilities for crowdsourcing, but with limited and specific control rules (e.g., the DeCo [64] system provides support for closing tasks based on temporal constraints). In summary, in spite of the great importance of crowd control, at the current state-of-the-art designing and deploying crowdsourcing applications with sophisticated controls is very difficult. All the existing platforms and approaches lack methods for systematically designing complex control strategies based on the state of tasks, results and performers.

In this chapter I describe a conceptual framework and a reactive execution environment for modelling and controlling crowdsourcing computations; reactive control is obtained through rules which are formally defined according to a rule specification language and whose properties (e.g., termination) can be easily proved in the context of a well-organized computational framework. As highlighted by [63], several programmatic methods for human computation have been proposed so far [53] [49] [3] [57] [60], but they do not support yet the complexity required by real-world, enterprise–scale applications. Due to its flexibility and extensibility, our approach covers the expressive power in reactive control which is exhibited by any of the cited systems.

Starting from task types, I then define the data structures which are needed for controlling the planning, execution, and reactive control of crowdsourcing applications. Control encompasses the evaluation of arbitrary conditions on result objects (e.g., on their level of confidence and of agreement), on performers (e.g., on the number of performed tasks and their correctness, leading to the classification of performers as experts or spammers) and on tasks (e.g., on their number and duration). My framework provides a reactive style for specifying these conditions and for defining the actions that must be correspondingly triggered, making decisions about the production of results, the classification of performers the early termination and re-planning of tasks, the dynamic definition of micro-tasks, and so on.

Besides the conceptual definitions and the control rule language, my proposal comprises a platform which acts either as a stand-alone system or as an interoperability framework on top of social networks or crowdsourcing systems, which are accessed through their APIs – and therefore are subject to APIs' limitations; the platform includes an engine which executes rules and crowdsourcing tasks. I present a set of experiments with different rule sets, demonstrating how simple changes to the rules can significantly affect the time, effort and quality of tasks.

The chapter is organised as follows: Section 4.2 defines the model designed for controlling the crowd; Section 4.3 describes in details our reactive language for controlling crowdsourcing tasks; Section 4.4 reports on my implementation and experiments; and Section 4.5 concludes.

Thorough this chapter I'll use as running example a simplified version of the experiments described in Section 4.4; I'll design the control for a task in which the crowd has to classify American politician as Democratic or Republican.

4.2 Crowd Control Model

For effectively monitoring the execution, it is most convenient to track the performer's response for each distinct object which is present in the micro-task. Thus, starting from the model described in Chapter 3, a final model transformation creates the **control mart**, shown in Fig. 4.1, where each *microTObjExecution* instance is connected exactly to one task, one performer and one object. The control mart is analogous to data marts used for data warehousing [43], as its central entity represents the facts, surrounded by three dimensions.

A minimal relational representation of the control mart, useful for describing reactive control design in the next section, is described in listing 4.2.1, and it includes the fact table `Execution` and the dimension tables `Politician, Performer` and `Task`. Each execution requires an `Eid`, as the same performer could be assigned the same politician in different micro-tasks; `Status` attributes trace the application evolution (e.g., a `Performer` can be `'Active'` or `'Spammer'`, a `Task` and an `Execution` can be in the status: `'Planned'`, `'Started'`, `'Completed'` or `'Invalid'`). Additional information of interest is present in the actual data marts, e.g., the starting and ending time of tasks and execution, the number of completed tasks for each performer, and so on.

4.2.1 Fact Table and Dimension Tables

```
Politician (Oid,Party)
Performer  (Pid,Status)
Task       (Tid,Status)
Execution  (Eid,Oid,Pid,Tid,Status,Party)
```

4.2.2 Aggregate Tables

```
Object_CTRL    (Oid,Eval,Dem,Rep,Answer)
Performer_CTRL (Pid,Eval,Right,Wrong)
Task_CTRL      (Tid,CompExec,CompObj)
```

In addition to the control mart, *aggregate tables* are derived that contain one tuple for each politician, performer and task, and are automatically maintained at each micro-task completion by computing aggregates.

Aggregate tables can be derived in an analogous way also for the other operations listed in Section 3.4: for instance, the `Object_ CTRL` table of a *like* operation could feature a `Preferences` attribute to count the preferences obtained by the object.

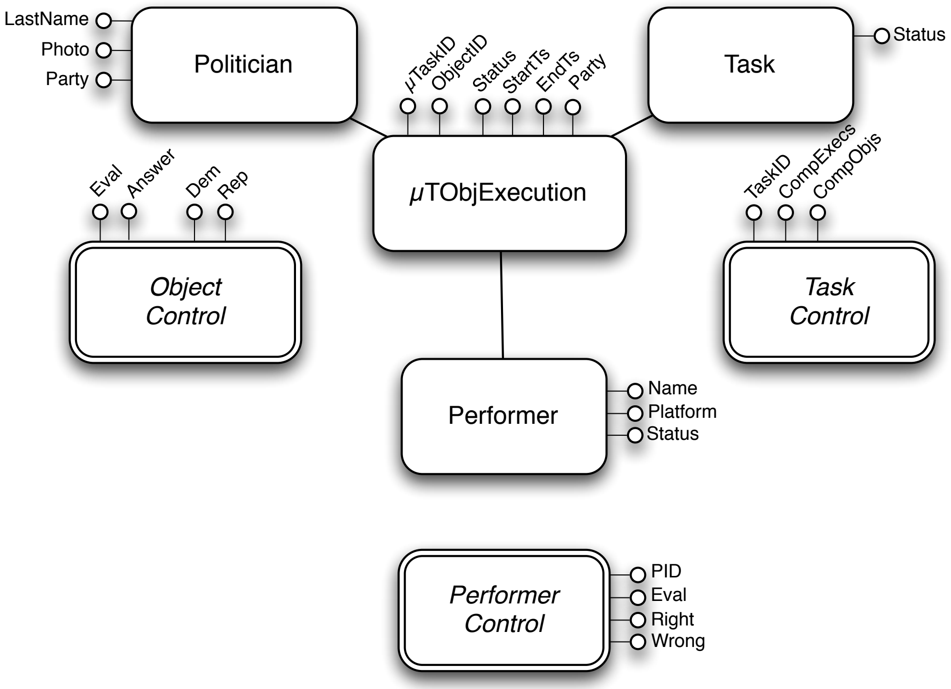


Figure 4.1: *Control mart.*

At workplan enacting, the system creates the instances of the control mart and aggregate tables. Some information might be still undefined (e.g., with a pull model, the identity of performers becomes known during execution), but otherwise we assume that each object, performer and task is associated with suitable entries in the control mart and aggregates tables. The next section is the core of this chapter, dedicated to object, performer, and task control.

4.3 Reactive Control Design

Control design consists of three activities:

- *Object control* is concerned with deciding when and how responses should be generated for each object.

- *Performer control* is concerned with deciding how performers should be dynamically selected or rejected, on the basis of their performance.

- *Task control* is concerned with completing a task or re-planning task execution.

The control of objects, performers, and tasks is performed by active rules, expressed according to the *event-condition-action* (ECA) paradigm. Each rule is triggered by **events** (`e`) generated upon changes in the control mart or periodically; the rule's **condition** (`c`) is a predicate that must be satisfied on order for the action to be executed; the rule's **actions** (`a`) change the content of the control mart and aggregate tables. This approach has the following advantages:

- *Automation*: most active rules are system-generated.
- *Flexibility*: encoding variants of simple controls require to change specific rules while preserving the rest of the rule set.
- *Power*: rules can be programmed to support arbitrarily complex controls.

4.3.1 Rule Language

The rule language has been inspired by the long-standing tradition of active databases [76]; its full syntax is reported in Appendix 4.A. Its peculiar syntactic features are the following:

- Rules can be triggered by classical data updates and by periodic `TIMER` events.
- Rules are at row-level granularity. Variables `NEW` and `OLD` denote the *before* and *after state* of each row.
- Special selector formulae are used to express subqueries synthetically; thus, `TABLE[<predicate>].ATTRIBUTE` extracts the same values as `SELECT ATTRIBUTE FROM TABLE WHERE <predicate>`.
- Special *functions* may perform operations on the workplan model (e.g., planning of new micro-tasks).

Two examples of simple active rules for maintaining the counters of 'Rep' and 'Dem' answers for a given Politician are reported in rules 1 and 2; they are triggered by the completion of a micro-task, which in turn consists of an update to the `Answer` attribute of one or more rows in `Execution`.

Rule 1 RepAnswer Rule.

```
e: UPDATE FOR Execution[Answer]
c: NEW.Answer ==  'Rep'
a: SET Object_CTRL[oid == NEW.oid].Rep += 1
```

Rule 2 DemAnswer Rule.

```
e: UPDATE FOR Execution[Answer]
c: NEW.Answer ==  'Dem'
a: SET Object_CTRL[oid == NEW.oid].Dem += 1
```

4.3.2 Active Rule Programming

It is known that active rule programming is rather subtle and unstable: the behaviour of a set of rules may change dramatically as a consequence of small changes in the rules [76]. To overcome this problem, I observe a *best practice* in writing rules. Functionally, I divide rules in three classes:

- *Control rules* for modifying the control tables;
- *Result rules* for modifying the result tables (Politician, Performer, Task);
- *Execution rules* for modifying the execution table - either directly or through replanning of crowdsearching.

Consider the *Precedence Graph* $PG =< N, E >$, where the nodes N are tables; an arc $< N_1, N_2 >$ is drawn when a rule R is triggered by an operation on N_1 and performs an action on N_2. Then, I impose that control and result rules in our system can only generate edges in the PG shown in Fig. 4.2. Intuitively, this best practice corresponds to propagating rule execution top-down (from execution to control to result tables) and left-to-right (from object to performer to task).

I further assume that only control rules can have cycles in PG, and in such case I assume their triggering graph to be acyclic.[1] Then the following result holds.

Theorem. Any execution of control and result rules terminates.

Proof. The acyclicity of the triggering graph of the rule set, a sufficient condition for termination [76], descends from the acyclicity of the PG graph shown in Fig. 4.2 (excluding execution rules) and from the acyclicity of ring rules.

[1]To guarantee this assumption it is sufficient that aggregate computations is performed progressively, from fine to coarse-grain aggregations, as shown by the example rules in Appendix 4.B.

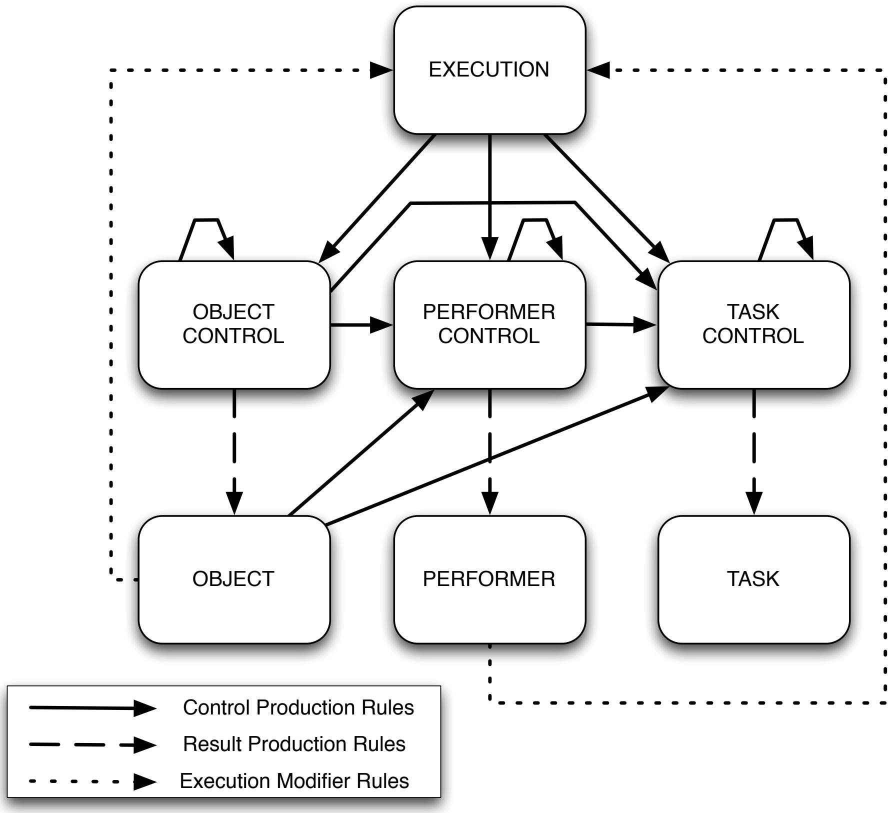


Figure 4.2: *Precedence Graph for rules.*

Thus, by adopting the best practice for active rule programming described above, I need to worry about rule termination only when I add execution rules, that typically create cycles in the triggering graph of the rule set. These rules have to be carefully considered, as I will show in Section 4.3.

4.3.3 Control Rules

Control rules maintain the control tables; they are triggered by updates of the execution table (e.g., when a micro-task execution is completed), or of the result tables (e.g., when an object is *closed*, see later) of the control tables themselves – e.g., the rules 1 and 2, which update the aggregate counts `Rep` and `Dem` on the basis of the `Answer` in the `Object_ CTRL` table. Control rules are automatically generated, due to the fact that operations are typed and to the model-driven application generation process described in Chapter 3. In addition to rules 1 and 2, several control rules are listed in Appendix 4.B. I next describe the interesting case of rules for spammer control.

Spammers are performers whose answers deviate significantly from correct answers; when performers are rewarded by money, spammers typically give random answers in order to maximise their pay, and providing unreliable answers; the definition of spammer is orthogonal to the reward and is defined just on statistical basis. Spammer detection requires two parts: first computing wrong answers (using a control production rule), and then deciding which performers are spammers based on the amount of wrong answers they provide (using a result production rule). Next I show three different control production rules corresponding to different scenarios; all rules can be automatically produced – the flexibility of the approach is demonstrated by the fact that a rule change is sufficient to change the control policy.

Performers evaluation against golden truth

A typical strategy [70] in crowdsourcing is to set a few *golden answers* known a priori (e.g., predefined by experts) and then check the correctness of performers against them, while tasks are executed. This is possible in this approach by adding the `Gold` property to the object schema; `Gold` stores the golden answer when available, and a `NULL` value otherwise. Then, the rule for managing the aggregate control information of performers is represented in Rule 3.

Rule 3 GoldenTruthRule.

```
e: UPDATE FOR Execution[Answer]
c: Politician[oid=NEW.oid].Gold <> NULL
a: IF(NEW.Answer == Politician[oid==NEW.oid].Gold)
     THEN SET Performer_CTRL[pid==NEW.PID].Right += 1
     ELSE SET Performer_CTRL[pid==NEW.PID].Wrong += 1
```

The rule is triggered by any answer on `Execution` and simply checks if the answer is about a politician with available golden answer; if so, it updates the counters of `Right` and `Wrong` answers of `Performer_ CTRL` depending on the correctness of the given answer against the golden value.

Performers evaluation on object completion

The second possibility for maintaining the performer counters is to wait for an object to be completed (Rule 4) – i.e., for a final evaluation to be produced.

Rule 4 ObjectResultRule.

```
e: UPDATE FOR Politician[Party]
c: NEW.Party <> NULL
a: FOREACH e IN EXECUTION[Oid==NEW.Oid]
      IF (e.Party == NEW.Party)
        THEN SET Performer_CTRL[Pid==e.Pid].Right += 1
        ELSE SET Performer_CTRL[Pid==e.Pid].Wrong += 1
```

The rule is triggered by the completion of an object, which is assigned a non-null `Party` value as effect of an agreed response. The rule then considers all the past executions of that object and compares the answers of the performers with the answer that has been written in the `Party` attribute, and updates the `Right` and `Wrong` counters of `Performer_ CTRL`.

Performers evaluation on each execution

The third possibility is to maintain the performer's counters at each execution; this anticipates the definition of right and wrong answers even if a final result is not available for the objects. In this case, the `Pid` attribute of the performer who caused the last change to an object is added to the `Object_ CTRL` table (the content of `Pid` is copied from `Execution` by a suitable control rule).

In Rule 5, at every update of the current `Answer` in the `Object_ CTRL` table, the specific performer who provided the last `Answer` in the

Rule 5 ExcutionResultRule.

```
e: UPDATE FOR Object_CTRL[Answer]
c: Answer<> 'Undefined'
a: SET Performer_CTRL[Pid==NEW.Pid].Right=0
   SET Performer_CTRL[Pid==NEW.Pid].Wrong=0
   FOREACH e IN Execution [Pid==NEW.Pid]
     FOREACH o IN Object_CTRL[Oid==e.Oid]
       IF [e.Answer == o.CurrentAnswer]
        THEN SET Performer_CTRL[Pid==NEW.Pid].Right += 1
        ELSE SET Performer_CTRL[Pid==NEW.Pid].Wrong += 1
```

execution table is considered, and all the past answers of that performer are compared with the corresponding current answers; given that current answers change their value during a crowdsourcing session, incremental maintenance is impossible, and the counters of the affected performer have to be set to zero and recomputed.

4.3.4 Result Rules

Result rules are triggered by changes in control tables and produce result tables; they express the result control logic, that can be specified through high level directives and be translated to rules. I consider how to decide that an object is closed or that a performer is a spammer.

Closing Objects

Objects are closed when they are associated with *enough* evaluations to provide a conclusive response, i.e. a majority of equal answers. The smallest possible majority calls for two equal answers, and is recognised by Rule 6.

Rule 6 MajorityResultRule.

```
e: UPDATE FOR Object_CTRL
c: (NEW.Rep== 2) or (NEW.Dem == 2)
a: SET Politician[oid==NEW.oid].Party = NEW.Answer,
   SET Task_CTRL[tid==NEW.tid].CompObj += 1
```

This rule is triggered by any change of the object control table, and simply checks that one of the two attributes `Rep` or `Dem` is equal to 2; then it sets the politician's party equal to the current answer and increases the number of completed objects in `Task_ CTRL`.

Of course, different majority conditions are possible, which can be arbitrarily complex and depend also on the number of evaluations, e.g.,

```
C1: (Eval>5) and ((Rep>0.5*Dem) or (Dem>0.5*Rep))
C2: (Eval>10) and ((Rep>0.8*Dem) or (Dem>0.8*Rep))
C3:  Eval>15
```

The above cases denote three distinct rule conditions; they can either be embedded into three different rules or their disjunction could be embedded into a single rule. The effect is to close the object as soon as one of the three conditions is true. With enough micro-task completions, the condition `Eval>15` becomes eventually true.

Identification of Spammers

Performers are identified as spammers when they are associated with *enough* wrong answers, which have been collected according to anyone of the methods discussed in Section 4.3.3. A simple rule for identifying spammers is:

Rule 7 SpammerIdentificationRule.

```
e: UPDATE FOR Performer_CTRL
c: (NEW.Eval > 10) and (NEW.Wrong > New.Right)
a: SET Performer[Pid==NEW.Pid].Status = 'Spammer'
```

This rule is triggered by any change of the performer control table, and simply checks that after 10 evaluations the number of wrong answers exceeds the number of right answers; then it sets the performer's status to `'Spammer'`.

Of course, different spammer identification conditions are possible, e.g., condition C1 identifies as spammer whoever performs 4 errors, condition C2 selects as spammer anyone who has given more than 20% of wrong answers, condition C3 uses two thresholds.

```
C1: Wrong == 4
C2: Wrong > 0.2*Eval
C3: ((Eval>10) and (Wrong>3)) or (Wrong>Right)
```

4.3.5 Execution Rules

Execution rules respond to the need of altering the execution plan; their action either changes the current micro-tasks or calls for task re-planning, which eventually produces new micro-tasks. These rules are triggered by changes in the control or result tables, and are perhaps the most powerful rules. They must be analysed because they may introduce danger of nontermination of the computation.

Remove Spammer's Micro-Task Executions

Spammer detection results in excluding performers from future assignments. In addition, I may want to propagate the effects of spamming detection upon micro-tasks. Rule 8 selects all the executions of the performer which has been recognised as `'Spammer'`, and checks whether the corresponding objects have been already completed; if not, it logically undoes the spammer's micro-tasks, by first subtracting 1 from `Eval` and either the `Rep` or `Dem` counters of the object control table, and then by deleting the execution tuple. Note that the subsequent propagation to `Answer` in `Object_ CTRL` is performed by a rule in Appendix 4.B, with no change.

Rule 8 RemoveMicroTask.

```
e: UPDATE FOR Performer[Status]
c: (OLD.Status != 'Spammer') and (NEW.Status=='Spammer')
a: FOREACH e IN Execution[Pid==NEW.Pid]
     SET e.Status= 'Invalid',
     FOREACH o IN Object_CTRL[o==e.Oid]
       IF (Politician[Oid==o.Oid].Party==NULL)
       THEN
           SET o.Eval -= 1 ,
           IF (e.Answer=='Rep') THEN o.Rep -=1
           IF (e.Answer=='Dem') THEN o.Dem -=1
```

Proving termination requires considering the cycles in the triggering graph and reasoning about their mutual triggering [8]. Updates to `Object_ CTRL` may cause the closure of objects (see Sect. 4.3.4), but the condition of the corresponding rule may become true only due to increments of the `Rep` and `Dem` counters (which should be equal to given thresholds), while the above rule performs decrements; thus, the condition of that rule fails. Updates to `Object_ CTRL` may also cause updates of `Performer_ CTRL` and the re-classification of the performer as a spammer; in such case a second instance of Rule 8 would be triggered, but the condition on the specific performer would fail. Thus, at least one triggered rules along every potential cycle has a false condition, and no cyclic behavior can occur.

Note that termination analysis for this rule must consider the actual rule set with mutual triggering and conditions, as the proof of termination cannot be inferred from the structure of the PG graph [76].

Re-planning

Re-planning occurs when, during execution, the answers accumulated so far do not guarantee convergence to a result. For instance, a system execution could require a majority of three responses for a given object and start by planning exactly 3 micro-task executions. Then, on the first conflictual answer, the system could plan to add 3 more micro-tasks, by using the `Plan` function, as shown in Rule 9:

Rule 9 ReplanWhenNoMajority.

```
e: UPDATE FOR Object_CTRL
c: NEW.Eval==3 and NEW.Dem >=0 and NEW.Rep >=0
a: action PLAN(NEW.oid, 3)
```

A different re-planning could be triggered by a timer event, e.g., in the case that the designer wishes to add 10 micro-tasks so as to quickly produce additional executions for each object with less than 30 evaluations, as shown by Rule 10.

Rule 10 ReplanWhenTimeout.

```
e: TIMER for Object_CTRL
c: Object_CTRL.Eval<30
a: action PLAN(Object_CTRL.oid, 10)
```

The `PLAN` function performs the planning of new micro-tasks which are then described through suitable tuples in the control and aggregate tables; this function therefore creates new crowdsourcing activities until the condition on evaluations is met, but rule termination is not affected.

Many complex planning rules can be programmed, as demonstrated in the experiments of Section 4.4; for instance, rules could compute the quality of performers and the difficulty of closing objects, and then assign difficult objects to good quality performers.

4.4 Experiments

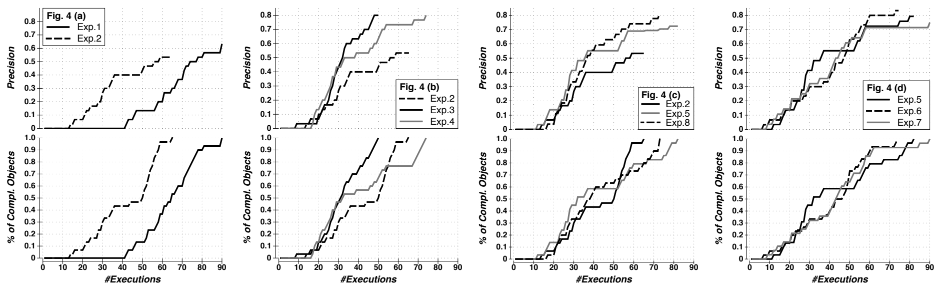


Figure 4.3: *Results precision and percentage of completed objects over micro-task executions.*

I implemented the reactive control approach in CrowdSearcher (it will be further discussed in Chapter 8). Rules are written in Javascript Each relational control rule is directly translated into a Javascript file; triggering is modelled through internal platform events. I developed three applications which demonstrate the flexibility and expressive power of reactive crowdsearching for different aspects of application design and deployment. 284 performers were recruited (mainly trough public mailing lists and social networks announcements), and 3500 micro-tasks were performed.

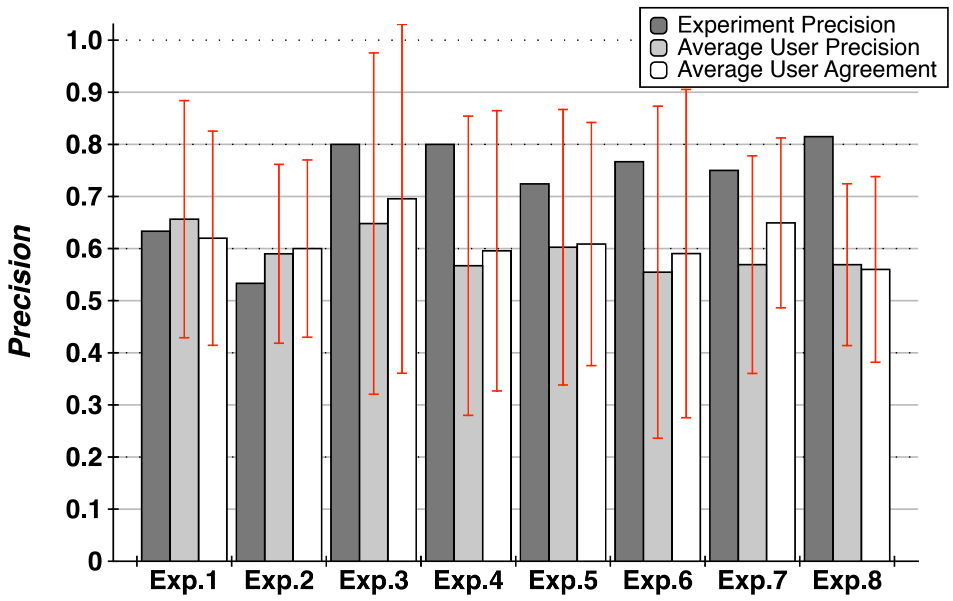


Figure 4.4: *User description of the* ***politician classification****.*

In the **politician's crimes** application, the system pre-sents photos of 50 members of the Italian parliament and asks performers to indicate if they have ever been accused, prosecuted or convicted. From the performers' point of view, this application is a game - of physiognomic nature, given that many faces are not known; each performer sees, in a fixed amount of time, a number of photos which raises as a function of the performer's ability; a higher number of photos gives to players higher possibility of improving their ranking. At the end of each round, the system presents a report with correct answers and the ranking of performers. Rules are used for assessing performers' ability and for creating micro-tasks based on the dynamically evolving quality of performers; I noted that games with higher variability of tasks are most appreciated by users (281 vs. 565 executions), and lead to a considerably higher evaluation precision (65% vs. 82% respectively).

In the **politician ranking** application, the task is to produce a total ranking of 25 politicians; at each interaction, the performer is presented a pair of politicians and is asked to chooses the one she likes the most. In this application, used by

Table 4.1: *Description of experiments of the politician classification experiment.*

	Description	Rules	NE	AT	NR	NP	NS	PR
`Exp.1`	*Maj@7*		90	21	N/A	17	N/A	0.63
`Exp.2`	*Maj@3/Maj@7*	`9`	65	29	17	12	N/A	0.53
`Exp.3`	*Maj@3/Maj@5/Maj@7*	`9`	50	40	20	11	N/A	0.80
`Exp.4`	*Maj@3/Maj@7/Maj@15*	`9`	74	21	20	22	N/A	0.80
`Exp.5`	*Exp.2,Early Spam@0.5*	`5,7,9`	82	31	24	24	4	0.72
`Exp.6`	*Exp.2,Early Spam@0.6*	`5,7,9`	74	32	20	27	8	0.83
`Exp.7`	*Exp.2,Early Spam@0.7*	`5,7,9`	90	48	27	22	8	0.75
`Exp.8`	*Exp.2,Late Spam@0.5*	`4,7,9`	73	17	23	24	2	0.81

159 distinct performers, the system performs an ordering task by splitting it into micro-tasks with pairwise politician comparisons. Control production rules update the current `score` of politicians after each comparison by using the ELO rating system [34].

Finally, in the **politician classification** application, performers are asked to classify the political affiliation of 30 members of the Italian parliament – this application is similar to the running case study of the chapter, with *six* Italian parties instead of two. Performers are provided with a set of photos of politicians, with associated names; there is no time limit, and performers are encouraged to use search engines. This application is therefore an example of human computation whose execution control aims at result precision and spammer detection.

Within the politician classification application I performed eight experiments, each featuring a different set of control rules similar to those presented in Section 3; a total of 593 micro-tasks were executed, involving 105 unique users. Table 4.1 reports, for each configuration, a short description, the rules that were used in the specific configuration, the number `NE` of executions, the average duration time `AT` of each micro-task, the number `NR` of object re-planning, the number `NP` of performers, the number `NS` of identified spammers, and the precision `PR` at the end of the experiment. All experiments use variations of the control rules 1, 2, and 6, and of the Appendix 4.B; the table reports only the optional rules, specific to the experiment. Tasks are considered closed when all their objects are fully evaluated.

Figure 4.4 shows the overall experiment's precision, the average performer's precision, and the average degree of agreement of performers. I measured the users' and experiments' precision against the available gold truth (the actual affiliation of each politician), but I did not used it for controlling spammers, so to simulate the worst case scenario were the attainable truth is the one agreed by the involved performers.

Fig. 4.3 (a) - (d) plot the precision and number of object evaluations as a function of the completed micro-task executions. I stressed four analysis dimensions: *result production*, *object re-planning*, *spammer identification*, and *spammer threshold tuning*.

Result production. I experimented with two result production policies. `Exp.1` – *Late Policy* – produces object's results as majority answers after 7 evaluations; `Exp.2` – *Early policy* – adopts the *Majority Result* policy of Rule 6, where a result is immediately produced if, after three executions, all the involved performers agree; otherwise, 4 additional evaluations are planned. As shown in Table 4.1 and Figure 4.3, the *Early policy* is able to considerably reduce the number of executions required for task closing, at the cost of a considerable quality penalization due to the possibility of performers agreement on wrong classification.

Object replanning. In the second set of experiments (`Exp.2`, `Exp.3`, and `Exp.4`), I compared three variations of the control logic expressed by Rule 9 for object replanning. While `Exp.2` performs a single stage of replan for objects which fail to have an early majority after 3 evaluations, `Exp.3` and `Exp.4` adopt a two-staged replanning policies, respectively testing for majority on each object after 5/7 evaluations or after 7/15 evaluations. As displayed in Figure 4.3 b), both `Exp.3`, and `Exp.4` achieve a considerably higher precision w.r.t. `Exp.2`; they differ for the number of executions required for completion.

Spammer identification. The third set of experiments, displayed in Figure 4.3 d), exploited the re-planning policy of `Exp.3` while adding spamming detections capabilities. `Exp.5` and `Exp.8` respectively implement the *early* and *late* evaluation of wrong answers (by rules 5 and 4) and a variant of Rule 7 where spammers are identified as performers with at least 50% of wrong answers. Both experiments required an higher number of micro-task executions compared to `Exp.3`; `Exp.5` detected 8 spammers, while `Exp.8` detected only 2 spammers.

Spammer threshold tuning. Finally, I performed a fine tuning of the threshold for judging a performer as spammer. Setting the threshold is critical: while a high threshold value may miss spammers, a low threshold value may detect too many performers as spammers. I respectively required wrong answers to be 60% in `Exp.6` and 70% in `Exp.7`; a close comparison of solutions in Figure 4.3 shows that the intermediate choice of `Exp.6` has better performances.

Figure 4.5 shows the number of activations of the various classes of rules during the execution of the experiments. Re-planning calls for additional triggering of both control rules (more control statistics to update) and result rules (more closures to be done). Execution rules only depend on the re-planning policies, and their executions are slightly higher for spamming control experiments as they have to recompute the aggregates relative to invalid micro-tasks of spammers.

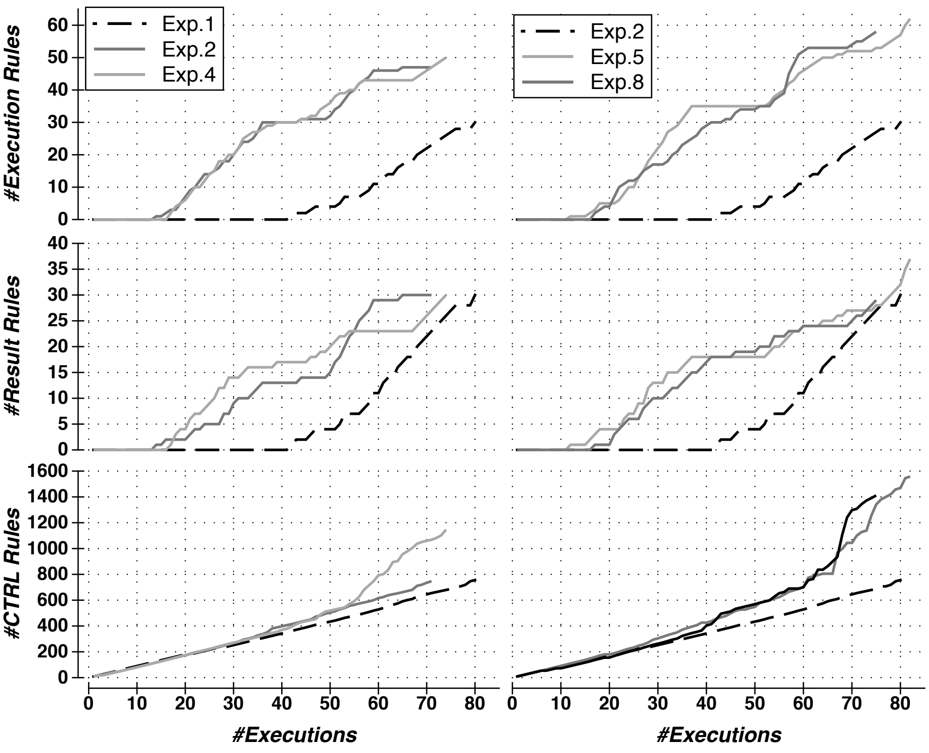


Figure 4.5: *Activations of rules of different classes.*

4.5 Conclusions

Supporting the dynamic control of crowdsourcing applications is increasingly important; however, most crowdsourcing platforms do not provide adequate solutions. Many platforms hide the control logic, a few expose limited program interfaces. In this framework I focused on reactive control of human computations by designing and deploying active rules for crowdsourcing control; my approach is complemented by a design method for crowdsourcing applications which uses standard operation types and model-driven transformations. In this way, rules have a default version that can be automatically derived from application design, but they can be modified or extended so as to implement arbitrary and sophisticated control policies.

I detailed in a running case study the exact structure of the relational data (control mart) and of rules, showing that simple rule substitutions or condition re-writings enable the encoding of different control policies; these are presented through extensive examples applied to classical human computations. The proposed approach is a good compromise between the conflicting requirements of design automation, flexibility, and expressive power.

4.A Grammar of Rule Language

```
<rule> ::= 'rule' <rulename> 'e:' <eventclause> ['c:' <condition-clause>] 'a:' <action>

<event-clause> ::= 'TIMER FOR' <TABLE> <timer-expression>
| 'INSERT FOR' <TABLE> | 'DELETE FOR' <TABLE>
| 'UPDATE FOR'       \\  <TABLE> ['['<ATTRIBUTE>\{,<ATTRIBUTE>\}']']

<condition-clause> ::= ( <predicate> ) | 'not' <predicate>
| <predicate> 'and' <predicate> | <predicate> 'or' <predicate>

<predicate> ::= <expression> <comp> <expression>

<expression> ::= <expression> <op> <expression>
| <op> <expression> \alt (<expression>) | <constant>
| <variable>.<ATTRIBUTE> | <selector>.<ATTRIBUTE>

<selector> ::= <TABLE>[<condition-clause>]

<action> := <statement> [\{,<statement>\}]

<statement> ::= 'IF' <condition-clause> 'THEN' <action> ['ELSE' <action>]
| 'FOREACH' <variable> 'IN' \\<selector>.<ATTRIBUTE><action>
| 'SET' <selector>'.'<ATTRIBUTE> '=' <expression>
| 'SET' <variable>'.'<ATTRIBUTE> '=' <expression>
| 'DELETE FROM' <selector>
| 'INSERT INTO' <TABLE> (<expression> \{,<expression>\})
| <FUNCTION> (<parameter>\{,<parameter>\})

<parameter> ::= <variable> | <constant>

<variable> ::= 'NEW' | 'OLD' | <variable-name>

<comp> ::= '==' | '>' | '<' | '>=' | '<=' | '!='

<op> ::= '+' | '-' | '*' | '/' | '+='

<timer-expression> ::= 'EVERY' <time-constant>
| 'AT' <time-constant>
```

<TABLE>, *<ATTRIBUTE>*, *<FUNCTION>*, *<variable-name>*, *<constant>*, *<time-constant>* are strings

4.B Rules For the Running Example

The complete rule set of the example is constituted by the seven rules below and by rules 1, 2, 6; and optionally one of (3, 4, 5), 7, 8, and one of (9, 10).

```
rule ObjectEvalCounter
     e: UPDATE FOR Execution[Answer]
     a: SET Object_CTRL[oid==NEW.oid].Eval += 1

rule PerformerEvalCounter
     e: UPDATE FOR Execution[Answer]
     a: SET Performer_CTRL[pid==NEW.pid].Eval += 1

rule TaskEvalCounter
     e: UPDATE FOR Execution[Answer]
     a: SET Task_CTRL[tid==NEW.tid].CompExec += 1

rule CurrentMajorityDem
     e: UPDATE FOR Object_CTRL[Dem,Rep]
     c: NEW.Dem > NEW.Rep
     a: SET NEW.Answer = 'Dem'

rule CurrentMajorityRep
     e: UPDATE FOR Object_CTRL[Dem,Rep]
     c: NEW.Rep > NEW.Dem
     a: SET NEW.Answer = 'Rep'

rule CurrentMajorityTie
     e: UPDATE FOR Object_CTRL[Dem,Rep]
     c: NEW.Rep == NEW.Dem
     a: SET NEW.Answer = 'Undefined'

rule TaskControlOnClosedObject
     e:  UPDATE FOR Politician[Status]
     c: NEW.Status == 'Complete'
     a: SET Task_CTRL[tid==NEW.tid].CompObj += 1
```

CHAPTER 5

Interoperable Crowdsourcing

The work described in this chapter was published in [22] and in [23]

5.1 Introduction

To get the best possible results, requestors need to dynamically adapt crowd-based applications so as to get the best quality of results, while minimising or focalising the interactions required to responders. However, in spite of the great importance of crowd adaptation and control, designing and deploying crowdsourcing applications with sophisticated controls is not well covered by existing systems, which lack methods for systematically designing complex adaptation strategies.

The main focus of this chapter is **platform and social community interoperability** of crowd-based applications, which includes both the possibility of statically determining the target crowds and crowd-based systems for an application, and also dynamically changing them, taking into account how the crowd behaves in responding to task assignments. Design-level interoperability is guaranteed by the use of a high-level, platform-independent model, that guarantees that tasks of given kinds can be deployed in a variety of ways on different systems, and that the same objects (and sometimes even performers) can be used across systems for the same application. Run-time interoperability is guaranteed by the use of a low-level, platform independent execution model, such that the tasks can be dynamically created or rerouted in response to performance monitoring. To the best of my knowledge, enabling system interoperability for crowd-based applications is new, and opens important opportunities, including tuning the cost of crowdsourcing campaigns, determining the expertise, profile and likely behavior of performers, and even allowing collaboration between them based on social relations within the community.

This chapter is organized as follows: Section 5.2 dwells into application interoperability by defining a taxonomy of interoperability solutions; in particular, it introduces platform and community interoperability, as well as other interoperability dimensions. Then, Section 5.3 illustrates several rules for achieving platform and community interoperability in the context of two applications, respectively dedicated to the classification of movie scenes and of professors photos extracted from Google Images. Section 5.4 presents our experimental results, showing how the applications are actually able to dynamically involve different platforms and communities, with different characteristics, cost, and quality of results. Finally Section 5.5 concludes.

5.2 Interoperability

Interoperable crowd-based applications are deployed over multiple crowd-based systems, including social networks and crowdsoucing platforms; they can change their deployment settings either prior to being launched or during execution. Two kinds of changes are possible:

- **Adaptation** is any change of allocation of the application to crowd-based systems or to their performers.
- **Migration** is the moving of the application from a given system to a different one.

Migration is a special case of adaptation. Adaptation requires replanning and reinvitation:

- **Replanning** is the process of generating new microtasks.
- **Reinvitation** is the process of generating new invitation messages for existing or replanned microtasks, with the aim of collecting performers for them.

Adaptation can be applied at different granularity levels:

- **Task granularity**, when the replanning or reinvitation occur for the whole task.

- **Object granularity**, when the replanning or reinvitation is focused on one (or a few) objects (for instance, objects on which it is harder to achieve an agreement among performers, with a majority-based decision mechanisms).

I classify the interoperability scenarios according to several features. Interoperability applies across communities or across platforms.

- With **Cross-Platform Interoperability**, applications change the underlying social network or crowdsourcing platforms, e.g., from Facebook to Twitter or to AMT.
- With **Cross-Community Interoperability**, applications change the performers' community, e.g., from the students to the professors of a university.

Adaptation at execution time requires a **switch-over**, which denotes the time interval during which adaptation occurs. A switch-over may have joining and detached crowd-based systems:

- **Joining Systems** do not participate to the application prior to the switch-over and becomes involved after the switch-over.
- **Detached Systems** participate to the application prior to the switch-over and become not involved after the switch.

A switch-over can be continuous, instantaneous, or reset:

- With a **Continuous Switch-Over**, results of initiated tasks of detached systems are considered and contribute to the application's outcome, even if they are produced after the switch over.
- With an **Instantaneous Switch-Over**, results of initiated tasks of detached systems are considered if they were produced before the switch-over, while the production of results after the switch-over is either blocked or disregarded.
- With a **Reset Switch-Over**, all the results from detached systems are disregarded, including the ones that were submitted before the switch-over.

Adaptation at execution time is either statically or dynamically determined.

- With **Static Interoperability**, adaptation is planned, and it occurs at a given time or after receiving a given number of task responses. E.g., an application could migrate from a community to another or from a platform to another at a given time of the day, so as to meet lower costs or better performances.

- With **Dynamic Interoperability**, adaptation occurs in reaction to specific events that are observed within the task control system. This covers the case of crowds which do not respond as expected.

Dynamic interoperability is quite relevant, as crowd reactions can hardly be anticipated. Thanks to dynamic interoperability, it is also possible to guarantee certain constraints or requirements on application execution:

- **Cost Constraints** can be enforced by limiting the number of tasks which are posted to crowdsourcing systems, or by adapting their cost to the allocated budget. This is made possible by the availability of communities (e.g., on social networks) that are willing to participate without a monetary reward.
- **Time Requirements** can be dealt with by adding more processing capability, and possibly by migrating the application, e.g. from social networks which use voluntary work to crowdsourcing platforms where the work is paid.
- **Diversification Requirements** can be dealt with by involving different systems and performers communities.

5.3 Interoperabiluty Rules

In this section, I show how to model cross-platform and cross-community interoperability through active rules. Rules can be derived from the abstract interoperability properties discussed in the previous chapters; for ease of presentation, I show them in the context of two concrete scenarios.

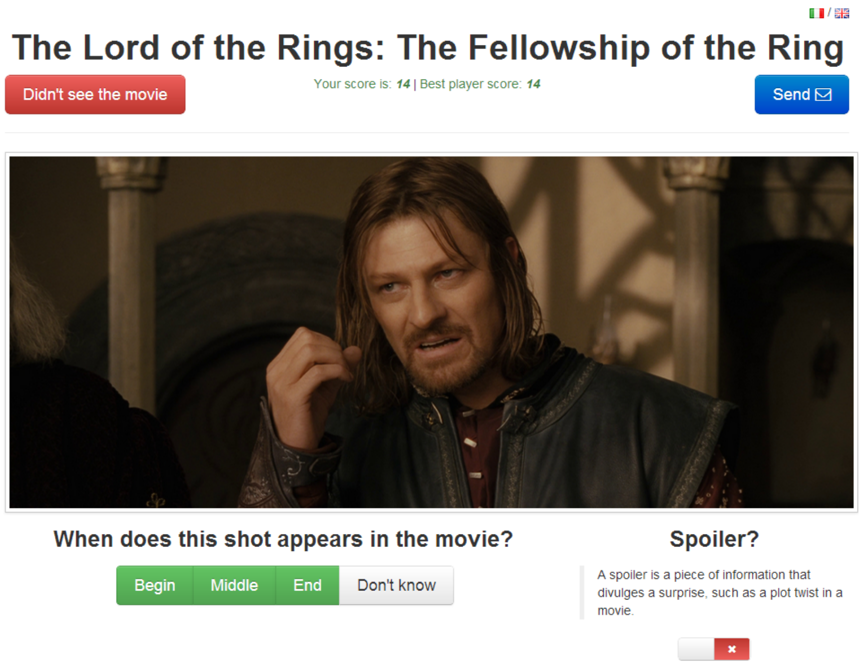
5.3.1 Cross-Platform Interoperability

The first scenario is concerned with *movie images classification*. Given a set of still images taken from a movie (produced by crawling the web site imdb.com), we ask performers to classify the image as belonging to the beginning, middle or final part of the movie. Furthermore, we ask to explicitly say if the image could be a spoiler for the movie, i.e., an anticipation of the plot which ruins the enjoyment of the movie. Given the peculiarity of the task, we let the performers declare that they haven't seen the movie (and thus they are not asked to evaluate other images of that movie) or that they don't remember the specific scene. The experimental setting is as follows:

- **Dataset:** I captured 20 still images from 16 recent and popular movies. For time positioning, we recorded the timestamp of the capture, we split the movie in three slots of the same length and we automatically assigned each capture to the corresponding slot. I manually defined a ground-truth on spoiler images, established by experts (i.e., unanimous agreement by 3 people that watched the movie).

- **Crowdsourcing:** each micro-task consists of evaluating one image. A customized, double language UI (Italian and English) is present within Crowdsearcher (as shown in Figure 5.1).[1] Results are accepted, and the corresponding request is closed, when an agreement between 5 performers is reached both on the temporal category and the spoiler option, independently on the number of executions.

- **Interoperability:** we implemented a set of static cross-platform interoperability steps, with continuous switch-over: the first step invites performers through mailing lists, the second step moves to Twitter; the third step to Facebook; and the fourth step to Amazon Mechanical Turk (AMT). Switch-overs are scheduled at every two days.


The Lord of the Rings: The Fellowship of the Ring

Your score is: 14 | Best player score: 14

When does this shot appears in the movie?

Spoiler?

A spoiler is a piece of information that divulges a surprise, such as a plot twist in a movie.

Figure 5.1: *Customized UI for the cross-platform scenario (Movies), where users can select the screenshot timeframe and whether it is a spoiler or not.*

[1]Movies experiment available at: `http://is.gd/expmovies`

Control Mart

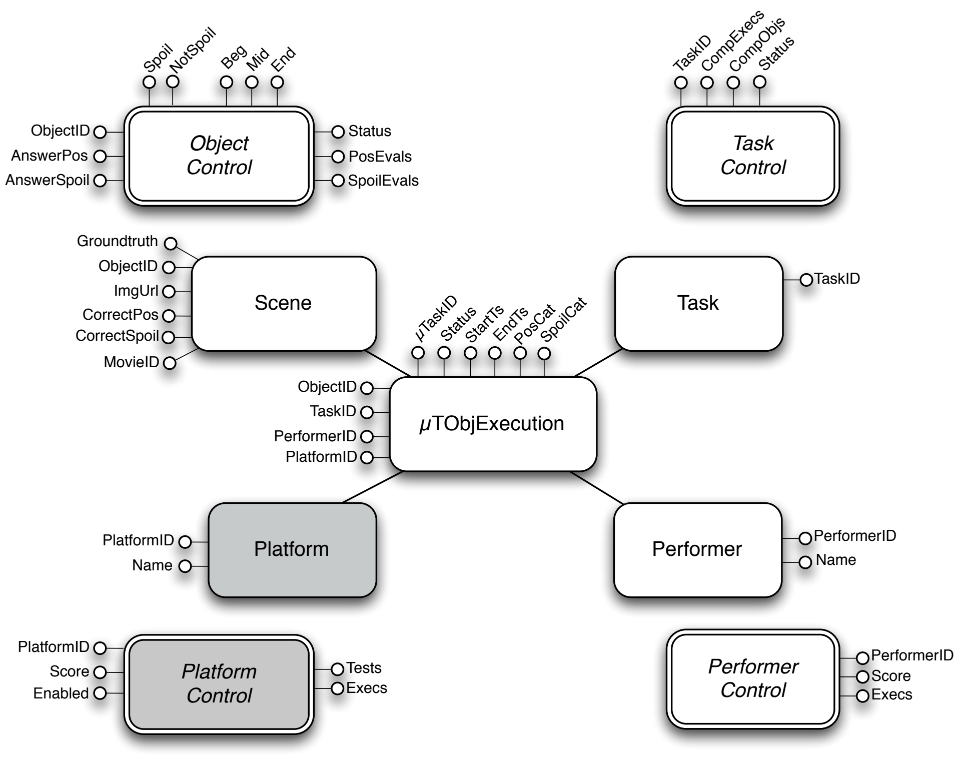


Figure 5.2: *Control mart for the Cross-Platform scenario (Movies).*

Figure 5.2 shows the control mart for the Cross-Platform scenario. In comparison with control marts introduced in Chapter 4, I added two dimensions, relative to the platform or the community used in the micro-task execution; in this figure we include the platform dimension, in the next section we include the community dimension; they are orthogonal, so an application can have both of them. The schema of the control mart is automatically constructed by taking into account the structure of data objects, the operations implemented by the task, and the rules for object, performer, task, and platform/community control.

Interoperability rules

Next, I show some exemplary rules corresponding to different interoperability options; all rules are concerned with dynamic adaptation that calls for executing on Amazon's Mechanical Turk (AMT) an application which was initiated on Facebook.

Rule 11 Static adaptation of an application which adds AMT to Facebook after 2 days, for non-closed objects.

```
e: AFTER 2 days ON TASK[TaskID == 'Movie']
c: ---
a: SET PLATFORM_ctrl[PlatformID == 'AMT'].Enabled =
true,
   replan(OBJECT_ctrl[Status<>'closed'].ObjectID,'AMT'),
   reinvite('Movie','AMT')
```

Rule 12 Maintenance of platform control data based on the groundtruth.

```
e: UPDATE for M_T_O_EXECUTION (PosCat)
c: Scene.Groundtruth[ObjectID == NEW.ObjectID] == true
a: SET PLATFORM_ctrl[platformID == NEW.platformID].Tests++
   IF NEW.PosCat == SCENE[ObjectID == NEW.ObjectID].CorrectPos
   THEN
      SET PLATFORM_ctrl[platformID == NEW.platformID].Score++
   ELSE
      SET PLATFORM_ctrl[platformID == NEW.platformID].Score--
```

- **Rule 11** replans the task on AMT after two days since it was started. Note that if the task completes before two days, then the rule does not fire and AMT is not used at all; note also that the Facebook platform remains enabled. This rule applies to all the objects that are not closed; thus, it implements a static interoperability at task granularity, with continuous switch-over.

- The next two rules are used for replanning a task when the quality of execution on the current platform is not good enough. **Rule 12** maintains platform control data by counting the number of performed tests against the groundtruth values and changing the score associated with the current platform at each correct or incorrect answer relative to objects for which a groundtruth is known. **Rule 13** migrates the application to AMT after 50 tests performed on Facebook if the average score is below 50%, by disabling Facebook and enabling AMT, and by inviting the application's perfomers on AMT; replanning applies only to the objects that are not yet closed. The two rules implement a dynamic interoperability determined at task granularity, with instantaneous switch-over.

- **Rule 14** is used for replanning a specific object, in the lack of ground truth, when the performers on Facebook are in total disagreement for that object, i.e., if there is at least one vote on every category of the classify operation; in that case, all the executions on that objects from the current platforms are invalidated. We assume that both Facebook and AMT are enabled platforms. This rule implements a dynamic interoperability determined at object granularity, with reset switch-over; this requires resetting all the counters and

Rule 13 Migration of an application from Facebook to AMT.

```
e: UPDATE for PLATFORM_ctrl (Score)
c: (NEW.Score / NEW.Tests) < 0.5 AND NEW.Tests > 50 AND
   NEW.PlatformID == 'Facebook'
a: SET PLATFORM_ctrl[PlatformID == 'Facebook'].Enabled = false,
   SET PLATFORM_ctrl[PlatformID == 'AMT'].Enabled = true,
   replan(OBJECT_ctrl[Status<>'closed'].ObjectID,'AMT'),
   reinvite('Movie','AMT')
```

Rule 14 Replanning of a single object from Facebook to AMT, with reset switch-over.

```
e: UPDATE for M_T_O_EXECUTION (PosCat)
c: OBJECT_ctrl[ObjectID == NEW.ObjectID].Beg >= 1 AND
   OBJECT_ctrl[ObjectID == NEW.ObjectID].Mid >= 1 AND
   OBJECT_ctrl[ObjectID == NEW.ObjectID].End >= 1 AND
   PLATFORM[ObjectID == NEW.ObjectID].Name = 'Facebook'
a: SET OBJECT_ctrl[ObjectID == NEW.ObjectID].Beg == 0,
   SET OBJECT_ctrl[ObjectID == NEW.ObjectID].Mid == 0,
   SET OBJECT_ctrl[ObjectID == NEW.ObjectID].End == 0,
   SET OBJECT_ctrl[ObjectID == NEW.ObjectID].PosEvals == 0,
   FOREACH e IN M_T_O_EXECUTION [ObjectID == NEW.ObjectID]
     SET e.Status = 'invalid',
   replan(NEW.ObjectID,'AMT'),
   reinvite(NEW.ObjectID,'AMT')
```

setting past executions as invalid.

5.3.2 Cross-Community Interoperability

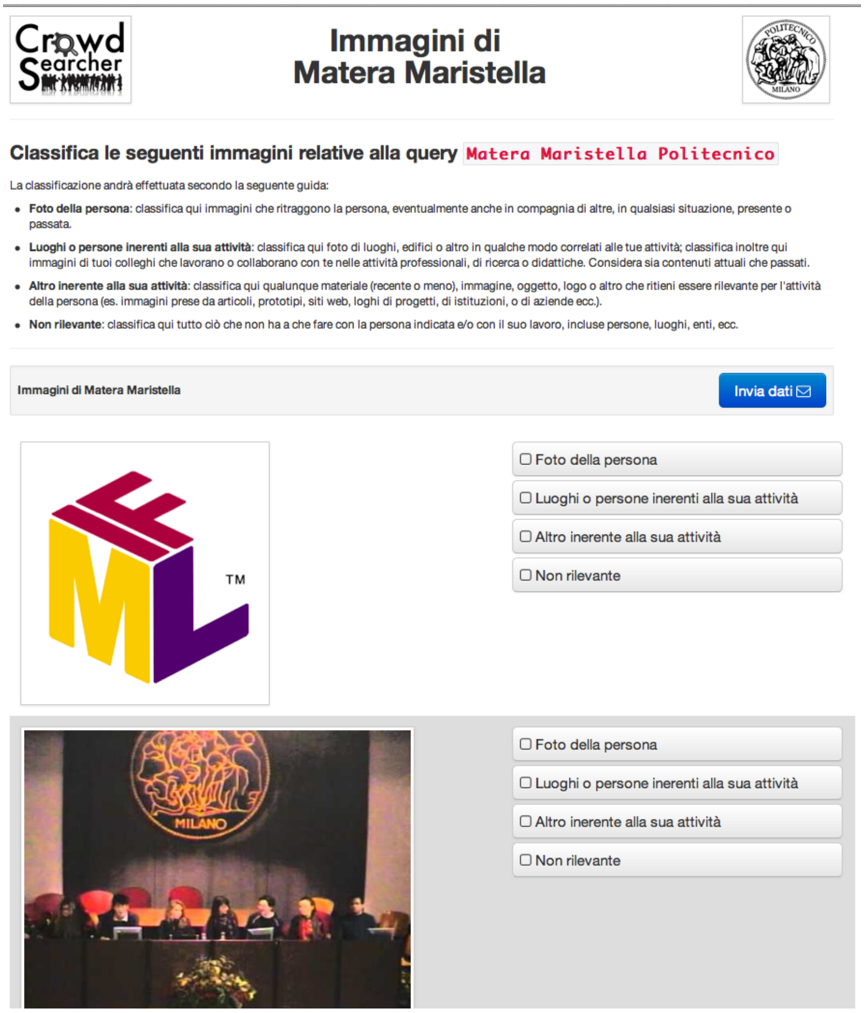
The second scenario is concerned with *image classification* and demonstrates cross-community interoperability. The dataset consists of images about professors of our department retrieved through the Google Image API. In the crowdsourcing campaign we ask the performers to specify whether each image represents the professor, or some relevant people or places, or other related materials (papers, slides, graphs or technical materials), or it is not relevant at all. The experimental setting is as follows:

- **Dataset:** I selected 16 professors within two research groups in our department (DB and AI groups) and downloaded the top 50 images returned by the Google Image API for each query (the professor's name followed by the keyword “Politecnico"); I excluded the images that were not linked or extremely small in size. I asked the professors themselves to define the ground-truth on the images, through a specific crowdsourcing task (not described here).

- **Crowdsourcing:** each microtask consisted of evaluating 5 images regarding a professor. A customized UI (in Italian) has been developed within

Crowdsearcher (as shown in Figure 5.3).[2] Results are considered accepted (and thus the corresponding object is closed) when enough agreement on the class of the image is reached among performers. Closed objects are removed from new executions.

- **Interoperability:** I defined the communities as the research group of the professor, the research area containing the group (e.g. computer science), and the whole department (which accounts for more than 600 people in different areas).



Immagini di Matera Maristella

Classifica le seguenti immagini relative alla query `Matera Maristella Politecnico`

La classificazione andrà effettuata secondo la seguente guida:

- **Foto della persona**: classifica qui immagini che ritraggono la persona, eventualmente anche in compagnia di altre, in qualsiasi situazione, presente o passata.
- **Luoghi o persone inerenti alla sua attività**: classifica qui foto di luoghi, edifici o altro in qualche modo correlati alle tue attività; classifica inoltre qui immagini di tuoi colleghi che lavorano o collaborano con te nelle attività professionali, di ricerca o didattiche. Considera sia contenuti attuali che passati.
- **Altro inerente alla sua attività**: classifica qui qualunque materiale (recente o meno), immagine, oggetto, logo o altro che ritieni essere rilevante per l'attività della persona (es. immagini prese da articoli, prototipi, siti web, loghi di progetti, di istituzioni, o di aziende ecc.).
- **Non rilevante**: classifica qui tutto ciò che non ha a che fare con la persona indicata e/o con il suo lavoro, incluse persone, luoghi, enti, ecc.

Crowd Searcher
POLITECNICO MILANO
Immagini di Matera Maristella
Invia dati
Foto della persona
Luoghi o persone inerenti alla sua attività
Altro inerente alla sua attività
Non rilevante
IFML TM
MILANO
Foto della persona
Luoghi o persone inerenti alla sua attività
Altro inerente alla sua attività
Non rilevante

Figure 5.3: *Customized UI for the cross-community scenario (Professors).*

[2]Profs experiment (in Italian) at: `http://is.gd/expprofs`

Rule 15 Invitation of CS-Area community after deadline.

```
e: AFTER 2 days ON TASK[TaskID == 'Professor']
c: ---
a: SET COMMUNITY_ctrl[CommunityID == 'CS-Area'].Enabled
= true,
   reinvite('GoogleImages','CS-Area')
```

Rule 16 Maintenance of last task execution timestamp.

```
e: UPDATE for M_T_O_EXECUTION (EndTS)
c: ---
a: SET TASK_ctrl[TaskID == NEW.TaskID].LastExec = NEW.EndTS
```

Control Mart

Figure 5.4 depicts the control mart of the Cross-Community scenario. Objects of interest are images returned by Google. In this scenario, interoperability is across communities and thus the interoperability dimension is represented by *Community* and *CommunityControl*.

Interoperability Rules

The rules of this section show how the application dynamically adapts by expanding to larger crowds.

- **Rule 15** invites performers from the research area (CS-Area) after two days since the initial invitations, which were sent to a specific research group (DB-group). Note that if the task completes before two days, then the rule does not fire and the task uses just the research group. This rule implements a static interoperability determined at task granularity, with continuous switch-over.

- The next two rules are used to invite the performers of a broader community when the current crowd ceases to produce answers. **Rule 16** saves the timestamp of the last execution of the current task; **Rule 17** invites performers of the broader community after one hour of *idle* time, i.e. when the last execution occurred one hour ago. Rules implement a dynamic interoperability at task granularity, with continuous switch over.

- **Rule 18** is used for replanning a specific object when the performers of a community are in disagreement, e.g., if there is are vote on every category of the classify operation. We assume that in this case the invitation was initially sent to CS-Area and then it is routed to the DB-Group, which is assumed to be a group of experts in recognizing images about colleagues of the same group; we also let the task evaluation for that object to continue. This

Rule 17 Invitation of CS-Area community when the DB-Group is idle.

```
e: EVERY 1 minute ON TASK[TaskID == 'GoogleImages']
c: now() - TASK_ctrl[TaskID == 'GoogleImages'].LastExec
> 1 hour
   AND COMMUNITY_ctrl[CommunityID == 'CS-Area'].Enabled=false
a: SET COMMUNITY_ctrl[CommunityID == 'CS-Area'].Enabled=true,
   reinvite('GoogleImages','CS-Area')
```

rule implements a dynamic interoperability determined at object granularity, with continuous switch-over.

5.3.3 Rule Design Principles

Termination of general control rules for crowdsourcing has been discussed in Chapter 4. I now discuss a set of best practices for ensuring the quality of interoperability rules, including their termination.

In general, I observe that interoperability rules are concerned with decisions that occur once the rules for object, task, and performer control have reached a quiescent state. Thus, interoperability rules should be a lower priority rule stratum. In addition, I assume that the actions of interoperability rules have specific limitations: they either enable or disable platforms and/or communities, they re-plan new executions upon existing objects and correspondingly activate invitation processes, and in some cases they initialize summarization data (by restoring their initial value); all these actions do not trigger object, performer, or task control rules as presented in Chapter 4. As a consequence, interoperability rules do not alter the quiescent state produced by control rules, and hence termination of the two strata can be discussed independently (see [7]).

Termination of interoperability rules, in turn, is not guaranteed if they incur into cyclic processes of activations and deactivations of platforms or communities. For instance, if both Facebook and Twitter executions do not meet the required level of quality, two rules could call opposite migrations from one platform to the other, thus producing an endless loop.

Such cause of non-termination cannot occur when platforms or communities are invoked in a sequence (and thus the number of involved platforms and commu-

Rule 18 Replanning of an object, by invoking an expert community (the DB-Group).

```
e: UPDATE for M_T_O_EXECUTION
c: OBJECT_ctrl[ObjectID == NEW.ObjectID].ProfPhoto >= 1 AND
   OBJECT_ctrl[ObjectID == NEW.ObjectID].PeoplePlace >= 1 AND
   OBJECT_ctrl[ObjectID == NEW.ObjectID].Materials >= 1 AND
   COMMUNITY_ctrl[CommunityID == 'DB-Group'].Enabled = false
a: SET COMMUNITY_ctrl[CommunityID == 'DB-Group'].Enabled = true,
```

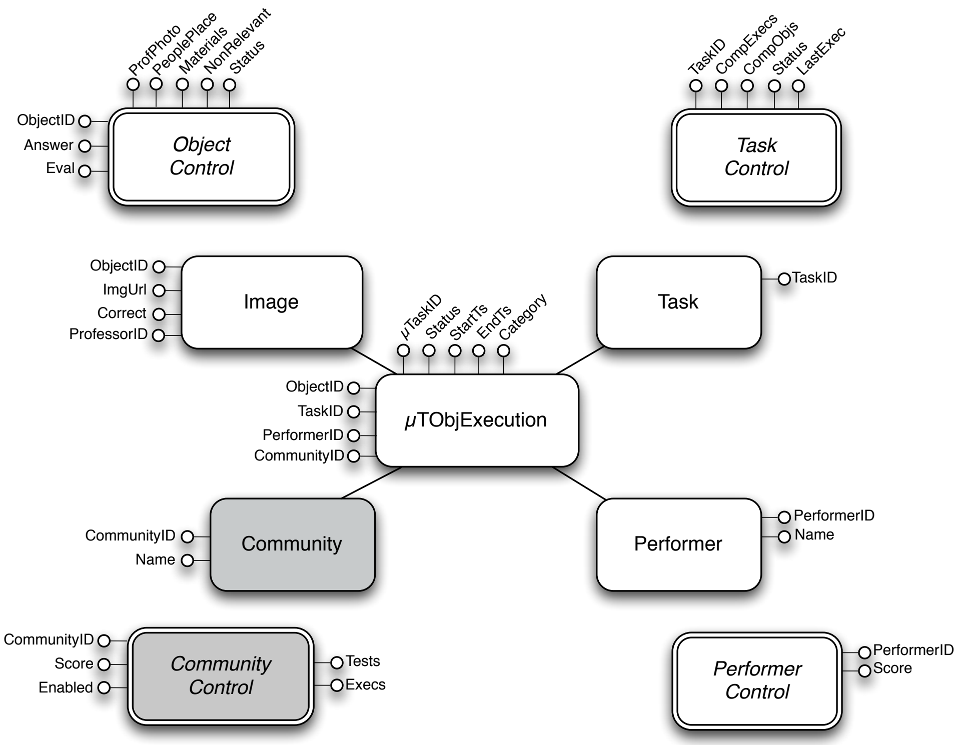


Figure 5.4: *Control mart for the Cross-Community scenario (Professors).*

nities grows monotonically); this situation arises very commonly, e.g. the *ouside-in* or *inside-out* strategies in the experiments of Section 5.4.2. When the invocation of platforms and communities is not monotonic, the designer should focus on the rules that perform enabling and disabling activities upon multiple platforms and communities, such as Rule 13, and consider their mutual triggering. Rules monitoring a single platform/community, e.g. by enabling one platform (such as Rules 11 and 14) or one community (such as Rules 15, 17 and 18), cannot be involved into enabling/disabling loops, and hence can be disregarded during such analysis.

One should also consider a different kind of termination, that we call *application termination*. Intuitively, users wish their applications to terminate fast, with the highest possible quality measures, and involving as little resources as possible, where resources not only include the cost of crowd activities, but also the amount of cooperation required from communities; in many cases, one would like to minimize crowd interactions so as not to abuse of contacts. Thus, designing rules for application adaptation is quite complex, and it can hardly be modeled as an optimization problem.

In Crowdsearcher (see Chapter 8) I support a declarative approach to application design, in which users describe interoperability requirements with high-level specifications (i.e., by indicating the platforms or expert communities, their sequence of invocation with the *outside-in* or *inside-out* strategy, when and how switch-over should occur) and then rules are automatically produced from high-level specifications; Crowdsearcher covers the most significant options, and offers to knowledgeable designers interfaces presenting the reactive implementation, so as to fine-tune the produced rules and meet better global performances. Note that Crowdsearcher offers an environment for fast prototyping of experiments which allows a progressive tuning of execution rules, as I did in the experiments reported in Section 5.4.

5.4 Experiments

I performed several experiments on the two scenarios described in Section 5.3. The groundtruth and results are available online.[3]

5.4.1 Cross-Platform Scenario

In the first experiment, I considered 4 platforms for inviting performers (email, Twitter, Facebook, and Amazon Mechanical Turk), and I redirected all the performers to a customized UI. The first round of invitations was sent to the mailing lists of students of 8 courses; subsequently, I engaged people on Twitter, then Facebook, and finally I posted HITS on AMT, with a payment of 1 cents per execution. The change of platform was performed through static interoperability at task granularity, using a continuous switch-over every 2 days, by using a rule similar to Rule 11. Objects were randomly assigned to executions, but performers never evaluated the same object twice. Objects (i.e., screenshots) were considered correctly evaluated (and thus closed) when agreement of 5 votes was reached on the same category, both for the positioning in time (beginning, mid, end) and for spoiler labeling (yes, no).

The number of performers that actually participated from each platform and their precision are reported in Table 5.1; precision is calculated in terms of correct answers for each evaluation. Figure 5.5 shows the number of executions and of performers for each platform, along the time of the experiment (one week). The graphs show substantial participation of people invited via email, which was an engaging invitation platform, given that movie classification was considered as a nice game by involved students. Invitations to circles of friends using Twitter were less popular, while invitations using Facebook were more popular. Note that after some time the executions within all the three social networks reach a plateau, also due

[3]`http://crowdsearcher.search-computing.org/multiplatform-and-community`

to the specific time (night-time and the weekend show very limited participation; in our scenario, all performers where from the same timezone).

Invitations on social networks were generated through various techniques: for instance, on Facebook, engagement was increased by first sharing a link, then an image, and finally by creating an event and inviting people to it. I also noticed that spontaneous resharing on social networks contributed to involving more people. When I turned to a crowdsourcing platform, I got a high number of performed with a small pay (1 cent per HIT). Performers on AMT closed the remaining objects of the experiment in few hours, despite such low pay.

Figure 5.6 shows in log scale the number of closed objects in time vs. the number of performed evaluations (log scale) for the Cross-Platform scenario. A small number of objects start closing only after 400 evaluations, while most of them closed after 800 evaluations. This is due to the need of reaching agreement between 5 performers on the same answer. A parallel experiment (not reported here) with agreement threshold set to 3 was much faster (all experiments closed in a matter of hours, instead of days).

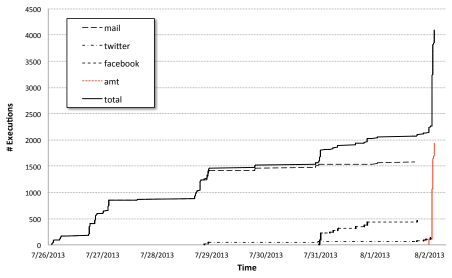
Figure 5.7 (a) shows the aggregate precision by performer along time and by platform; each curve starts from the invitation time on the respective platform. The X axis represents the number of evaluations, and therefore the contribution of AMT is greater (while it was very compressed in time in the other graphs). Initial peaks on each platforms are not significant,they represents the oscillations of the first evaluations.

Precision of performers invited via email is higher than precision of performers invited by Twitter and AMT, while Facebook positions in the middle. Note also that precision is slightly decreasing, as the first evaluations are involved with objects whose classification is easier and less error-prone. This effect can be seen also on Figure 5.7(b), which shows the actual precision of the system by keeping the two classification operations (scene positioning and spoiler detection) separate.

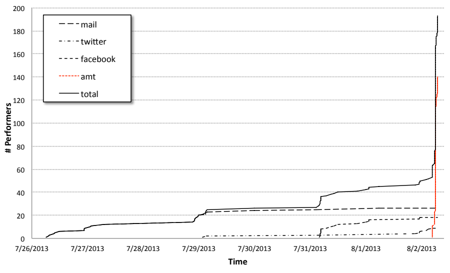
The system's precision is evaluated after an agreement on five evaluations, hence it is higher than execution precision; the final system's precision is respectively 0.85 for spoiler classification and 0.62 for scene positioning. Note that the precision of spoiler detection is higher and does not change much with time, while the precision of image detection reduces with time, both because the last scenes to be closed are most difficult to classify and because AMT users become more relevant, and they have a lower individual precision.

Table 5.1: *Performers and average precision of evaluations for each platform in Cross-Platform scenario.*

Platform	#Performers	Precision
Mail	26	0,73
Twitter	9	0,64
Facebook	18	0,69
AMT	140	0,63
Total	193	0,67



(a)



(b)

Figure 5.5: *Number of executions (a) and performers (b) by platform for the Cross-Platform scenario.*

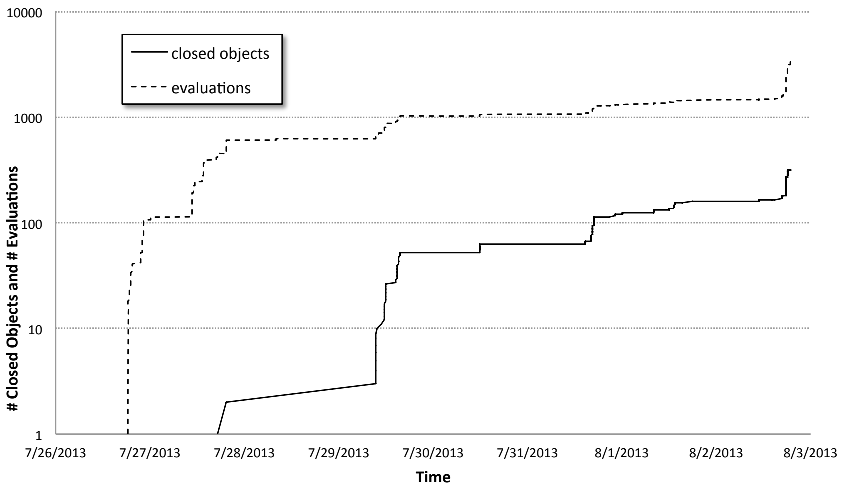


Figure 5.6: *Number of closed objects vs. number of performed evaluations (log scale) for the Cross-Platform scenario.*

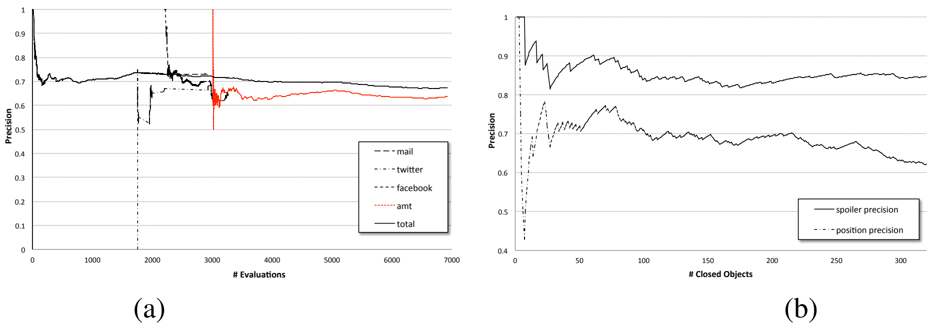


Figure 5.7: *Precision of evaluations by platform (a); and precision on closed objects after majority is applied (b) for the Cross-Platform scenario.*

s

5.4.2 Cross-Community Scenario

I next report several experiments of the cross-community scenario described in Section 5.3. I devised two experiments: in the first one, named *inside-out*, I started with invitations to experts, e.g. people the same groups as the professor (DB and AI), and then expanded invitations to Computer Science, then to the whole Department, and finally to open social networks (Alumni and PhDs communities on Facebook and LinkedIn); in the second one, named *outside-in*, I proceeded in the opposite way, starting with the Department members, then restricting to Computer Scientists, and finally to the group's members.

All invitations (except for the social networks in the first experiment) were sent by email by the system. The communities were not overlapping: every performer received only one invitation. For doing that, the members of the Department, of Computer Science area, and of the DB Group were randomly split into two sets. Invitations have been implemented as a set of dynamic, cross-community interoperability steps, with task granularity and with continuous switch-overs starting one working day after a community was idle (stopped to produce results); interoperability control rules very similar to Rules 16 and 17.

Table 5.2 shows the number of invitations sent out, the number of performers responding, and the average precision of their evaluations. Notice that precision is decreasing when moving towards less expert people, while the social network had good precision as the invitation was posted on groups that know very well the people involved (who were their professors or advisors).

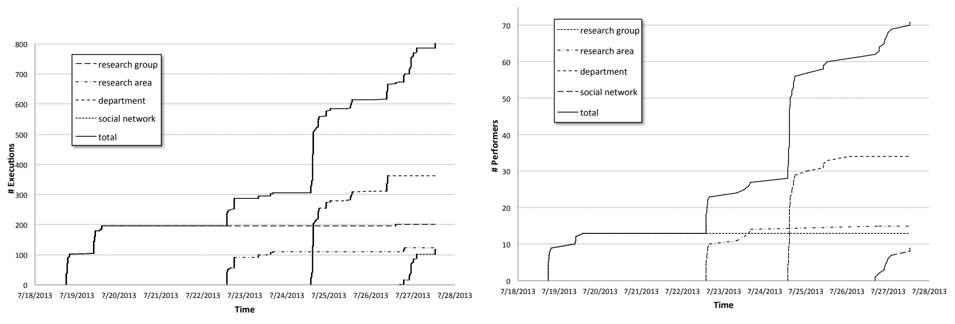
Figure 5.8 shows the number of executions (a) and performers (b) by community. Again, influence of nighttime and weekend on executions is very evident. Figure 5.9 shows the number of closed objects vs. the number of performed evaluations. Figure 5.10 (a) shows the precision of evaluations by community and

Table 5.2: *Cross-Community scenario statistics*

Community	#Invites	#Performers	Precision
Research Group	28	13	0,68
Research Area	61	15	0,64
Department	214	34	0,58
Social Networks	N/A	9	0,65
Total	303	71	0,63

Figure 5.10(b) shows the final precision on closed objects.

Figure 5.10(b) compares also the precisions of the *inside-out* and *outside-in* experiments, and shows that former performs better than the latter in terms of quality of results. This is quite evident in the initial phases (when the first half of the objects close), as the performance of experts (research group) is much higher than performance of the people of the rest of department.



(a) (b)

Figure 5.8: *Number of executions (a) and performers (b) by community for the Cross-Community scenario.*

5.5 Conclusions

This chapter proposes an empowered programming and control of crowdsourcing applications, through the use of multiple crowdsourcing platforms and social networks. We describe a taxonomy of interoperability scenarios, and show how each scenario can be implemented through suitable active rules; we also disclose general design principles for interoperability, and show the rules at work in applications which engage their performers through cross-platform and cross-community

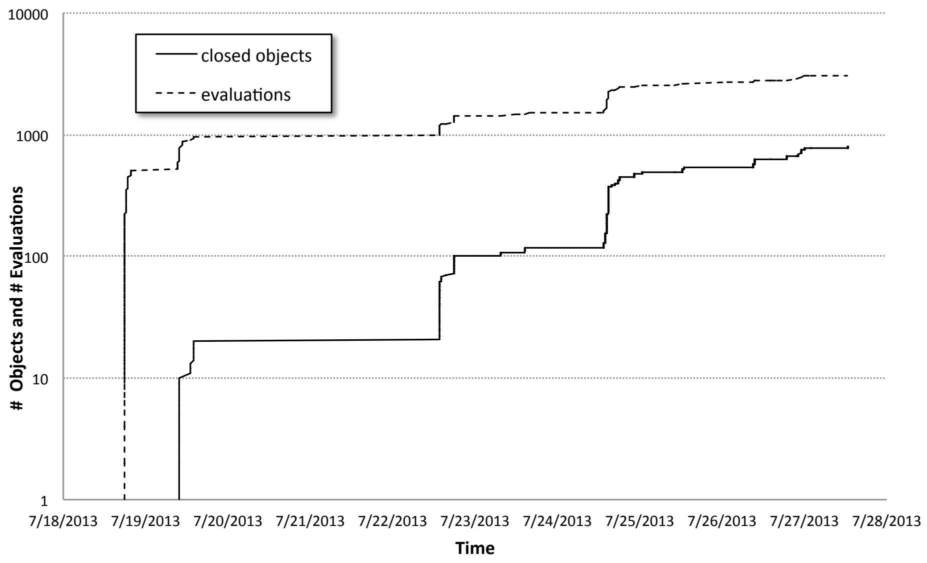


Figure 5.9: *Number of closed objects vs. number of performed evaluations (log scale) for the Cross-Community scenario.*

invitations.

Experimental results show that the method can improve the effectiveness and efficiency of crowd-based applications, by improving quality through dynamic re-planning strategies. Experiments let us collect interesting lessons learned regarding interoperability, especially with respect to the social communities. We noticed that expert performers have a completely different attitude towards the tasks: in a sense, they felt more involved and part of a “mission", they frequently contacted us (about 30% of performers sent us messages) for providing feedback for improving the application, they way questions were asked, or even the dataset. They wanted to understand the purpose of the work, commented on the results, and expected a feedback after the experiment was completed. Participants appeared more demanding than generic crowds with respect to the quality both of the application UI and of the evaluated objects. The limited number of participants implied a strong impact of the temporal aspect (responses come in more slowly than in traditional crowdsourcing systems). One obvious aspect to be considered in cross-platform interoperability is the reach and visibility of the requestor upon the involved social networks: the more people receive and re-share the invites, the highest the probability of getting more performers.

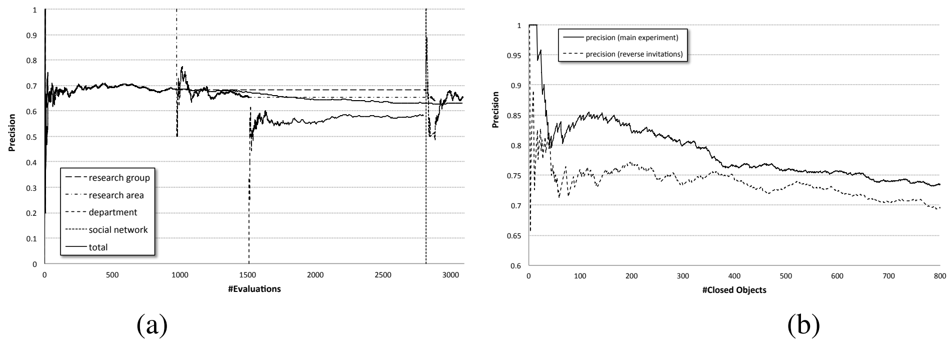


Figure 5.10: *Precision of evaluations by community (a) and comparison of the precision for the inside-out and outside-in approaches (b) in the Cross-Community scenario.*

CHAPTER 6

Crowdsourcing Design Patterns

The work described in this chapter was published in [18]

6.1 Introduction

The goal of this chapter is to present a systematic approach to the design and deployment of crowd-based applications as arbitrarily complex workflows of elementary tasks, which emphasises the use of crowdsourcing patterns. While the previous chapters I addressed the design and deployment of a single task, in this chapter I model and deploy applications consisting of arbitrarily complex task interactions, organised as a workflow; I use either *data streams* or *data batches* for data exchange between tasks, and illustrate that tasks can be controlled through *tight coupling* or *loose coupling*. I also show that my model supports the known crowd management patterns, and in particular I use my model as a unifying framework for a systematic classification of patterns.

The chapter is structured as follows. Section 6.2 introduces the task and workflow models and design processes. Section 6.3 details a set of relevant crowdsourcing patterns. Section 6.4 illustrates how workflow specifications are embodied within the execution control structures of Crowdsearcher, and finally Section 6.5.3 discusses several experiments, showing how differences in workflow design lead to different application results.

6.2 Models and Design of Crowd-based Workflows

A **crowdsourcing workflow** is defined as a control structure involving two or more interacting tasks performed by humans. Tasks have an input buffer that collects incoming data objects, described by two parameters: 1) The **task size**, i.e. the minimum number of objects (*m*) that allow starting a task execution; 2) The **block size**, i.e. the number of objects (*n*) consumed by each executions.

Clearly, $n \leq m$, but in certain cases at least m objects must be present in the buffer before starting an execution; in fact n can vary between 1 and the whole buffer, when a task execution consumes all the items currently in the buffer. Task execution can cause **object removal**, when objects are removed from the buffer, or **object conservation**, when objects are left in the buffer, and in such case the term **new items** denotes those items loaded in the buffer since the last execution.

Tasks communicate with each other with **data flows**, produced by extracting objects from existing data sources or by other tasks, as streams or batches. **Data streams** occur when objects are communicated between tasks one by one, typically in response to events which identify the completion of object's computations. **Data batches** occur when all the objects are communicated together from one task to another, typically in response to events related to the closing of task's computations.

Flows can be constrained based on a condition associated with the arrow representing the flow. The condition applies to properties of the produced objects and allows transferring only the instances that satisfy the condition. Prior to task execution, a **data manipulator** may be used to compose the objects in input to a task, possibly by merging or joining incoming data flows.

Tasks can be represented within workflows as described in Fig. 6.1, where each task is equipped with an input buffer and an optional data manipulator, and may receive data streams or data batches from other tasks. Each task consists of micro-tasks which perform given operations upon objects of a given object type; the parameter r indicates the number of executions that are performed for each micro-tasks, when statically defined (default value is 1). Execution of tasks can be performed according to intra-task patterns, as described in Section 6.3.

6.2.1 Workflow Design

Workflow design consists of designing tasks interaction; specifically, it consists of defining the workflow schema as a directed graph whose nodes are tasks and whose edges describe dataflows between tasks, distinguishing streams and batches.

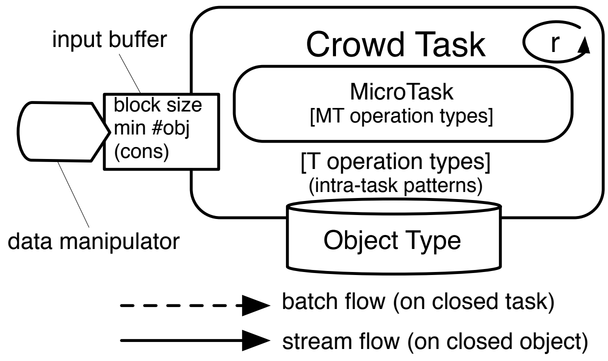


Figure 6.1: *Notation for crowdsourcing workflows*

In addition, the coupling between tasks working on the same object type can be defined as loose or tight.

Loose coupling is recommended when two tasks act independently upon the objects (e.g. in sequence); although it is possible that the result of one task may have side effects on the other task, such side effects normally occur as an exception and affect only a subset of the objects. Loosely coupled tasks have independent control marts and monitoring rules (as described in Chapter 4).

Tight coupling is recommended when the tasks intertwine operations upon the same objects, whose evolution occurs as combined effect of the tasks' evolution; tightly coupled tasks share the same control mart and monitoring rules.

Figure 6.2 shows a simple workflow example in the domain of movie scenes annotation. The *Position Scenes* tasks asks performers to say whether a scene appears at the beginning, middle or end of the film; it is a classification task, one scene at a time, with 5 repetitions and acceptance of results based on an agreement threshold of 3. Scenes in the ending part of the movies are transmitted to the *Spoiler Scenes* task, which asks performers whether the scene is a spoiler or not;[1] scenes at the beginning or in the middle of the movie are transmitted to the *Order Scenes* task, which asks performers to order them according to the movie script; each micro-task orders just two scenes, by asking the performer to select the one that comes first. The global order is then reconstructed. Given that all scenes are communicated within the three tasks, they are considered as tightly coupled.

[1]A *spoiler* is a scene that gives information about the movie's plot and as such should not be used in its advertisement.

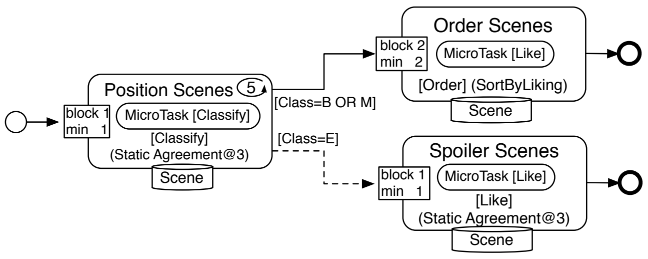


Figure 6.2: *Example of crowd flow.*

6.3 Crowdsourcing Patterns

Several patterns for crowd-based operations are defined in the literature. I review them in light of the workflow model of Section 6.2. I distinguish them in three classes and I implement them in Crowdsearcher (see Section 6.4):

- **Intra-Task Patterns.** They are typically used for executing a complex task by means of a collection of operations which are cognitively simpler than the original task. Although these patterns do not appear explicitly in the workflow, they are an essential ingredient of crowd-based computations.

- **Workflow Patterns.** They are used for solving a problem by involving different tasks, which require a different cognitive approach; results of the different tasks, once collected and elaborated, solve the original problem.

- **Auxiliary Patterns.** They are typically performed before or after both intra-task and workflow patterns in order either to simplify their operations or to improve theirs results.

6.3.1 Intra-Task Patterns

Intra-task patterns apply to complex operations, whose result is obtained by composing the results of simpler operations. They focus on problems related to the planning, assignment, and aggregation of micro tasks; they also include quality and performer control aspects. Figure 6.3 describes the typical set of design dimensions involved in the specification of a task. When the operation applies to a large number of objects and as such cannot be mapped to a single pattern instantiation, it is customary to put in place a *splitting strategy*, in order to distribute the work, followed by an *aggregation strategy*, to put together results. This is the case in many data-driven tasks stemming from traditional relational data processing which are next reviewed.

Consensus Patterns. The most commonly used intra-task patterns aim at producing responses by replicating the operations which apply to each object, collecting multiple assessments from human workers, and then returning the answer which is more likely to be correct. These patterns are referred to as *consensus* or *agreement patterns*. Typical consensus patterns are: *a*) **StaticAgreement** [17]: accepts a response when it is supported by a given number of performers. For instance, in a tag operation I consider as valid responses all the tags that have been added by at least 5 performers. *b*) **MajorityVoting** [62]: accepts a response only if a given number of performers produce the same response, given a fixed number of total executions. *c*) **ExpectationMaximisation** [31]: adaptively alternates between estimating correct answers from task parameters (e.g. complexity), and estimating task parameters from the estimated answers, eventually converging to maximum-likelihood answer values.

Join Patterns. Crowd join patterns, studied in [56], are used to build an equality relationship between matching objects in the context of crowdsourcing tasks. I identify: *a*) **SimpleJoin** consists in defining microtasks performing a simple classification operation, where each execution contains a single pair of items to be joined, together with the join predicate question, and two buttons (Yes, No) for responding whether the predicate evaluates to true or false; *b*) **OneToManyJoin** is a simple variant that includes in the same microtask one left object and several right candidates to be joined; *c*) **ManyToManyJoin** includes in the same microtask several candidate pairs to be joined;

Sort Patterns. Sort patterns determine the total ordering of a set of input objects. The list includes: *a*) **SortByGrouping** [56] orders a large set of objects by aggregating the results of the ordering of several small subsets of them. *b*) **SortByScoring** [56] asks performers to rate each item in the dataset according to a numerical scale. *c*) **SortByLiking** [17] is a variant that simply asks the performer to select/like the items they prefer. The mean (or sum) of the scores achieved by each image is used to order the dataset. *d*) **SortByPairElection** [17] asks workers to perform a pairwise comparison of two items and indicate which one they like most. Then ranking algorithms calculate their ordering. *e*) **SortByTournament** [74], presents to performers a tournament-like structure of sort tasks; each

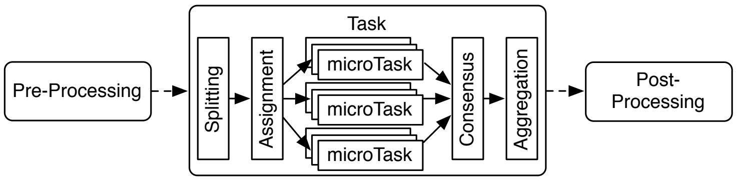


Figure 6.3: *Building blocks of an Intra-Task Pattern.*

tournament elect its champions that progress to the next level, eventually converging to a final order.

Grouping Patterns. Grouping patterns are used in order to classify or clustered several objects according to their properties. I distinguish:
a) **GroupingByPredefinedClasses** [30] occurs when workers are provided with a set of known classes. *b*) **GroupingByPreference** [2] occurs when groups are formed by performers, for instance by asking workers to select the items they prefer the most, and then clustering inputs according to ranges of preferences.

Performer Control Patterns. Quality control of performers consists in deciding how to engage qualified workers for a given task and how to detect malicious or poorly performing workers. The most established patterns for performer control include: *a*) **QualificationQuestion** [4], at the beginning of a microtask, for assessing the workers expertise and deciding whether to accept his contribution or not. *b*) **GoldStandard**, [17] for both training and assessing worker's quality through a initial subtask whose answers are known (they belong to the so-called *gold truth*. *c*) **MajorityComparison**, [17] for assessing performers' quality against responses of the majority of other performers, when no gold truth is available.

6.3.2 Auxiliary Intra-Task Patterns

The above tasks can be assisted by auxiliary operations, performed before or after their executions, as shown in Figure 6.3. *Pre-processing steps* are in charge of assembling, re-shaping, or filtering the input data so to ease or optimise the main task. *Post-processing steps* is typically devoted to the refinement or transformation of the task outputs into their final form.

Examples of auxiliary patterns are: *a*) **PruningPattern** [56], consisting of applying simple preconditions on input data in order to reduce the number of evaluations to be performed. For instance, in a join task between sets of actors (where you want to identify the same person in two sets), classifying items by gender, so as to compared only pairs of the same gender. *b*) **TieBreakPattern** [56], used when a sorting task produces uncertain rankings (e.g. because of ties in the evaluated item scores); the post-processing includes an additional step that asks for an explicit comparison of the uncertainly ordered items.

6.3.3 Workflow Patterns

Very often, a single type of task does not suffice to attain the desired crowd business logic. For instance, with open-ended multimedia content creation and/or modification, it is difficult to assess the quality of a given answer, or to aggregate the output of several executions. A **Workflow Pattern** is a workflow of heterogeneous h

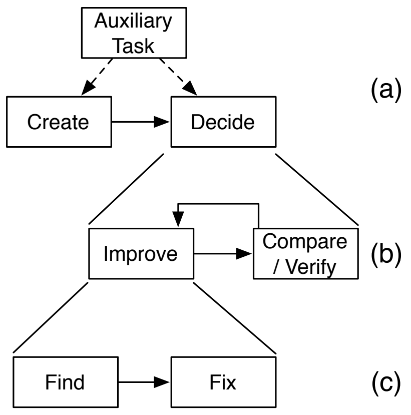


Figure 6.4: *Template for complex task patterns.*

crowdsourcing tasks with co-ordinated goals. Several workflow patterns defined in the literature are next reviewed; they are comparatively shown in Figure 6.4:

a) **Create/Decide** [54], shown in Figure 6.4(a), is a two-staged pattern where first workers create various options for new content, then a second group of workers vote for the best option. Note that the *create* step can include any type of basic task. This pattern can have several variants: for instance, with a stream data flow, the vote is typically restricted to the solutions which are produced faster, while with a batch data flow the second task operates on all the generated content, in order to pick the best option overall. *b*) **Improve/Compare** [53], shown in Figure 6.4(b), iterates on the decide step to progressively improve the result. In this pattern, a first pool of workers creates a first version of a content; upon this version, a second pool of workers creates an improved version, which is then compared, in a third task, to decide which one is the best (the original or the improved one). The improvement/compare cycle can be repeated until the improved solution is deemed as final. *c*) **Find/Fix/Verify** [11], shown in Figure 6.4(c), further decomposes the *improve* step, by splitting the task of finding potential improvements from the task of actually implementing them.

6.3.4 Auxiliary Workflow Patterns

Auxiliary tasks can be designed to support the creation and/or the decision tasks. They include: *a*) **AnswerBySuggestion** [52]: given a create operations as input, the provided solution can be achieved by asking suggestions from the crowd as follows. During each execution, a worker can choose one of two actions: it can either stop and submit the most likely answer, or it can create another job and receive another response to the task from another performer. The auxiliary suggestion task produces content that can be used by the original worker to complete or improve her answer. *b*) **ReviewSpotcheck** strengthens the decision step by means of a two-staged review process: an additional quality check is performed after the corrections and suggestions provided by the performers of the decision step. The revision step can be performed by the same performer of the decision step or by a different performer.

6.4 Workflow Execution

Starting from the declarative specification described in Sections 6.2 and 6.3, an automatic process generates task descriptors and their relations. Single tasks and their internal strategies and patterns are transformed into executable specification; the prototype supports all the intra-task patterns described in Section 6.3, through model transformations that generate the control marts and control rules for each task. Task interactions are implemented differently depending on whether interacting tasks are tightly coupled or loosely coupled.

Tightly coupled tasks share the control mart structure (and the respective data instances), thus coordination is implemented directly on data. Each task posts its own results and control values in the mart. Dependencies between tasks are transformed into rules that trigger the creation of new micro-tasks and their executions, upon production of new results by events of object or task closure.

Loosely coupled tasks have independent control marts, hence their interaction is more complex. Each task produces in output events such as `ClosedTask`, `ClosedObject`, `ClosedMicrotask`, `ClosedExecution`. I rely on an event based, publish-subscribe mechanism, which allows tasks to be notified by other tasks about some happening. Loosely coupled tasks do not rely on a shared data space, therefore events carry with them all the relevant associated pieces of information (e.g., a `ClosedObject` event carries the information about that object; a `ClosedTask` event carries the information about the closed objects of the task).

The workflow structure dictates how tasks subscribe to events of other tasks. Once a task is notified by an incoming event, the corresponding data is incorporated in its control mart by a-priori application of the data manipulation program, specified in the data manipulator stage of the task. Then, reactive processing takes place within the control mart of the task.

Modularity allows executability through model transformations which are separately applied to each task specification. Automatically generated rules and mart structures can be manually refined or enriched when non-standard behaviour is needed.

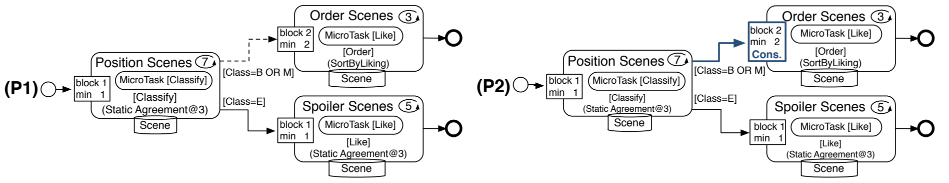


Figure 6.5: *Flow variants for the Positioning scenario.*

6.5 Experiments

I demonstrate various pattern-based workflow scenarios, defined using our model and method and deployed by using Crowdsearcher as design framework and Amazon Mechanical Turk as execution platform. I consider several scenes taken from popular movies, and we enrich them with crowd-sourced information regarding their position in the movie, whether the scene is a spoiler, and the presence of given actors in each scene. In the experiments reported here I considered the movie “The Lord of the Rings: the Fellowship of the Ring". 20 scenes were extracted and the groundtruth dataset created regarding temporal positioning and actors playing in the scenes. I compare cost and quality of executions for different workflow configurations.

Table 6.1: Scenario 1 (Positioning)*: number of evaluated objects, microtask executions, elapsed execution time, performers, and executinytions per performer (for each task and for each scenario configuration).*

	Position Scenes (payed $0.01)					**Order Scene (payed $0.01)**					**TOTAL**		
	#Obj	***#Exe***	***Time***	***#Perf***	***#Exe/Perf***	***#Obj***	***#Exe***	***Time***	***#Perf***	***#Exe/Perf***	**Time**	**Cost**	**#Perf**
P1	20	147	123	16	9.19	17	252	157	14	18.00	342	3.99$	26
P2	20	152	182	12	12.67	17	230	318	17	13.53	349	3.82$	26

Table 6.2: Scenario 2 (Actor)*: number of evaluated objects, microtask executions, elapsed execution time, performers, and executions per performer (for each task and for each scenario configuration).*

	Find Actors (payed $0.03)					**Validate Actors (payed $0.02)**					**TOTAL**		
	#Obj	***#Exe***	***Time***	***#Perf***	***#Exe/Perf***	***#Obj***	***#Exe***	***Time***	***#Perf***	***#Exe/Perf***	**Time**	**Cost**	**#Perf**
A1	20	100	120	18	5.56	–	–	–	–	–	120	3.00$	18
A2	20	100	128	10	10.00	–	–	–	–	–	128	3.00$	10
A3	20	100	123	14	7.15	20	21	154	10	2.10	159	3.42$	20
A4	20	100	132	10	10.00	41	19	157	9	2.10	164	3.38$	16
A5	20	100	126	13	7.69	69	60	242	17	3.53	257	4.20$	24
A6	66	336	778	56	6.00	311	201	821	50	4.02	855	14.10$	84

6.5.1 Scenario 1: Scene Positioning

The first scenario deals with extracting information about the temporal position of scenes in the movie and whether they can be considered as as spoilers. Two variants of the scenario have been tested, as shown in Figure 6.5: the task *Position Scenes* classifies each scene as belonging to the beginning, middle or ending part of the movie. If the scene belongs to the final part, I ask the crowd if it is a spoiler (*Spoiler Scenes* task); otherwise, I ask the crowd to order it with respect to the other scenes in the same class (*Order Scenes* task).

Tasks have been configured according to the following patterns:

- *Position Scene*: task and microtask types are both set as *Classify*, using a *StaticAgreement* pattern with threshold 3. Having 3 classes, a maximum number of 7 executions grants that one class will get at least 3 selections. Each microtask evaluates 1 scene.

- *Order Scene*: task type is *Order*, while microtask type is set as *Like*. Each microtask comprises two scenes of the same class. Using a *SortByLiking* pattern, I ask performers to select (Like) which scene comes first in the movie script. A rank aggregation pattern calculates the resulting total order upon task completion.

- *Spoiler Scene*: Task and microtask type both set as *Like*. A *StaticAgreement* pattern with threshold 3 (2 classes, maximum 5 executions) defines the consensus requirements. Each microtask evaluates 1 scene.

I experiment with two workflow configurations. The first (**P1**) defines a *batch* data flow between the *Position Scene* and *Order Scene* tasks, while the second configuration (**P2**) defines the same flow as *stream*. In both variants, the data flow between *Position Scene* and *Spoiler Scenes* is defined as *stream*.

The **P2** configuration features a dynamical task planning strategy for the the *Order Scenes* task, where the construction of the scene pairs to be compared in is performed every time a new object is made available by the *Position Scenes* task. A conservation policy in the *Order Scenes* data manipulator ensures that all the new scenes are combined with the one previously received.

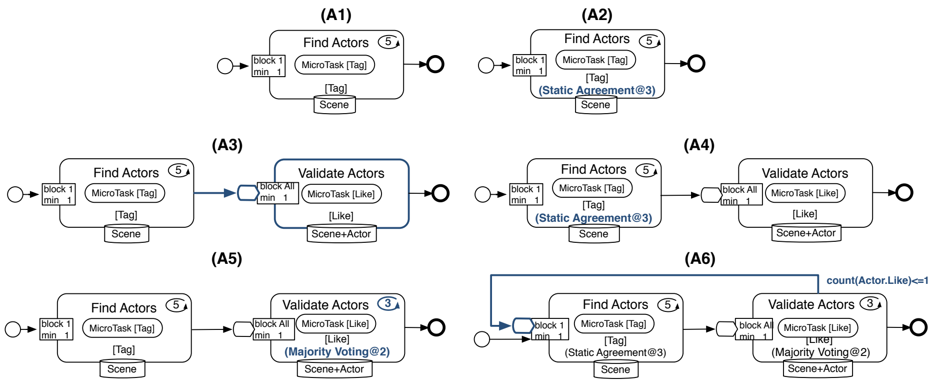


Figure 6.6: *Flow variants for the Actor scenario.*

6.5.2 Scenario 2: Actors

In the second scenario, I model a **create/decide** workflow pattern by asking the crowd to identify the actors that take part in the movie scenes; in *Find Actors*, performers indicate actors, in *Validate Actor* they confirm them. Tasks are designed as follows:

- *Find Actors*: Task and microtask types are set as *Tag*. Each microtask evaluates one scene; each scene is evaluated five times. Depending on the configuration, either no consensus pattern (**A1**, **A3**, **A5**) or a *StaticAgreement* pattern with threshold three (**A2**, **A4**, **A6**) is employed.

- *Validate Actors*: the task is preceded by a data manipulator function that transform the input Scene object and associated tags into a set of tuples $(Scene, Actor)$, which compose an object list subject to evaluation. In all configurations, microtasks are triggered if at least one object is available in the buffer. Note that each generated microtask features a different number of objects, according to the number of actors tagged in the corresponding scene. Configurations **A5** and **A6** features an additional *MajorityVoting* pattern to establish the final actor validation.

I tested this scenario with five workflow configurations, shown in Figure 6.6, and designed as follows:

- Configuration **A1** performs 5 executions and for each scene collects all the actors tagged at least once;

- Configuration **A2** performs 5 executions and for each scene collects all the actors tagged at least three times (StaticAgreement@3);

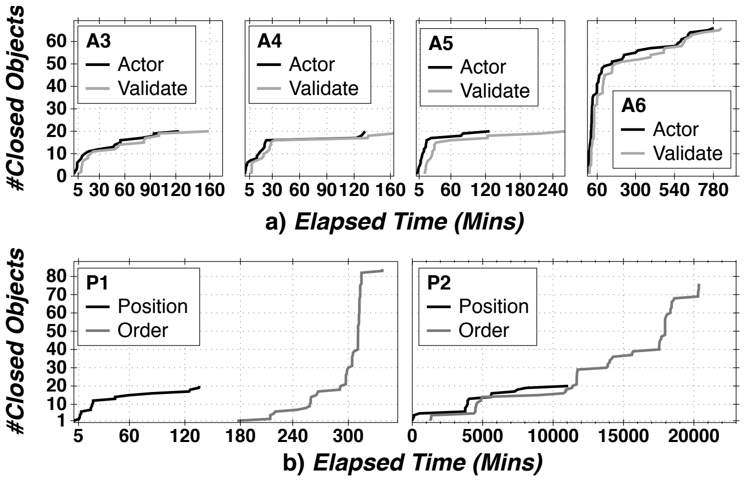


Figure 6.7: *Temporal distributions of closed objects.*

- Configuration **A3** adds the validation task to **A1**; the validation asks one performer to accept or reject the list of actors selected in the previous step;
- Configuration **A4** adds a validation task to **A3**, performed as in **A3**;
- Configuration **A5** is similar to **A3**, but the validation task is performed 3 times and a MajorityVoting@2 is applied for deciding whether to accept or not the object;
- Configuration **A6** extends **A5** by adding a StaticAgreement@3 on FindActors a feedback stream flow, originating from the *Validate Actors* task and directed to the *Find Actors* task, which notifies the latter about actors that were wrongly tagged in a scene (i.e., for which agreement on acceptance was not reached). Misjudged scenes are then re-planned for evaluation; for each scene, the whole process is configured to repeat until validation succeeds, or at most 4 re-evaluations are performed.

6.5.3 Results

I tested the performance of the described scenarios in a set of experiments performed on Amazon Mechanical. Table 6.1 and Table 6.2 summarise the experiment statistics for the two scenarios, 1700 HITS for a total cost of 39$.

Streaming Vs. Batch (Scenario 1: Positioning) In the first scenario I tested the impact on the application performance of the adoption of a stream data flow in a crowd workflow.

Time. Figure 6.7(b) shows the temporal distribution of closed objects for the **P1** and **P2** configurations. As expected, a stream flow (**P2**) allows for almost synchronous activation of the subsequent task in the flow, while batch scenario (**P1**) shows a strict sequential triggering of the second task. However, the overall duration of the workflow is not significantly affected by the change. While the first task of the flow behaves similarly in the two configurations, the second task runs significantly quicker in the batch flow, thus recovering the delay due to the sequential execution.

Quality. Table 6.3a shows the precision of the classification results of task *Position Scenes* (note that for this first part the two configurations are exactly the same, it makes no sense to compare the two results). Table 6.3b shows a measure of the quality of the obtained orders of scenes, i.e., Spearman's rank correlation coefficient of the resulting ranks from the *Order Scenes* task against the real order of scenes. Both tables show that the attained quality was not significantly influenced by the different task activation modes.

In summary, we didn't notice a different behaviour due to streaming. One possible reason is that in the batch configuration the entire set of assignments is posted at once on AMT, thus becoming more prominent in terms of number of available executions (and thus being preferred by performers, as widely studied [50]), while in a stream execution a small number of assignments is posted on AMT at every closing event of objects from the previous tasks.

Intra-Task Consensus Vs. Workflow Decision (Scenario 2: Actors) The second scenario aimed at verifying the impact that different intra-task and workflow patterns produced on the quality, execution time, and cost. We focused in particular on different validation techniques.

Time. Figure 6.7(a) and (c) shows the temporal distribution of closed object for configurations **A3-A6**. Configurations **A1** and **A2** are not reported because they are composed of one single task and thus their temporal distribution is not comparable. The temporal behaviour of the first and second tasks in the flow are rather similar (in the sense that the second one immediately follows the other).

Table 6.3: *Scenario 1 (Positioning), configuration P1 and P2: a) Precision of the* Position Scenes *classification task; b) Spearman's rank correlation coefficient of the resulting ranks from the* Order Scenes *task against the real order of scenes.*

(a)

Config.	**P Beg.**	**P Mid.**	**P End**
P1	0.50	1	0.11
P2	0.50	0.80	0.33

(b)

	Spearman Beg.	**Spearman Mid.**
P1	0.500	0.543
P2	0.900	0.517

Table 6.4: *Scenario 2 (Actor): Precision, Recall, and F-score of the 6 configurations.*

	A1	**A2**	**A3**	**A4**	**A5**	**A6**
Precision	0.79	1	0.92	0.99	0.95	0.89
Recall	0.98	0.87	0.97	0.90	1	0.96
F-Score	0.85	0.91	0.93	0.93	0.97	0.90

Validation is more delayed in **A5** due to the MajorityVoting pattern that postpones object close events. Configuration **A6** (Figure 6.7(c)) is significantly slower due to the feedback loop, which also generates a much higher cost of the campaign, as reported in Table 6.1. Indeed, due to the feedback, many tasks are executed several times before converging to validated results.

Quality. Table 6.4 reports the precision, recall and F-Score figures of the six configurations. The adoption of increasingly refined validation-based solutions (configurations **A3-A4-A5**) provides better results with respect to the baseline configuration **A1**, and also to the intra-task agreement based solution **A2**; validations do not have a negative impact in terms of execution times and costs. On the other hand, the complexity of of case **A6**, with the introduction of feedback, proved counter-productive, because the validation logic harmed the performance, both in monetary (much higher cost) and qualitative (lower results quality) senses, bringing as well overhead in terms of execution time. Notice that the configuration **A3** reaches the highest precision score. That's because the StaticAgreement strategy ensures that all the selected actors really appear in the image, while using the crowd for the validation part can add some errors (for instance some actors recognized in the *Find Actor* can be discarded in the *Validate Actors*). However note that the other configurations (**A3 - A5**) reach an higher recall and F-score value, meaning an overall better quality of the final result.

In summary, the above tests show an advantage of concentrating design efforts in defining better workflows, instead of just optimising intra-task validation mechanisms (based e.g. on majority or agreement), although overly complex configurations should be avoided.

6.6 Conclusions

I present a comprehensive approach to the modeling, design, and pattern-based specification of crowd-based workflows. I discuss how crowd-based tasks communicate by means of stream-based or batch data flows, and define the option between loose and tight coupling. I also discuss known patterns that are used to create crowd-based computations either within a task or between tasks and we show how the workflow model is translated into executable specifications which are based upon control data, reactive rules, and event-based notifications.

A set of experiments demonstrate the viability of the approach and show how the different choices in workfllow design may impact on the cost, time and quality of crowd-based activities.

CHAPTER 7

Users Engagement

7.1 Introduction

In this chapter I study the user engagement by observing their activities in a social challenges. While other works focused on monetary incentives I studied other type of means for engaging the crowd.

In particular I'm concerned with the social challenges which take place over one or more social networks and whose main purpose is to raise awareness or interest on a specific brand or event. The core of the challenge is a social game, played by visitors of the social networks. The success of the challenge is measured by the *social mobilization* generated by the game, e.g. in terms of number of accesses to the challenge resources or of positive actions (likes and follows) which are generated towards them.

Designing a social challenge is not obvious, as it requires not only to raise the initial interest of the participants, but also to keep it high throughout the lifetime of the challenge, and actually the main objective of the challenge organizers is to preserve or even increase the social mobilization more or less continuously throughout the planned lifetime, as success is typically measured not only by the global mobilization but also by a growing interest of players while the challenge is in progress. Therefore, the **challenge organizer** has to plan and schedule a number of actions which go beyond the publishing of the game, and use the typical mechanisms of social networks; the organizer may take the role of participants in order to solicit peer reactions, or may be helped by the participants who take the role of the organizer in further publishing the game, typically in order to mobilize participants who can help them in winning the challenge. While such actions are fundamental to boost the challenge, they must be used carefully, as an excess of their use might be seen as an intrusion in the game or an excess of advertisement that typically causes a loss of interest.

Participants to a challenge can be typically divided into two categories: **players** and **voters**. The former make specific actions, that we further classify as *producing content* or *enriching content*. The player's intent is to perform actions which are popular, where popularity is typically expressed by voters through a positive vote (using *likes* actions) and not adversed by a negative vote (using *dislikes* actions); in most cases, they play an active game in soliciting votes, thereby being the main mechanism for increasing the social mobilization. Winners of the challenge are determined by voters, through a guided process which is set up by the organizers and typically uses several intermediate stages.

The most interesting aspects in organizing social challenges is the possibility of deploying them over multiple social networks. The same social challenge can use many social platforms in co-ordination, typically adapting to the activities that are best supported natively by each social platform (e.g., a challenge can use a social network in order to collect the player's content and a different social network to collect the voter's likes.) Alternatively, different social challenges can reinforce each other by using cross-challenge and cross-networks reinforcement actions. Players themselves are active in many social networks and therefore can perform different activities upon each social network involved in a given challenge.

Another interesting aspect in organizing a social challenge is the careful use of external contributions to mobilization. We denote as *catalysts* any external organization capable of generating social mobilization. Typically, the catalyst's role is to announce or promote the challenge, or to reward to the player by means of tangible prizes or visible social actions.[1]

Given the above premises, the objective of this chapter is to *define and compare a number of strategies that can be used by organizers for increasing social mobilization, in a challenge which uses multiple social networks, over a planned lifetime, in the presence of players, voters, and catalysts.*

`YourExpo2015` was solicited to by the organizers of *Expo 2015* in order to increase the social awareness towards the *Expo 2015* event in Milano, which

[1]In this paper, we omit to consider the economic aspects connected to managing a challenge and we focus just on how to raise social mobilization; of course, as the challenge's objective is to raise the interest towards a brand, it has to be compared to other mechanisms - such as plain advertising - in terms of cost/effectiveness; the cost of catalyst involvement is just an element of such trade-offs.

opened on May 1st 2015; the game consists of a social production of content (on *Instagram*) and voting (on both *Instagram* and *Facebook*), with some awareness creation activities performed also on *Twitter*. The challenge consists of twelve rounds of separate contexts, where each context consists of producing and voting photos describing specific pairs of topical tags inspired by Expo 2015; the challenge has been running since Dec. 15, 2014 and will run until March 15, 2015. Observations of the first nine weeks of the challenge allow us to draw interesting conclusions on the efficacy of the various actions that we performed (as organizers) or induced (by using voters and catalysts).

The paper is chapter as follows. Section 7.2 introduces the dimensions and primitives for the engagement strategies and discusses how such engagement can be measured. Section 7.3 presents our implementation architecture, Section 7.4 presents the `YourExpo2015` challenge and provides several diagrams which illustrate the effect of our actions on social participation and Section 7.5 concludes.

7.2 Instruments for Building Social Challenges

In this section, I analyze the strategies and instruments that are in the organizers' availability for building social challenges. I observe that social engagement depends on a number of factors (including the time of the day when activities occur); continuous monitoring of player's actions and careful planning of reactions (with a correct level of repetition) can effectively enhance social engagement.

7.2.1 Roles

A social challenge requires the interplay of four kinds of actors.

- **Organizer.** Sets the rules of the game, typically encoded in the *game regulations*, and then performs the activities which are prescribed by such rules. In addition, the organizer performs activities targeted to enhancing social participation; these activities normally favor the visibility of the top players, but they should not alter the outcome of the game.

- **Player.** Autonomously decides to perform actions in the game. Players essentially aim at obtaining visibility for the actions that they perform, granted either by other members of the social network or by the organizers of the challenge (or by both of them).

- **Participant.** Decides the outcome of the game by voting on the content shared or enriched within the game or on the players.

- **Catalyst.** Catalysts represent external entities which support the organizers in boosting the social participation; their actions may or may not be under

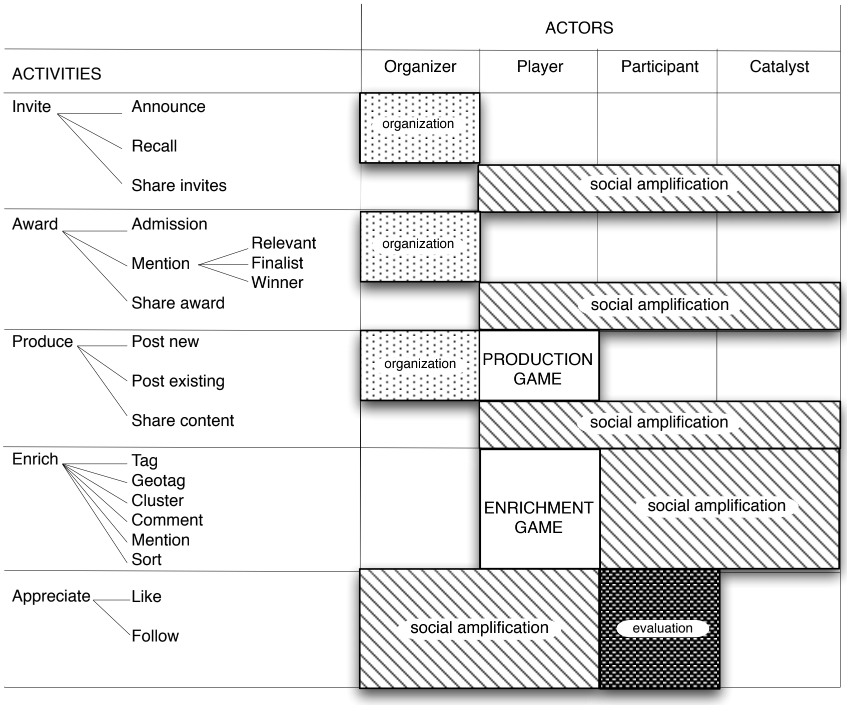


Figure 7.1: *Actors and activities interplay in a social network challenge*

the control of the organizers, and at times have consequences which are hard to predict. Catalysts include popular social accounts (actors, celebrities, institutions, well known brands), or real world events that have a strong impact on the public (such as TV programs, commercial advertising, endorsement by government or large companies).

7.2.2 Actions

I define a set of actions and I characterize them in terms of expected reactions and impact on the public; effort involved in the action and possible level of automation that can be applied; and applicability on the different social networking platforms.

Concrete actions that can be performed on the social network are grouped based on the purpose they have. I identify 5 main purposes in a social game:

- **Invite.** The purpose is to make the game known on the social network and to convince potential players to participate. The concrete actions that can be performed in order to obtain this are:
  - **Announce**, i.e., explicitly declare the start of the challenge, through posting of rules of engagement, deadlines, or aim of the game.

  - **Recall**, i.e., sending our reminders and repetitive messages about the topics, the game rules, and the duration of the challenge.
  - **Share invites**, i.e., endorsing, sharing with contacts or republishing an announcement or recall of the challenge.

  These actions may imply publishing of direct descriptions of the game, or also examples of participant content, or evocative content for the topic or focus of the game.

- **Award.** The purpose is to select a subset of players for awarding them some special prize or enabling them for the next level of participation or selection. This can be obtained through:
  - **Admission**, i.e., the selection of potential players that are therefore allowed to participate to the game. This is relevant when some kind of moderation of the contents is needed.
  - **Mention**, i.e., the selection of a piece of content (or its author) because of its quality, which makes it a relevant candidate for becoming a winner of the challenge. This is usually done through multiple level selection of participants, e.g., initial selection of **nominees**, then selection of **finalists**, and finally selection of **winners**.
  - **Share awards**, i.e., sharing the mentions of winners on the social network. This is done either by the person that has won the award, or by any other participant that wants to congratulate or acknowledge the quality of the winner.

- **Produce.** The purpose is to generate content within the gaming platform. The options are:
  - **Post new**, i.e., the player posts original content (photos, videos or text) produced by himself to the challenge, specifically devised for the challenge and prepared during the challenge period.
  - **Post existing**, i.e., the player posts some content that was pre-existing to the challenge. This entails both content produced by other people (e.g., photos or videos found on the web) or content produced by the player himself for other reasons (typically in the past) and reused for the challenge.
  - **Share content**, i.e., various actors share the content of the challenge with their social network, for various reasons (including: making one's own content more popular or distributing valuable content of others).

- **Enrich.** The purpose is to expand the description or media materials of a piece of content participating in the game.
  - Tag, i.e., adding tags to the content.
  - Geo-tag, i.e., adding a geo-location to the content.
  - Cluster, i.e., collecting similar contents in clusters.
  - Comment, i.e., adding textual comments to the content.
  - Mention, i.e., mentioning people or, more generally, entities related to or appearing in the content.
  - Sort, i.e., listing the contents in some specific order or ranking.

- **Appreciate.** The purpose is to express and share appreciation for the activities or content of a participant, or for the participant himself.
  - **Like**, i.e., annotating the content with a “like" or “preference" tag (typical in social networks like Facebook, Instagram or Twitter).
  - **Follow**, i.e., declaring interest in a person and following his activities.

7.2.3 Actor/Actions Interplay

The interplay between actors and actions is shown in Fig. 7.1. I envisage two categories of challenges based on content:

- **Creational Games.** The player generates content (e.g.: photos, videos, text, tweets, links or others), according to the *Produce* objective of the activities, i.e., by *posting new or pre-existing content*, as described in Section 7.2.2.

- **Enrichment Games.** The player enriches content which is generated by the organizer (or possibly by other actors), according to the *Enrich* objective described in Section 7.2.2.

The main role of **organizers** is to produce the invitations and to manage awards. Shares of invitations and of awards is typically performed by catalysts, but it can also be performed by players with the objective of enhancing the awareness of their presence in the challenge. It can also sporadically be performed by participants.

The main role of **players** to creational games is to produce posts, either of new or of existing content; such content is also shared by players, catalysts, and (to a less extent) participants to amplify the social participation. The main role of players to enrichment games is to enrich posts. Such enrichments can also be socially amplified by catalysts and participants.

Finally, the main role of **participants** is to appreciate the actions of players, by following them and liking their actions, thereby performing the evaluation. Actions of liking and follow can tactically be performed by organizers and by players to enhance the social participation. Typically, catalysts don't engage directly into appreciation actions, because catalysts role must appear as neutral, and their size and visibility is incomparable to those of players.

7.2.4 Other Aspects that Influence Engagement

Aspects such as the challenge's staging, timing and multiplatform execution should be also considered in organizing a challenge.

Staging.

A challenge can be a single-shot event, where players make their actions and participants vote within a short time interval. However, an important aspect of social challenges is to build **fidelization**, which occurs when the players and participants become acquainted with the game and perform multiple actions. Fidelization can only occur when the game is **staged**, i.e. it is structured in a way that allows a player to anticipate the game progression and engage in a multi-action participation. Staging can be obtained by:

- **Repeating the challenge periodically**, with a winner for each period.
- **Sub–structuring the challenge into phases**, and giving to the player different task to perform at each stage.

Influence of daytime.

As shown in Section 7.2.3, organizer's actions and their social amplification occurs in response to players' actions. However, the effect of such actions is extremely different depending on the time at which the action is performed and also of the particular day (e.g., workday/weekend/vacation). Time-dependent effects must be studied for each challenge, e.g. certain challenges may attract higher participation during nighttime or weekends.

Cross-social Network Fertilization.

Cross-network fertilization is made possible because most actors and participants are part of multiple social networks at the same time (e.g., they share content on Instagram and Twitter, and have friends on Facebook and collaborators on LinkedIn). Thus, it is possible to influence their behavior on one social network through actions that occur on a different social networks.

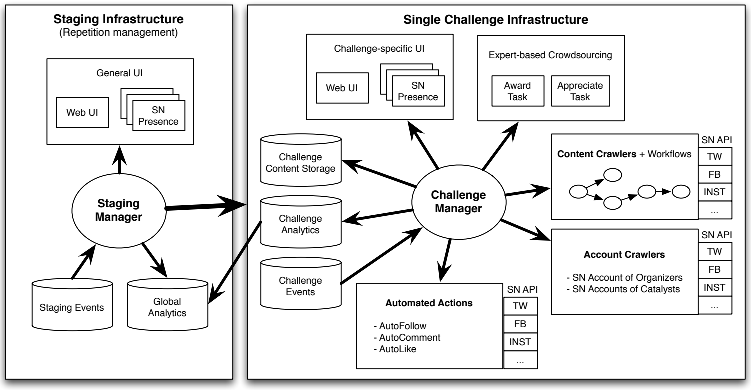


Figure 7.2: *Architecture of the system*

7.2.5 Sensing the Social Activities

Challenges must be effectively monitored throughout their execution for understanding users' behavior, mapping it to specific aspects of the challenge, and deciding as a consequence the best ways to further boost individual or collective engagement. I distinguish different levels of monitoring:

- **Native Sensing**. This level exploits basic quantitative analytics that are natively provided by social network platforms. This includes count of contributions, their re-shares and likes or preferences, and (for some platforms like Facebook) their global reach.

- **Deep Sensing**. This level entails deep monitoring of the event occurrences and user interactions through detail measures that are not natively provided by the social network platforms; an example is the extraction of timed events (e.g., likes on Facebook and on Instagram) for which the social network only provides global counts. Deep sensing occurs by means of periodical and/or adaptive invocation of social network APIs.

Through both kinds of sensing, I build aggregated popularity measures of content and I understand and classify the behavior of players in terms of amount of activity, continuity of the actions in time, reactivity to solicitation from organizers, willingness to share challenge content and messages of the challenge, and extent of his social amplification. Similar analysis can be performed over individual participants and catalysts.

7.3 Architecture and Implementation

I designed and implemented an architecture for managing staged social challenges, shown in Fig. 7.2. The architecture is layered, with an external layer, called **stag-**

ing infrastructure, which generates several versions of the internal layer, called **single challenge infrastructure**. They both are associated with several resources dedicated to user interaction. Normally, these include a **Web Interface** which publishes the rules of the challenge, and several **social network pages**, describing the specific portion of the challenge which takes place on each social network. These resources may be present only at the stage level, if each stage is managed as an independent challenge. The **staging manager** holds information about the series of events which are associated with each stage, and in addition it manages global data analytics obtained as the summary of the various stages.

The organization of the core of challenges is performed by the **challenge manager**. It instantiates and manages several crawlers belonging to predefined classes, that are addressed to specific social networks and perform specific tasks, which are either **content-specific actions** or **account monitoring** on each social networks; some of them are **automatic actions** generated in response to the task's output (e.g., following accounts or liking contents). Some of the content-specific tasks need to be organized through mini-workflows (e.g., the monitoring of *likes* given to a specific content must follow the *post* of that content.)

Crawlers activities are timed so as to respect the constraints on API usage which are imposed by each social network; content is accessed through the given challenge's **hashtag**, generated by the staging infrastructure and used as parameter in the calls to the social network APIs. The data collected by crawlers are stored into stage-specifics data analytics; finalists and winners can be automatically determined by the system based upon such analytics, but in many cases it is useful to provide some crowd-based control of the outcome (e.g., inappropriate content should be discarded). Such activity is controlled by a crowd-based task which is executed by experts on behalf of the organizer.

7.4 YourExpo2015

Expo 2015 is the Universal Exhibition hosted in Milano from May 1 to October 31, 2015. Over this six-month period, more than 140 participating countries will run their own exhibitions around the theme of guaranteeing healthy, safe and sufficient food for everyone; Expo 2015 expects over 20 million visitors to its 1.1 million square meters of exhibition area. During the period which precedes the opening, Expo 2015 has created a number of marketing campaigns on traditional and social channels. In such framework, I was able to perform an experiment for increasing the Expo 2015 brand awareness through the development of a photo challenge, called `YourExpo2015`, that has been independently managed by me and my colleagues, but has benefited of some interaction with the large social channels organized by Expo 2015; these can be considered as *catalyst activities* according to the scheme of the previous section.

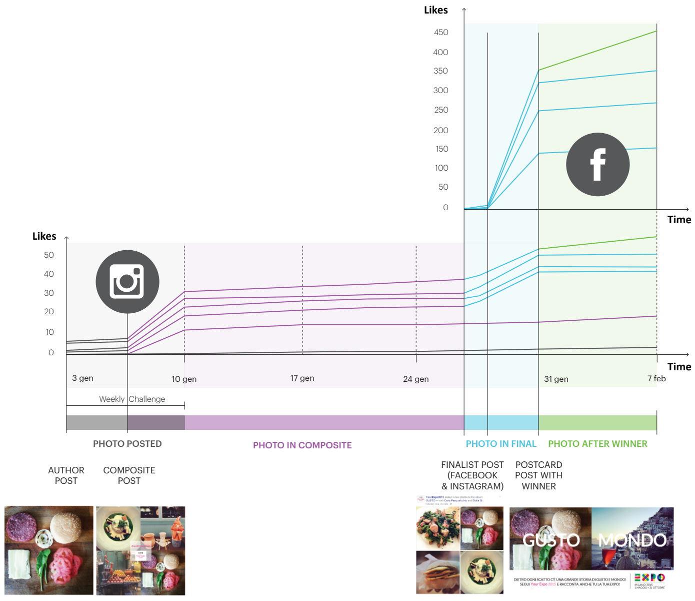


Figure 7.3: *Phases of YourExpo2015, a multi-platform challenge*

The game `YourExpo2015` is based on the social production of photos on *Instagram*, in response to specific hashtags which are generated by Expo 2015. The purpose of the challenge is twofold: to engage users and increase visibility of the brand and initiative of Expo 2015, as well as to collect and disseminate relevant content associated to the exhibition, so as to increase awareness on the topics and purposes of Expo 2015. Hashtags are paired in a way that hints to a contrast (e.g. `Fast/Slow`, `Art/Fun`, `Land/Sea`, and so on). Most photos show food, but they can be on arbitrary subjects. The best two photos for each of the two hashtags, separately selected, compose a **postcard** where photo and hashtag are shown together (see the last post in Fig. 7.3); as the main reward of the challenge, Expo 2015 promotes the postcards on its social media[2]. The challenge has been running since Dec. 7, 2014 and will run until March 7, 2015, with one pair of hashtag published every week, for twelve weeks; photos can be posted in the week following the challenge announcement; thus, each week is associated with is a different stage, each with its own set of competing photos. Instagram is used for the initial posting of photos and for the selection of finalists; Facebook is used only after Jan. 20, for the selection of the winner after the posting of finalists. We deliberately evolved the format of the challenge from week to week, allowing

[2]On 2/22/2015, the Facebook page `Expo2015Milano.it` header shows the postcard of the two winning photos of the `Terra/Mare` challenge.

us to try different actions and to monitor their effect.

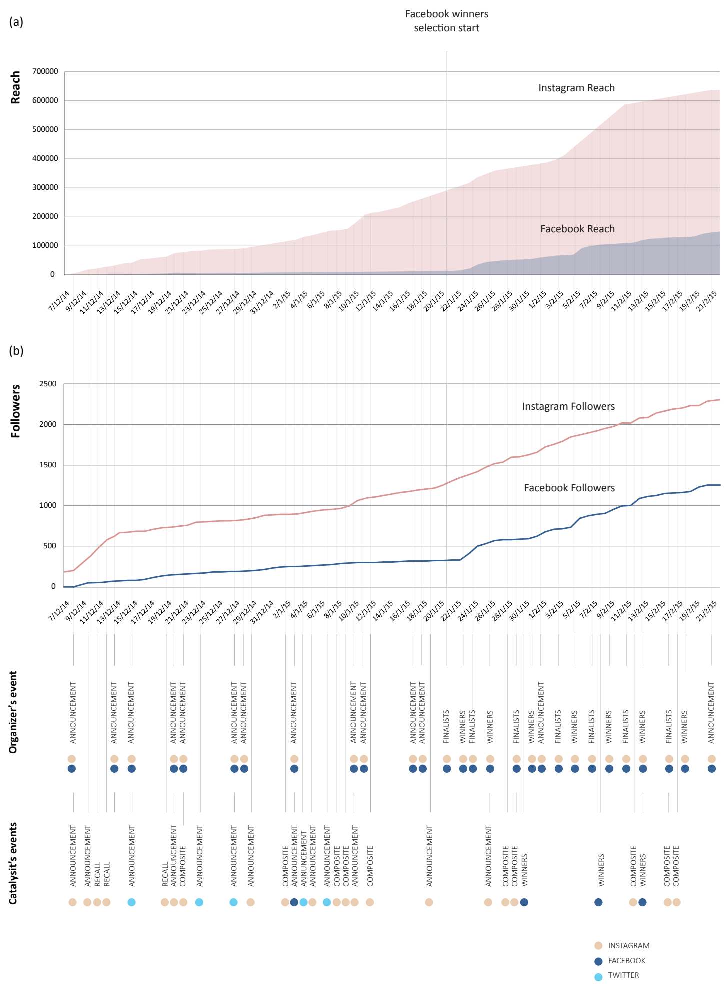


Figure 7.4: *Cumulative impact of the YourExpo challenge before Feb. 21, 2015*

The votes accumulated by a few players are illustrated in Fig. 7.3; we trace the history of votes (likes) expressed upon photos which were posted in response to the hashtag published on Dec. 27, from the initial posting of photos to the definition of the winner[3]. A few days after the start of the stage, we repost the most voted photos in a format, named **composite**, that consists of four photos (see the second

[3]The trace is a close description of real events but is not a reading of real data, postponed to the next section.

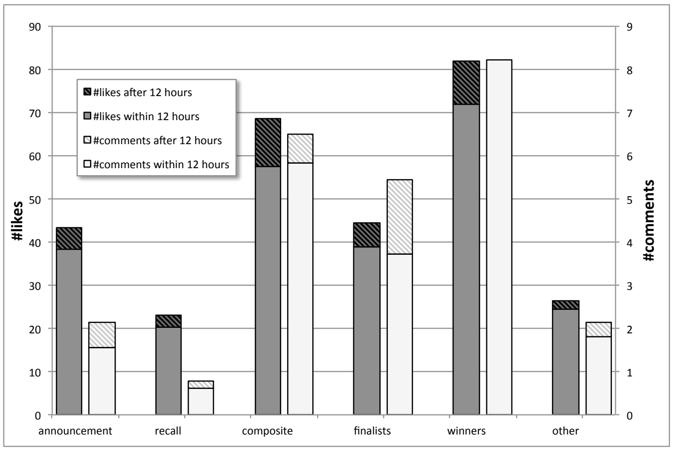


Figure 7.5: *Histogram showing the number of likes and comments received by the different types of our posts on Instagram*

and third posts of Fig. 7.3); in such selection, we ignore the votes (likes) collected by photos before their posting. The publication of composites starts a first round of promotion of the authors, yielding to a growth of their votes on Instagram. After about three weeks, we select the four photos with higher number of votes as finalist, and we publish the finalists challenge, this time on Facebook; the Facebook challenge was started on Jan. 27, and lasted 3 days. Initially, finalist publication has a low reaction, but with a small delay we republish finalists on Instagram; at this time, the finalists (whose identity on Facebook is not known to us) come to know about the selection, and they start a second round of promotion through their friends, this time on Facebook, thereby performing cross-platform engagement.

Each round of promotion corresponds to targeted re-posting, which is not known to us (thus, we cannot evaluate its impact in terms of additional reach); we see a significant growth of the votes of finalist photos on Facebook, which we take as the consequence of promotion activities. Eventually, one winner is selected (by counting its Facebook votes); this event boosts the likes on the specific photo as effect of its increased visibility.

7.4.1 Global View

Figure 7.4 shows the global impact of the challenge, which caused more than 600000 actions (post, like, comment) on Instagram and about 150000 contacts on Facebook, with more than 2000 followers on Instagram and more than 1000 followers on Facebook. The figure shows the main actions performed by the organizers (challenge announcements, finalists and winners) and all the posts made by Expo 2015 on their official social media accounts.

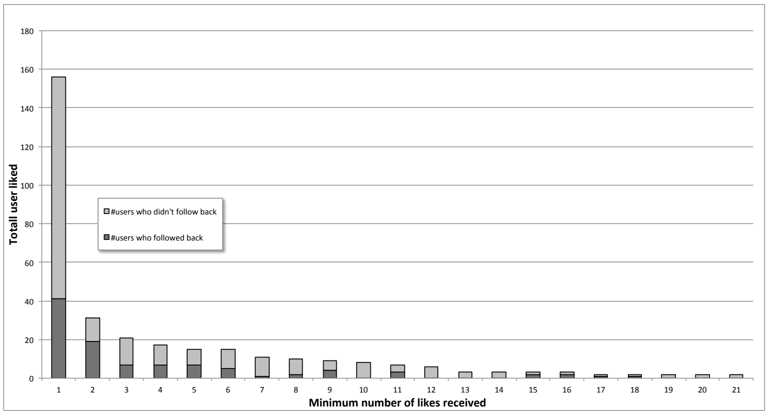


Figure 7.6: *Histogram showing the number of users who followed back versus the total number of like received*

This global view can also be given in terms of specific reactions to our activities as organizers. Figure 7.5 shows the global number of likes and comments to our posts on Instagram; the diagram shows that composites got the maximum of reactions, specifically when announcing the winner of the challenge. Most of reactions take place in the 12 hours immediately following the posts.

An interesting feature of our challenge architecture is the ability to automatically perform actions in response to the activities of players, that we monitor through crawlers. In particular, during three weeks we set automatic likes to every post, and we inspected the player's response to such automatic *likes* in the form of a *follow (back)* action; in Figure 7.6 the reaction is plotted as a function of the total number of posts made by each player. It turns out that the first and especially the second *like* got the strongest reaction (respectively with around 25% and 60% of follow backs), while subsequent *likes* were less effective. The diagrams also shows that a few players posted a large number of photos.

Table 7.1: *Statistical significance of post classes based on number of likes*

	win.	comp.	finalists	announc.	other	recall
winnners	-	NO	$p<0.001$	$p<0.01$	$p<0.001$	$p<0.01$
composite		-	$p<0.01$	$p<0.01$	$p<0.001$	$p<0.01$
finalists			-	NO	NO	$p<0.01$
announc.				-	$p<0.05$	$p<0.001$
other					-	NO
recall						-

In order to verify that different types of posts generate different amounts of interaction, we performed the *t*-test on the classes, comparing the distribution of likes and comments of each groups. The null hypothesis is that the different type

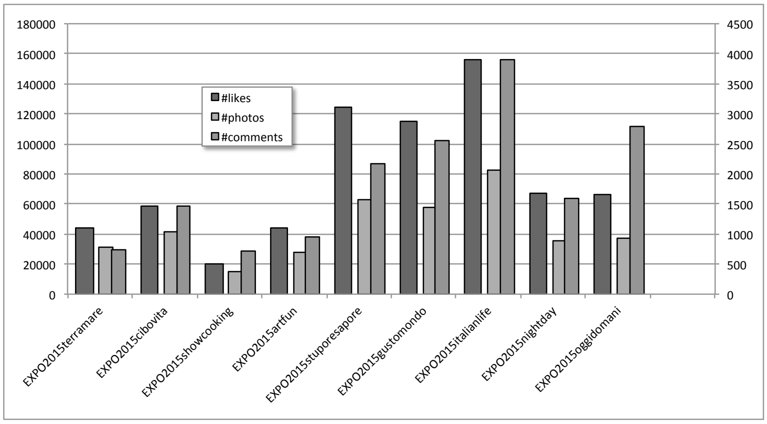


Figure 7.7: *Total number of likes, photos and comments for each stage of the challenge (associated with the week's hashtag)*

Table 7.2: *Statistical significance of post classes based on number of comments*

	win.	comp.	finalists	announc.	other	recall
winnners	-	NO	NO	NO	NO	NO
composite		-	NO	NO	p<0.05	p<0.05
finalists			-	NO	NO	NO
announc.				-	NO	NO
other					-	NO
recall						-

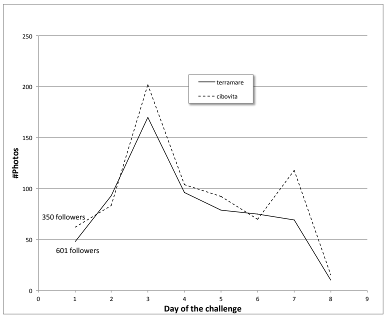
of posts generate the same amount of interaction. In particular we used the Welch's t-test, since the limited non-normality of our data and because the assumption of homogeneity of variances failed. We also applied the Holm-Bonferroni correction for multiple tests. Table 7.1 show the results of the *t*-test using the number of likes as feature. The *p*-value is acceptable for most of the comparisons.

Table 7.2 shows the results of the test using the number of comments as feature. Instead in this case, the *t*-test fails in most of the cases, so we can not conclude that the different types of post have impacts on the number of comments received.

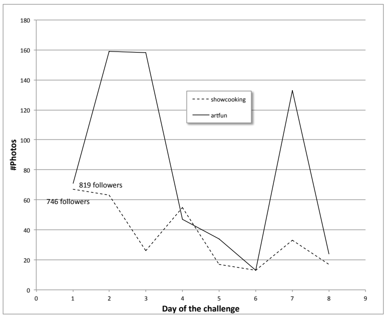
7.4.2 Staging

Figure 7.7 shows the distribution of likes, comments, and photo posts on Instagram, divided in the nine stages of the challenge. We cluster the nine stages into four periods of the challenges:

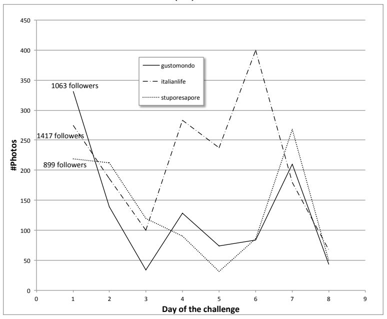
- The **initial period**, when people came to know about the challenge.
- The **vacation period**, during Christmas vacation, when all the activities had a reduction.



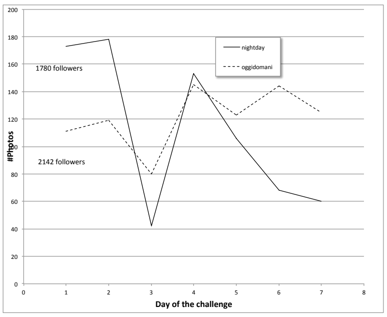
(a)



(b)



(c)



(d)

Figure 7.8: *Graph showing the number of photo posted during the week for each stage of the challenge*

- The **full-engagement period**, with carefully planned catalyst actions.

- The **low-engagement period**, with few catalyst actions on Instagram (and more focus on Facebook).

The number of photos posted for each stage during the first week (when the photo posting is open) shows some differences in the four periods. Two initial weeks are represented in Fig. 7.8(a), with few followers and thus few players' posts, typically boosted by catalyst actions on day 3. Two vacation weeks are represented in Fig. 7.8(b), where day 6 reports lower activity on Christmas and New Year's Eve. Three full-engagement weeks are represented in Fig. 7.8(c), where we note that players responded immediately to the challenge announcement of our own account (thanks to the presence of several followers), and then there responded with peaks of activities on days 4, 6 and 7 to catalyst's actions. Finally, two low-engagement periods are represented in Fig. 7.8(d), where interest moved to Facebook (and catalyst actions reduced).

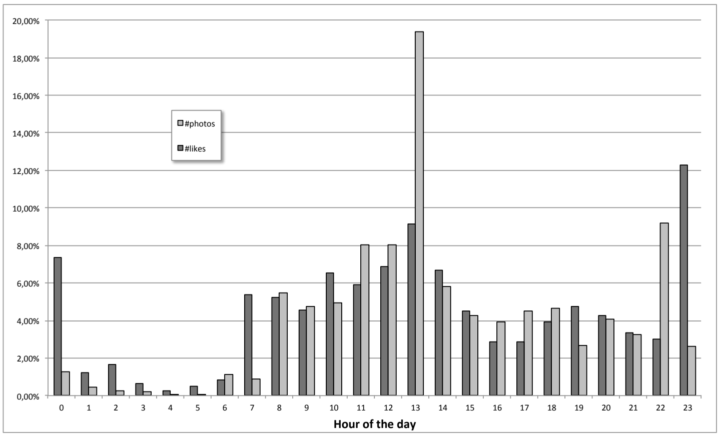


Figure 7.9: *Distribution of like and of photo postings in the hours of the day*

7.4.3 Time dependency

Actions of players and participants are heavily influenced by the daytime. Fig. 7.9 shows the distribution of likes and photo postings during the day; note the peaks of posting at 1PM and 10PM, and high commenting activity at 11PM and 12PM.

We also organized an automatic *follow* action to players' posts of photos, and we monitored the player's responses to our action; Fig. 7.10 shows such responses, relative to the time of the day when our automatic *follow* action was issued (we report the first reaction of users within twelve hours). We note that the reactions to our early morning *follows* are limited to few *likes*, then users' reactions rise in quantity and quality, reaching a maximum of responses at 11AM, when 8 out of 30 players decide to *post* another photo. One can also note that, at all hours, most reactions occur within the first two-three hours.

7.4.4 Individual variability

Finally, Fig. 7.11 analyzes the votes of some winners, and show that they exhibit strong differences. Votes are collected both on Instagram and Facebook; the intermediate vertical line indicates the switch from one platform to another, and voting time is normalized (as number of hours since the starting time of the challenge). Notice that photos start to collect votes from the moment they enter the challenge (not necessarily at the beginning). Furthermore, some photos enter the challenge as content posted in the past, now tagged with the challenge tag. This implies they enter with a number of likes already in place. The figure shows a winner capable of collecting about 800 votes on Instagram and much fewer votes on Facebook; and conversely, another winner with very few votes on Instagram who was able to collect more than 600 votes on Facebook; other winners exhibit a more coherent trend within the two platforms.

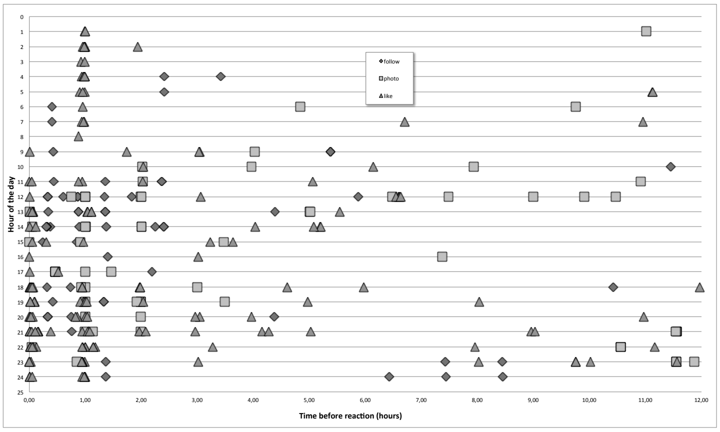


Figure 7.10: *Reaction of the users who have been followed based on the day hour*

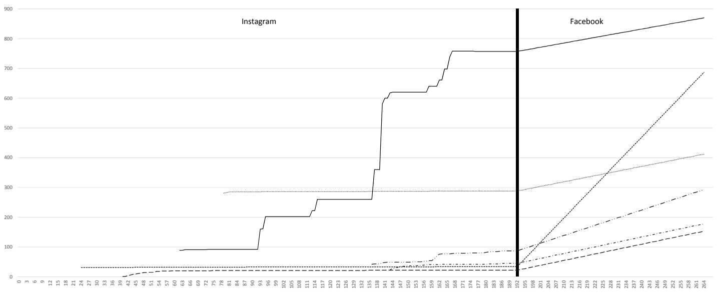


Figure 7.11: *Story of the winners on Instagram and Facebook*

7.4.5 Cluster Analysis of Players

We applied cluster analysis to determine how users are involved in the challenge, either as players or as voters; to evaluate their behavior, we selected the number of likes and number of posted photos as dimensions. We first removed the outliers that were qualitatively detected as the account of the organizer (YourExpo2015) and all the users that liked only one image and never posted a photo (100k over 160k users), then we run the k-mean cluster algorithms, which produced 7 clusters, shown in Fig. 7.12. Clusters 7 and 1 group users who specialized as players and voters respectively; then, clusters 6 and 4 group users who were less active but still specialized. The other clusters include users which were active in both roles, with decreasing activity going from cluster 3 to cluster 2 and 5. Clusters 6 and 7 identify the most active players (158 users), while cluster 5 represents the least active users (57504 users). Based on the objectives of the challenge, one cluster can bring better results than another. For instance if it's a content creation challenge, the

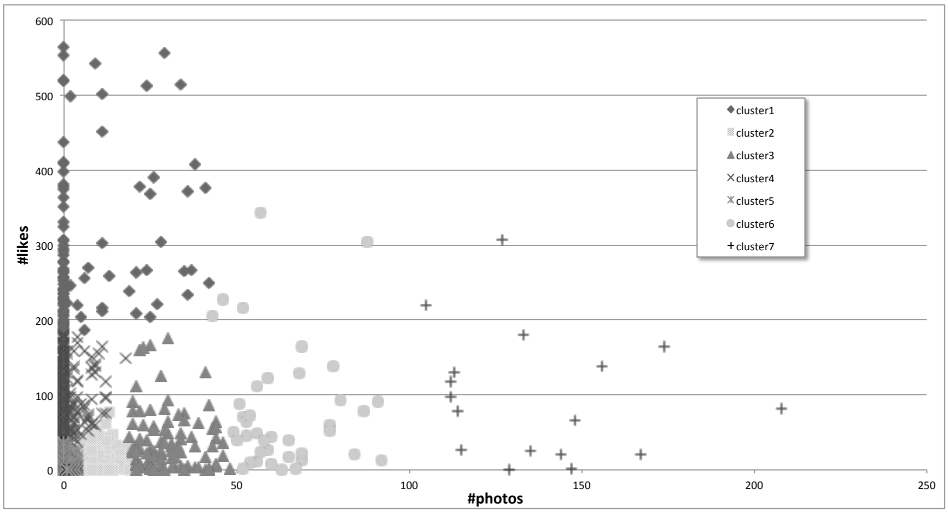


Figure 7.12: *Clusters of users*

Organizer should leverage cluster 7 and 6. While if it's a enriching game, it's better to target people belonging to clusters 1 and 4.

7.5 Conclusions

In conclusion, I report a few lessons learned: first of all, management of Expo2015 was satisfied with results: the experience was at risk of being negatively received. In general, players reported to us that they were pleased by the management of the challenge, including its fairness. It is important to use social networks at their best, in our experience we used Instagram for posting and Facebook for final voting, while our attempts to propagate attention using Twitter were less successful. The timing of the day must be considered for any kind of action, including the automatic ones which directly target specific users (likes, follows), which are useless/negative if posted at times when people don't react (including vacation or holiday time, where reactions become unpredictable).

I also understood that the mere announcements of challenges and call to actions fail to engage people, but active management pays. In particular, mentioning players is much more viral, especially when several of them are mentioned together, because this action stirs interactions among them, even if they didn't know each other in advance. The support of popular catalysts is crucial, although it is typically not under full control of the organizer; through a careful study of few, well-calibrated actions, we managed to organize regular and timely catalyst's contributions.

With this approach, the players to the online challenge constitute a large body of executors of classic crowd sourcing tasks, such as content creation or tagging. This study demonstrates that social platforms can be used for effective content generation, and that a large and rather continuous players' participation can be stimulated by well-designed non-monetary rewards. I conclude that: (a) Although players want to win, the development of social relationship is perhaps an even stronger driver, therefore organizers should spend lots of efforts in creating mutual engagement. In my experiments, composite mentioning of several players created a sub-network of participants who positively interacted, both by contributing content and by mobilizing voters. This constitutes a notable difference with respect to traditional crowdsourcing platforms, where instead workers are not engaged by actions or interactions with others. (b) Visibility provided by catalysts is fundamental to boost the challenge and to keep it alive; assuming that catalyst actions are scarce / expensive resources, they must be programmed in a way that provides maximum effect upon the players. (c) It is quite important to produce regular challenges, e.g. through their periodic staging, so that many players can repeat their actions several times (e.g., several players posted tens of photos); repetition and long duration also helps in growing a large audience; at the same time, it is important to be fully aware of temporal factors, e.g. daytime or festivities, in order to properly plan automatic and catalyst actions that may engage new players.

CHAPTER 8

CrowdSearcher

8.1 Introduction

In this chapter I describe the prototype that has been developed during this thesis. Crowdsearcher is a platform for crowd management written in Javascript running on Node.js server; this is a full-fledged event-based system, which fits the need of the rule-based approach described in the previous chapters. It offers a plug-in environment to transparently interface with social networks and crowdsourcing platforms. It implements the model and the process defined in Chapter 3 allowing to easily define multi-platform crowd-based applications through step-by-step specifications, where the application is initially configured and then automatically generated. In the remainder of this chapter I describe in details the functionality of the tool by showing a step-by-step tutorial on how to configure a crowdsourcing task.

The chapter is structured as follow: Section 8.2 describe the design process, from the selection of the operation that need to be performed, to the specification of the UI, Section 8.5 describes the architecture, and Section 8.6 lists the real world scenario in which this tools was used.

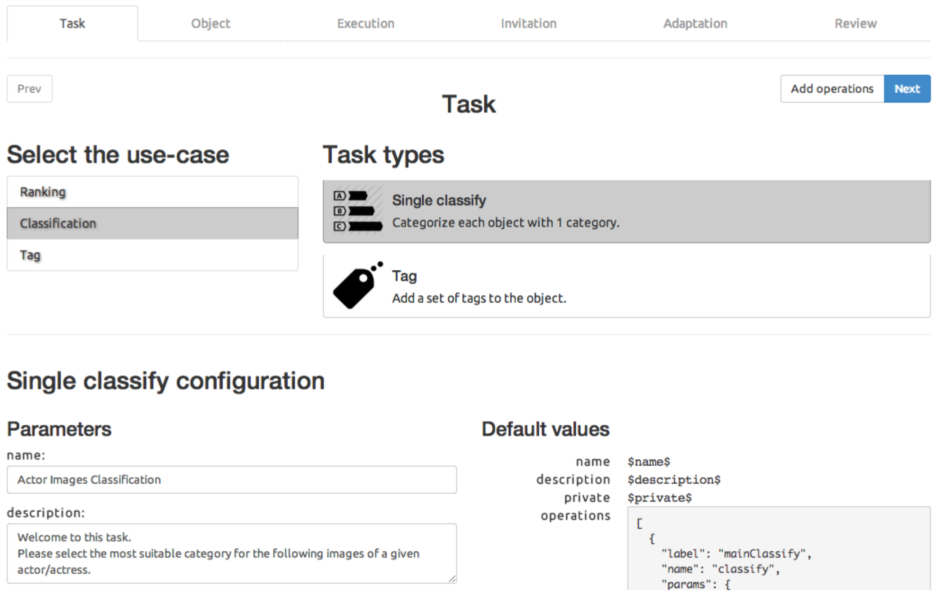
A running demo and other resources can be found online at: `http://demo.search-computing.com/cs-demo/`.

A short video summarizing the demo steps is available at: `http://youtu.be/9ZAQCqAfwzA`.

8.2 Task Configuration

The crowdsourcing task configuration is covered by a 7-step wizard that guides the task designer in the creation of a task. Each step is described in details in the following subsections.

8.2.1 Task Design



Single classify
Categorize each object with 1 category.

Tag
Add a set of tags to the object.

Single classify configuration

Parameters

name:
Actor Images Classification

description:
Welcome to this task.
Please select the most suitable category for the following images of a given actor/actress.

Default values

```
       name  $name$
description  $description$
    private  $private$
 operations  [
               {
                 "label": "mainClassify",
                 "name": "classify",
                 "params": {
```

Figure 8.1: *Task design step.*

The Task Design step consists in the selection of the task types that will be performed in the task. These task types are selected from an abstract model described in Chapter 3, crafted after a careful analysis of the systems for human task executions and of many applications and case studies.
As shown in Figure 8.1, the user first selects which type of problem he wants to solve (e.g., a classification problem, ranking problem, and so on), then he chooses the task type he wants to use. After selecting a task type, the tool allows to fully configure the task by inserting a title, a description (it supports the markdown notation[1]) and setting various parameters that depend on the chosen task type. Some

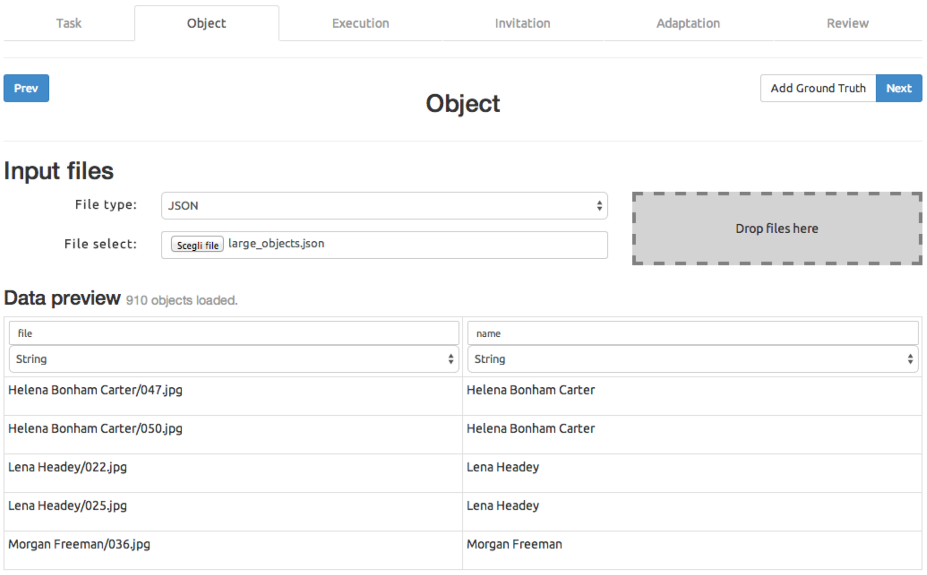
[1]http://daringfireball.net/projects/markdown/

Task type	Description
Classify	Categorize an object by selecting one category
Tag	Add one or more of tags to an object
Comment	Write a comment related to the object
Hot or Not	Compare two objects
Like	Rate an object by giving a like
Score	Rate an object by giving a score

Table 8.1: *Task types supported by the system.*

examples are: number of objects per microtask, level of agreement between multiple performers, an the list of the categories to be used in a classification task. Each task type come with a set of configuration parameters and an underlying set of control rules (the actual implementation of the ones described in Chapter 4) that handle the aggregation of the results. Then the system allows to add other secondary operations.

8.2.2 Object Design



File type: JSON

File select: large_objects.json

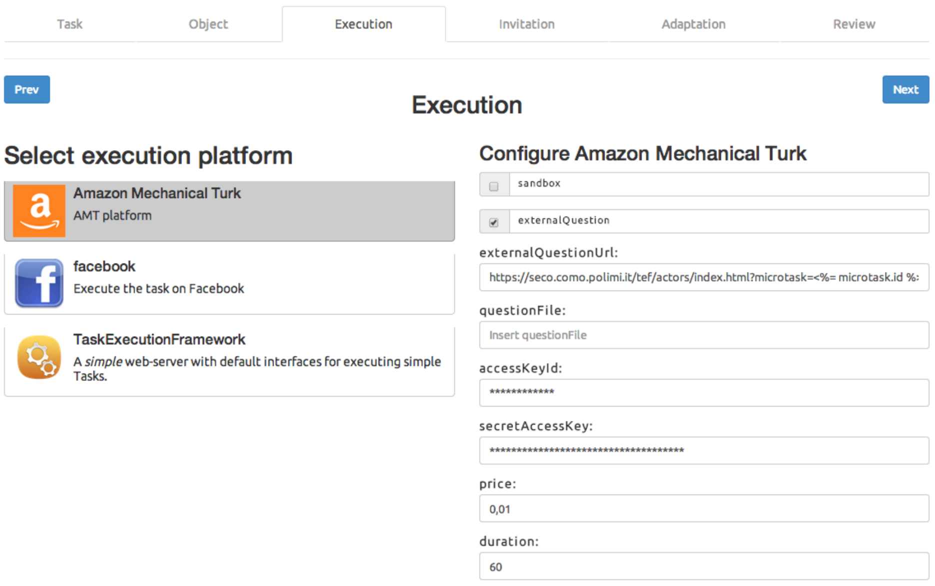
Data preview 910 objects loaded.

file	name
String	String
Helena Bonham Carter/047.jpg	Helena Bonham Carter
Helena Bonham Carter/050.jpg	Helena Bonham Carter
Lena Headey/022.jpg	Lena Headey
Lena Headey/025.jpg	Lena Headey
Morgan Freeman/036.jpg	Morgan Freeman

Figure 8.2: *Object design step.*

Object Design consists of defining the dataset which is subject to the analysis. In particular, it entails schema definition, instance collection and data cleaning (so as to eliminate irrelevant objects and make them conforming to their schema). In this step the designer can upload the set of objects of interest (e.g., as a JSON file), possibly together with a partial ground truth, i.e. correct solutions for a subset of the objects. Figures 8.2 shows the tool interface for acquiring and previewing the dataset. A similar UI is in place for collecting the ground truth.

8.2.3 Execution Design



Execution

Select execution platform

Amazon Mechanical Turk
AMT platform

facebook
Execute the task on Facebook

TaskExecutionFramework
A *simple* web-server with default interfaces for executing simple Tasks.

Configure Amazon Mechanical Turk

☐ sandbox

☑ externalQuestion

externalQuestionUrl: https://seco.como.polimi.it/tef/actors/index.html?microtask=<%= microtask.id %:

questionFile: Insert questionFile

accessKeyId: ************

secretAccessKey: ************************************

price: 0,01

duration: 60

Figure 8.3: *Selection of the execution platform.*

In this stage the user select the execution platform. Execution can be performed on traditional crowdsourcing platforms (e.g., AMT), on social networks (e.g. Facebook), or on custom user interfaces, implemented *ad hoc* by the designer. Figure 8.3 shows the selection of a particular execution platform in the tool. According to the selected platform, different parameters need to be provided: access token for accessing to the social network API and configuration parameter for the HITs on mechanical turk. The supported execution platforms are:

- Amazon Mechanical Turk: the task is translated into HITs on mechanical turk. The tool allows to use both the native AMT interface or a custom interface built with the Task Execution Framework.

- Facebook: it allows to directly execute the task on the social network. Since it uses the user interface provided by Facebook, it only support the Like and

Comment task type. For instance in case of a Like operation on Facebook, the system will spawn a Post on the wall for each microtask, containing a comment for each objects. Then periodically the Crowdsearcher will check for new answers by calling the social network API.

- Task Execution Framework: an external *execution* platform provided together with the CrowdSearcher. It provides both a default interface for executing the tasks, and a set of APIs for the creation of custom user interfaces to be deployed as stand-alone application, or embedded within third-party platforms such as Amazon Mechanical Turk. It uses a template approach (specifically Handlebars.js [2]) for building the web interface and it offers a set of JavaScript API in order to interact with the CrowdSearcher. Figure 8.4 show an example of custom user interface that can be built using the Task Execution Framework.

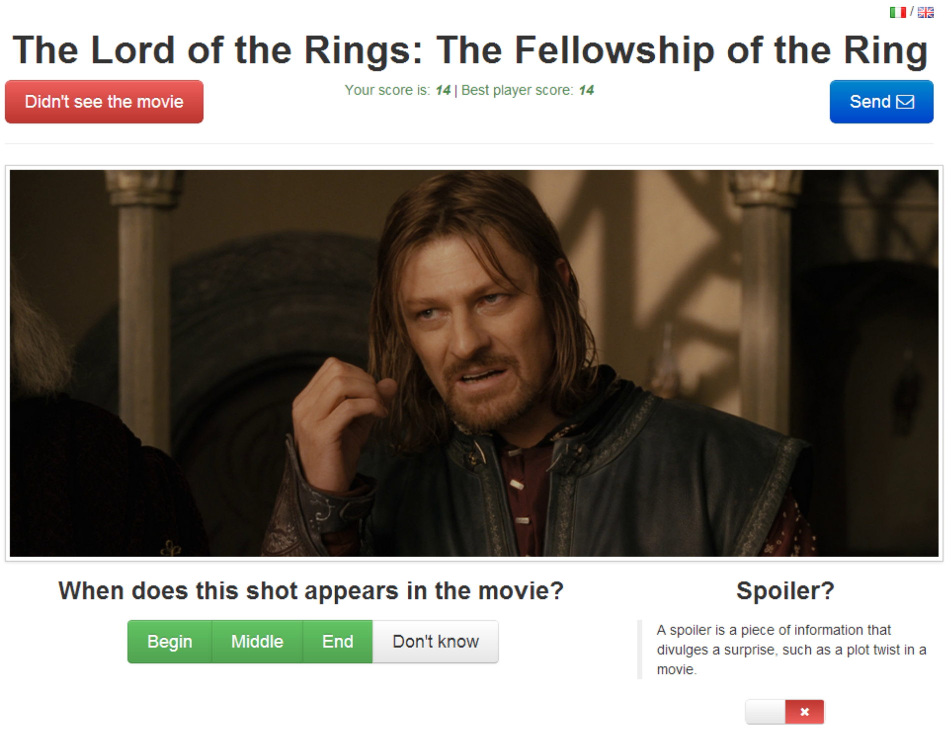
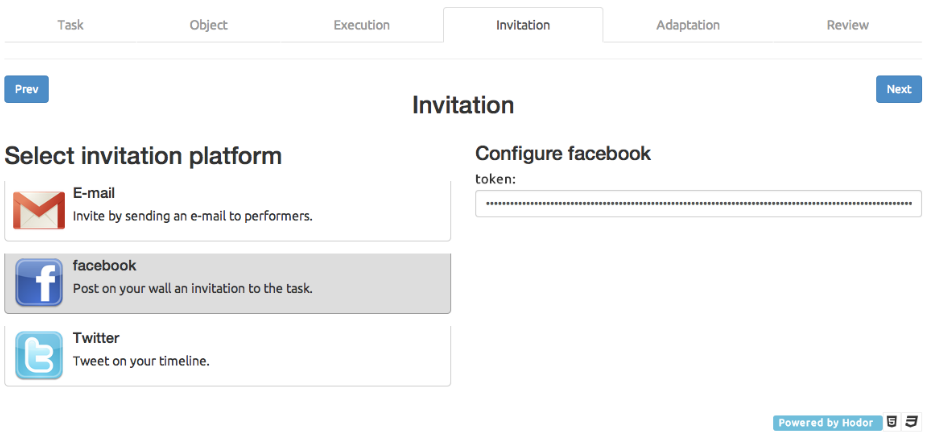

The Lord of the Rings: The Fellowship of the Ring
Didn't see the movie
Your score is: 14 | Best player score: 14
Send
When does this shot appears in the movie?
Begin
Middle
End
Don't know
Spoiler?
A spoiler is a piece of information that divulges a surprise, such as a plot twist in a movie.

Figure 8.4: *An example of custom task execution UI.*



Invitation

Select invitation platform

E-mail
Invite by sending an e-mail to performers.

facebook
Post on your wall an invitation to the task.

Twitter
Tweet on your timeline.

Configure facebook

token: ••

Figure 8.5: *Selection of the invitation platform.*

8.2.4 Performer Selection or Invitation Design

In this step the user select the invitation channel to be used to invite people to perform the task. For instance, if Facebook is selected as invitation platform (as shown in Figure 8.5), users which are reachable from a specific Facebook account will be invited through a post on the account's wall. The supported invitation platforms are:

- Mail: the invitation in sent via email to a list of contact provided by the user;

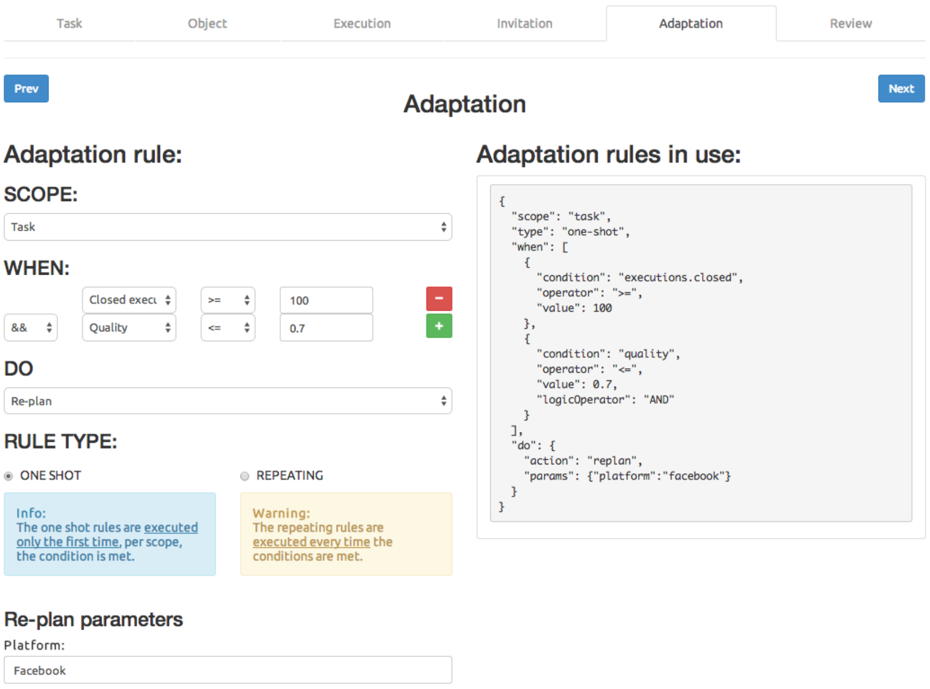
- Facebook: the invitation to perform the task is posted on the wall of the user's account;
- Twitter: the invitation to perform the task is posted as a tweet on the timeline of the user's account.

Furthermore, a URL pointing to the task execution UI is generated by the system and can be sent manually, at any time, by the designer, through the channels of choice.

8.2.5 Adaptation Design

The adaptation design consists in the definition of active rules that change the behavior of the system according to the status of the execution of the crowdsourcing task. This part implements the concept described in Chapter 5. Adaptation and control can be applied to Data Objects, Tasks, Performers, and Platforms. Each

[2] http://handlebarsjs.com/



Adaptation rule:

SCOPE: Task

WHEN:

	Closed execu	>=	100
&&	Quality	<=	0.7

DO: Re-plan

RULE TYPE: ● ONE SHOT ○ REPEATING

Info: The one shot rules are executed only the first time, per scope, the condition is met.

Warning: The repeating rules are executed every time the conditions are met.

Re-plan parameters

Platform: Facebook

Adaptation rules in use:

```
{
  "scope": "task",
  "type": "one-shot",
  "when": [
    {
      "condition": "executions.closed",
      "operator": ">=",
      "value": 100
    },
    {
      "condition": "quality",
      "operator": "<=",
      "value": 0.7,
      "logicOperator": "AND"
    }
  ],
  "do": {
    "action": "replan",
    "params": {"platform":"facebook"}
  }
}
```

Figure 8.6: *Configuration of the adaptation.*

of these dimensions is associated with simple statistics, such as Number of executions and Quality, which are computed by formulas taking into account the current crowd answers. Tasks and Platforms have also parameters describing their idle time and the number of objects which were either closed (obtained a result supported by a sufficient number of answers) or invalid (unsupported by adequate answers). Figure 8.6 shows the interface that allows the composition of this active rules by selecting the scope of the rule (Task, Object, Performer, Platform), building simple predicates that use target's properties (e.g. executions and quality) and then choosing the action to perform. Actions depend on each specific element and include:

Object

- REDO: A single object is re-sent to the crowd to be evaluated. The previous executions are deleted. Similar to a object granularity reset switch over.
- CANCEL: An object is removed from the evaluation.

Task

- RE-INVITE: A new set of performers is invited to perform the task. Similar to a task granularity continuous switch over

- RE-PLAN: The unfinished objects of the task are sent again to the crowd on a different platform. Similar to a task granularity instantaneous switch over.

Performer

- BAN: The performer is marked as spammer and he won't be able to execute the task anymore.

Platform

- MIGRATE: The task is sent to another platform. All the previous execution are deleted. Similar to a task granularity reset switch over.

8.3 Task Execution and Control

Once the task is created, the Crowdsearcher takes the specification built in this process and generate the data structure needed for its execution.

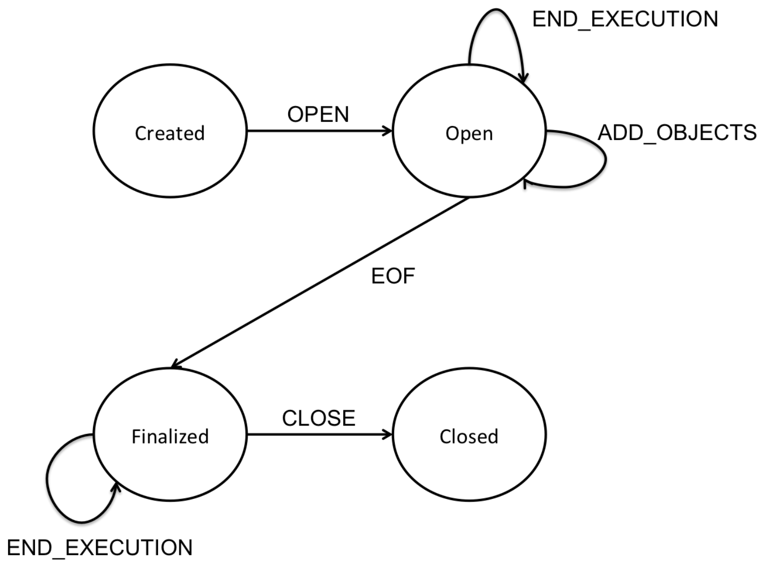


Figure 8.7: *The task life cycle*

Figure 8.7 show the life cycle of a task in the system. The circles are the states, while the arrows represent the internal events that occur in the reactive environment. Once a task is created it only exist as a persistent object in the database, in order to be executed by the workers it needs to be deployed to the platforms through the *opening* phase. In this phase the system instantiates the reactive rules and the data structure required for the task control (the control mart) and deploys the task on the correct platform, it creates the post on the social networks or the HIT on Amazon Mechanical Turk.

While the task is in the *OPEN* state, the system receives and aggregate the answers given by the performer (represented by the *END_EXECUTION* arrows). Furthermore is still possible to add objects to the task, in order to support the patterns described in Chapter 6. Once the *EOF* event is sent, no more objects are accepted, but the answers are still collected.

When, according to some control rules, the evaluation is completed, the task is closed.

8.4 Task Monitoring

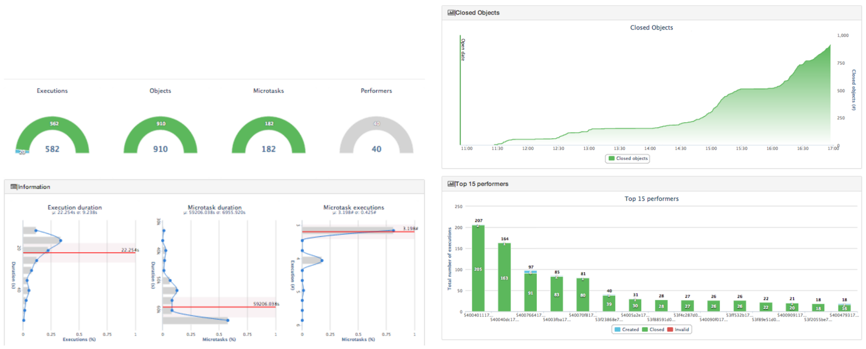


Figure 8.8: *CrowdSearcher dashboard (excerpt).*

CrowdSearcher offers also a dashboard, shown in Figure 8.8, that allows designer to continuously monitor crowd-based applications. The dashboard provides visibility upon:

- Count of executions (closed, active and invalid), objects (closed or currently under evaluation), microtasks (closed or open), and performers.
- Distribution of execution and microtask duration. It shows in a more detailed way the status of the task. A long average execution duration can either mean that the performers are carefully executing the task or that the task is too difficult, while extremely short durations may mean that the performers are careless in evaluating the objects.
- Distribution of the closed object over time. This graph gives a hint on how well the task is performing. If it shows a rapid growth, it means that the performers are quickly agreeing on the evaluations of the objects.
- Distribution of executions assigned to top performers. The executions are divided in *closed*, *active* and *invalid*. Too many invalid executions on a given performer can be a signal of bad behavior.

This allows monitoring the performance of tasks over time. This, together with the possibility of easily and quickly configuring the crowdsourcing task with the tool, allows to compare different task configurations and to see how the crowd reacts to different types of control.

8.5 CrowdSearcher Architecture

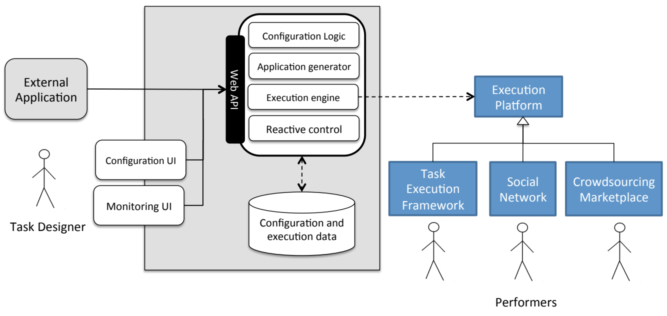


Figure 8.9: *The CrowdSearcher architecture.*

Figure 8.9 shows the architecture of the system. The core of the system is a *Node.js* application, integrating the data structures stored in a non-sql data source (namely a MongoDB instance) connected to the reactive control engine, and offering a set of APIs to support integration with external applications. The *Task Designer* can create and manage tasks either through the CrowdSearcher web interface or through a custom application invoking the API.
The application generator takes the task model built in the process described in Section 8.2 and generates the data structure, the rules and the objects required for the task execution and control; finally, it deploys the task on the execution platform of choice. Reactive rules are translated into scripts, whose triggering is modeled through internal platform events. Precedence between rules is implicitly obtained by defining the scripts in the proper order.

For example, Listing 8.1 show the rule that computes majority of answers for the classify operation. numberOfAnswers is the minimum number of answers needed and agreement is the number of performers that must agree on a particular category. The rule has three main parts: *(i)* lines 15–18 updates the control variable (total number of answer and the count of the selected category); *(ii)* lines 21–39 select the category that currently has the higher count and set the control variables; *(iii)* lines 43–49 close the current object if termination conditions are met.

Listing 8.1: *Calculation of the majority for a classify operation.*

```
1  var performRule = function( data, config, callback){
```

```
// Array of categories
var categories = data.task.operation.params.categories;

// Minimum number of answers
var numberOfAnswers = config.numberOfAnswers;

// Agreement needed
var agreements = config.agreement;

var applyMajority = function(annotation,callback){
  var object = annotation.object;

  // Updating the metadata
  var count = object.getMetadata('count');
  object.setMetadata('count',count+1);
  var selectedCategoryCount = object.getMetadata(annotation.
      response);
  object.setMetadata(annotation.response,selectedCategoryCount
      +1);

  // Build the data structure [category, count]
  var categoriesMetadata = [];
  _.each(categories,function(category){
    var count = {
    category:category,
    count: object.getMetadata(category)
  };

  categoriesMetadata.push(count);

  // Selecting the category with maximum count
  var max = _.max(categoriesMetadata,function(categoryCount){
    return categoryCount.count;
  });

  // Verifying that the maximum is unique
  var otherMax = _.where(categoriesMetadata,{count:max.count});
  if(otherMax.length==1){
    object.setMetadata('result',max.category);
  }

  // Checking if the object should be closed
  // If numberOfAnswers is equal to 0, then ignore the parameter
  if(count === numberOfAnswers || numberOfAnswers=== 0){
    if(max.count >= agreement){
    object.setMetadata('status','CLOSED');
    }
  }

  return object.save(domain.bind(callback));
  )};

  //Call the applyMajority function of each annotation
  return async.eachSeries(annotations,domain.bind(applyMajority),
      callback);
};

var params = {
  agreement: ['number'],
  numberOfAnswers: ['number']
};
```

The execution UI interacts with the core of the system and the execution platform to prepare the questions, collect the answers and send them back to the CrowdSearcher.

8.6 Evaluation

The implementation of Crowdsearcher granted the opportunity to put the approach developed in this thesis at work in the development of several real-world applications.

Multimedia Analysis and Search [21] [38]: By combining CrowdSearcher with an infrastructure for multimedia analysis, several applications were created (e.g. trademark and logo detection in video) that demonstrates how the contribution of (expert) crowds can improve the recall of state-of-the-art traditional algorithms, with no loss in terms of precision.

In addition, several experiments scenarios were built, aimed at supporting my research activities, while validating the applicability and flexibility of our approach (most of them are described in this thesis). For instance:

Politician party [17]:In this experiment the crowd was asked to classify the political affiliation of 30 members of the Italian parliament. To single performer is presented a set of photos and names and has to match the single politician to the correct political party.

Politician law [17]: In this experiment photos of 50 members of the Italian parliament were presented to the crowd. The users had to indicate if they have ever been accused, prosecuted or convicted. Each performer sees, in a fixed amount of time, a number of photos which raises as a function of the performer's ability, and, after he give his answer, the system presents a report with correct answers and the ranking of the other performers.

Politician order [17]: The objective of this experiment was to produce the total ranking of 25 politicians. At each performer is presented a pair of politicians and is asked to choose the one he likes the most.

Model search (1) [12]: In this experiment the crowd evaluated the results of a query performed on a model repository. Given the query, and two possible results, the performers had to choose which one is better.

Model search (2) [13]: In this experiment, the crowd evaluated the ranked results of a query performed over a model repository. Given the query and two possible ranks, the performer had to decide which one was better.

Transportation: In this experiment the crowd had to validate the classification of tweets related to public transportation made by an automatic tool. The performers had to evaluate the correctness of the topic, geo-localization and polarity of the tweets.

Movie Scenes [19]: In this experiment the crowd had to classify images taken from movie scenes. Each performer had to tell if an image belonged to the initial, middle or final part of the film, and, in the latter two cases, if the image was a spoiler.

Movie actors [19]: In this experiment the crowd had to identify actor in movie scenes. In particular this scenario was divided in two parts: in the first the performers had to insert the name of the actors present in the image and in the second they had to validate the answers given by the others

Professors images [22]: In this experiment the crowd was asked to evaluate of relevance images about the professor of retrieved through the Google Image API. The performers needed to specify whether each image represents the professor himself, some relevant people or places, other related materials (papers, slides, graphs or technical materials), or it is not relevant at all.

Among the various aspects I studied, I considered the cost of development of the different applications. Figure 8.2 reports the approximate development effort of nine recent application. Note that data preparation and UI generation (ad hoc) were required regardless of the adopted method. This approach to monitoring required large efforts for the first applications, which was well compensated by a high reuse of rules in the subsequent applications. Note also that this method enables fast prototyping of applications in the small scale, with small crowds who give interaction feedbacks; tuning is quite efficient, as it can be done by changing configuration parameters from within the design framework.

Experiment	Dataset preparation	UI generation	Monitoring	Tuning
Politicians Party	1.5	4	5	1
Politicians Law	1.5	2	7	1.5
Politicians Order	1.5	2	3	1
Model Search 1	3	3	0 (*)	1
Model Search 2	3	3	0 (*)	1.5
Transportation.	1	5	0 (*)	1
Movies Scenes	1	4	2	1
Movies Actors	2	3.5	0.5 (*)	0.5
Image Select	1.5	4.5	1.5	1

Table 8.2: *Development effort for different applications (man/days). (*) = high reuse of existing rules.*

8.7 Conclusions

In this chapter I described the prototype that has been developed during the the work of this thesis. Thanks to this application I was able to:

- Demonstrate the validity of my approach described in each chapters.
- Exploring the flexibility and the effectiveness of my method.
- Utilize my method in real world scenarios.

CHAPTER 9

Conclusions and Future Work

9.1 Conclusion

In this work I developed methodologies for creating applications that leverage the knowledge of the crowd or social communities. I first defined a platform and domain agnostic model for modeling Crowdsourcing application and a design process that guide the user that need to build the application. I also explored an alternative method for designing Crowdsourcing task based on empirical evaluation of possible task model.

Then I focused on the problem of controlling the execution of a Crowdsourcing application. Unlike other crowdsourcing system that only provide limited and predefined controls; in contrast, I presented an approach which provides fine-level, powerful and flexible controls. The control can be automatically generated starting from the task model or easily configured through active rules. In the experiment I showed that the proposed approach is a good compromise between the conflicting requirements of design automation, flexibility, and expressive power.

Then I extended the model in order to dynamically change the crowds and crowd-based systems according to how the crowd responds to task assignments. I illustrated hot to achieve platform and community interoperability using the concepts defined in the previous chapters of the thesis. In the experiments I showed how the applications are actually able to dynamically involve different platforms and communities, with different characteristics, cost, and quality of results.

Next I extended the model in order to be able to design and deploy crowd-based applications as arbitrarily complex workflows of elementary tasks, which emphasises the use of crowdsourcing patterns. In the experiments I demonstrated the viability of the approach and showed how the different choices in workfllow design may impact on the cost, time and quality of crowd-based activities.

Then I studied how to boost user engagement using non monetary rewards. In particular I studied the problem of organizing social challenges because as it requires not only to raise the initial interest of the participants, but also to keep it high throughout the lifetime of the challenge, and actually the main objective of the challenge organizers is to preserve or even increase the social mobilization more or less continuously throughout the planned lifetime, as success is typically measured not only by the global mobilization but also by a growing interest of players while the challenge is in progress. I defined and compared a number of strategies that can be used by organizers for increasing social mobilization, in a challenge which uses multiple social networks, over a planned lifetime, in the presence of players, voters, and catalysts.

Finally I described the prototype that was developed during work on this thesis in order to perform the experiments to validate my approach. This tool also allowed me to apply my method to real world scenarios.

9.2 Future Work

There are a few future research possibilities for extending my work:

- Expert Finding: in my work I only experimented with simple push and pull methods for task assignment, even though the model already supports complex matching between task and performer through the control rules. This approach can be further extended by not only taking in account the expertise of a worker but also the relation among them. For instance in [20] the authors address the problem of selecting experts inside a social network. They extract the expertise of a user by mining his activities on the social network (status updates, tweet posted, post liked, user followed, etc..). For instance a user could be a soccer expert because he post about soccer or if he follow a soccer expert.

- Automatic Generation from the Problem: in my work the design phase is done manually (with the exception of the process proposed in Section 3.6), only starting from the control phase the application is generated automatically (it's not necessary to manually write the control rules). In the current state the Problem model is used as guide for designing the task, but it is possible to derive all of part of the Task model. I started to study this part in the prototype described in Chapter 8, where the user first select the problem he want to solve, and then select the task type from a set of predefined ones. These options are partially configured according to the problem selected.

Bibliography

[1] The design with intent method: A design tool for influencing user behaviour. *Applied Ergonomics*, 41(3):382 – 32, 2010.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[3] Salman Ahmad, Alexis Battle, Zahan Malkani, and Sepander Kamvar. The jabberwocky programming environment for structured social computing. In *UIST '11*, pages 53–64. ACM, 2011.

[4] Omar Alonso, Daniel E. Rose, and Benjamin Stewart. Crowdsourcing for relevance evaluation. *SIGIR Forum*, 42(2):9–15, November 2008.

[5] Yoram Bachrach, Thore Graepel, Tom Minka, and John Guiver. How to grade a test without knowing the answers — a bayesian graphical model for adaptive crowdsourcing and aptitude testing. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1183–1190, New York, NY, USA, 2012. ACM.

[6] Krisztian Balog, Pavel Serdyukov, and Arjen P de Vries. Overview of the trec 2010 entity track. Technical report, DTIC Document, 2010.

[7] Elena Baralis, Stefano Ceri, and Stefano Paraboschi. Modularization techniques for active rules design. *ACM Trans. Database Syst.*, 21(1):1–29, 1996.

[8] Elena Baralis, Stefano Ceri, and Jennifer Widom. Better termination analysis for active databases. In *Rules in Database Systems*, pages 163–179, 1993.

[9] Pippin Barr, James Noble, and Robert Biddle. Video game values: Human-computer interaction and games. *Interacting with Computers*, 19(2):180–195, 2007.

[10] C. Daniel Batson, Nadia Ahmad, and Jo ann Tsang. Four motives for community involvement. *Journal of Social Issues*, pages 429–445, 2002.

[11] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. Soylent: a word processor with a crowd inside. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 313–322, New York, NY, USA, 2010. ACM.

[12] Bojana Bislimovska, Alessandro Bozzon, Marco Brambilla, and Piero Fraternali. Search upon uml repositories with text matching techniques. In *Search-Driven Development-Users, Infrastructure, Tools and Evaluation (SUITE), 2012 ICSE Workshop on*, pages 9–12. IEEE, 2012.

[13] Bojana Bislimovska, Alessandro Bozzon, Marco Brambilla, and Piero Fraternali. Textual and content-based search in repositories of web application models. *ACM Transactions on the Web (TWEB)*, 8(2):11, 2014.

[14] Gee-Woo Bock, Robert W. Zmud, Young-Gul Kim, and Jae-Nam Lee. Behavioral intention formation in knowledge sharing: Examining the roles of extrinsic motivators, social-psychological factors, and organizational climate. *MIS Q.*, 29(1):87–111, March 2005.

[15] Alessandro Bozzon, Marco Brambilla, and Stefano Ceri. Answering search queries with crowdsearcher. In *21st World Wide Web Conference (WWW 2012)*, pages 1009–1018, 2012.

[16] Alessandro Bozzon, Marco Brambilla, and Stefano Ceri. Answering search queries with crowdsearcher. In *21st Int.l Conf. on World Wide Web 2012*, WWW '12, pages 1009–1018. ACM, 2012.

[17] Alessandro Bozzon, Marco Brambilla, Stefano Ceri, and Andrea Mauri. Reactive crowdsourcing. In *22nd World Wide Web Conf.*, WWW '13, pages 153–164, 2013.

[18] Alessandro Bozzon, Marco Brambilla, Stefano Ceri, Andrea Mauri, and Riccardo Volonterio. Pattern-based specification of crowdsourcing applications. In *Web Engineering*, pages 218–235. Springer, 2014.

[19] Alessandro Bozzon, Marco Brambilla, Stefano Ceri, Andrea Mauri, and Riccardo Volonterio. Pattern-based specification of crowdsourcing applications. In *Web Engineering, 14th International Conference, ICWE 2014, Toulouse, 2014*, pages 218–235, 2014.

[20] Alessandro Bozzon, Marco Brambilla, Stefano Ceri, Matteo Silvestri, and Giuliano Vesci. Choosing the right crowd: expert finding in social networks. In *16th International Conference on Extending Database Technology*, EDBT '13, pages 637–648, New York, NY, USA, 2013. ACM.

[21] Alessandro Bozzon, Ilio Catallo, Eleonora Ciceri, Piero Fraternali, Davide Martinenghi, Marco Tagliasacchi, and M Tagliasacchi. A framework for crowdsourced multimedia processing and querying. *CrowdSearch*, 842:42–47, 2012.

[22] Marco Brambilla, Stefano Ceri, Andrea Mauri, and Riccardo Volonterio. Community-based crowdsourcing. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 891–896. International World Wide Web Conferences Steering Committee, 2014.

[23] Marco Brambilla, Stefano Ceri, Andrea Mauri, and Riccardo Volonterio. Adaptive and interoperable crowdsourcing. *IEEE Internet Computing - In print*, 2015.

[24] Marco Brambilla, Stefano Ceri, Andrea Mauri, and Riccardo Volonterio. An explorative approach for crowdsourcing tasks design. In *Proceedings of the 24th International Conference on World Wide Web Companion*, WWW '15 Companion, pages 1125–1130, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee.

[25] Michael D. Buhrmester, Tracy Kwang, and Samuel D. Gosling. Amazon's Mechanical Turk: A new source of inexpensive, yet high-quality data? *Perspectives on Psychological Science*, in press.

[26] I. Celino, D. Cerizza, S. Contessa, M. Corubolo, D. Dell'Aglio, E.D. Valle, and S. Fumeo. Urbanopoly – a social and location-based game with a purpose to crowdsource your urban data. In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*, pages 910–913, Sept 2012.

[27] Irene Celino, Simone Contessa, Marta Corubolo, Daniele Dell'Aglio, Emanuele Della Valle, Stefano Fumeo, and Thorsten KrÃ¼ger. Linking smart cities datasets with human computation - the case of urbanmatch. In Philippe CudrÃ©-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, JÃ©rÃ´me Euzenat, Manfred Hauswirth, JosianeXavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *The Semantic Web - ISWC 2012*, volume 7650 of *Lecture Notes in Computer Science*, pages 34–49. Springer Berlin Heidelberg, 2012.

[28] Ran Cheng and Julita Vassileva. Design and evaluation of an adaptive incentive mechanism for sustained educational online communities. *User Model. User-Adapt. Interact.*, 16(3-4):321–348, 2006.

[29] Roberta Cuel, Oksana Tokarchuk, Monika Kaczmarek, Jakub Dzikowski, Elena Simperl, and Szymon Lazaruk. Making your semantic application addictive: incentivizing users! In *2nd Interaking your semantic application addictive: incentivizing users!national Conference on Web Intelligence, Mining and Semantics, WIMS '12, Craiova, Romania, June 6-8, 2012*, page 4, 2012.

[30] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Using the crowd for top-k and group-by queries. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 225–236, New York, NY, USA, 2013. ACM.

[31] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY*, 39(1):1–38, 1977.

[32] Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. Gamification. using game-design elements in non-gaming contexts. In *International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, pages 2425–2428, 2011.

[33] Catherine Dwyer. Digital relationships in the "myspace" generation: Results from a qualitative study. In *40th Hawaii International International Conference on Systems Science (HICSS-40 2007), 2007, USA*, page 19, 2007.

[34] Arpad E. Elo. *The rating of chessplayers, past and present.* Arco Pub., New York, 1978.

[35] Rebecca Ermecke, Philip Mayrhofer, and Stefan Wagner. Agents of diffusion - insights from a survey of facebook users. In *42st Hawaii International International Conference on Systems Science (HICSS-42 2009)), 5-8 January 2009, Waikoloa, Big Island, HI, USA*, pages 1–10, 2009.

[36] B. J. Fogg. Persuasive technology: Using computers to change what we think and do. *Ubiquity*, 2002(December), December 2002.

[37] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In *ACM SIGMOD 2011*, pages 61–72. ACM, 2011.

[38] Piero Fraternali, Marco Tagliasacchi, Davide Martinenghi, Alessandro Bozzon, Ilio Catallo, Eleonora Ciceri, Francesco Nucci, Vincenzo Croce, Ismail Sengor Altingovde, Wolf Siberski, et al. The cubrik project: human-enhanced time-aware multimedia search. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 259–262. ACM, 2012.

[39] Ujwal Gadiraju, Ricardo Kawase, and Stefan Dietze. A taxonomy of microtasks on the web. In *Proceedings of the 25th ACM Conference on Hypertext and Social Media*, HT '14, pages 218–223, New York, NY, USA, 2014. ACM.

[40] L. Galli, P. Fraternali, D. Martinenghi, M. Tagliasacchi, and J. Novak. A draw-and-guess game to segment images. In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*, pages 914–917, Sept 2012.

[41] Duncan Gilchrist, Luca Michael, and Malhotra Deepak. When 3+1>4: Gift structure and reciprocity in the field. In *Working Paper. (April 2015. Revised and resubmitted, Management Science.)*, 2015.

[42] Christopher Harris. You're hired! an examination of crowdsourcing incentive models in human resource tasks. *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)*, pages 15–18, 2011.

[43] William H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[44] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 64–67, New York, NY, USA, 2010. ACM.

[45] Huan Jiang and Shigeo Matsubara. Efficient task decomposition in crowdsourcing. In *PRIMA 2014: Principles and Practice of Multi-Agent Systems*, pages 65–73. Springer, 2014.

[46] Hyun Joon Jung and Matthew Lease. Inferring missing relevance judgments from crowd workers via probabilistic matrix factorization. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 1095–1096, New York, NY, USA, 2012. ACM.

[47] Gabriella Kazai, Jaap Kamps, and Natasa Milic-Frayling. An analysis of human factors and label accuracy in crowdsourcing relevance judgments. *Inf. Retr.*, 16(2):138–178, 2013.

[48] George A. Khoury, Adam Liwo, Firas Khatib, Hongyi Zhou, Gaurav Chopra, Jaume Bacardit, Leandro O. Bortot, Rodrigo A. Faccioli, Xin Deng, Yi He, Pawel Krupa, Jilong Li, Magdalena A. Mozolewska, Adam K. Sieradzan, James Smadbeck, Tomasz Wirecki, Seth Cooper, Jeff Flatten, Kefan Xu, David Baker, Jianlin Cheng, Alexandre C. B. Delbem, Christodoulos A. Floudas, Chen Keasar, Michael Levitt, Zoran Popovic, Harold A. Scheraga, Jeffrey Skolnick, Silvia N. Crivelli, and Foldit Players. Wefold: A coopetition for protein structure prediction. *Proteins: Structure, Function, and Bioinformatics*, 82(9):1850–1868, 2014.

[49] Shailesh Kochhar, Stefano Mazzocchi, and Praveen Paritosh. The anatomy of a large-scale human computation engine. In *HCOMP '10*, pages 10–17. ACM, 2010.

[50] Edith Law and Luis von Ahn. *Human Computation*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2011.

[51] Matthew Lease and Gabriella Kazai. Overview of the trec 2011 crowdsourcing track. Technical report, 2011.

[52] Christopher H. Lin, Mausam, and Daniel S. Weld. Crowdsourcing control: Moving beyond multiple choice. In *UAI*, pages 491–500, 2012.

[53] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Turkit: tools for iterative tasks on mechanical turk. In *HCOMP '09*, pages 29–30. ACM, 2009.

[54] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Exploring iterative and parallel human computation processes. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 68–76, New York, NY, USA, 2010. ACM.

[55] Greg Little, Lydia B Chilton, Max Goldman, and Robert C Miller. Turkit: human computation algorithms on mechanical turk. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 57–66. ACM, 2010.

[56] Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. Human-powered sorts and joins. *Proc. VLDB Endow.*, 5(1):13–24, September 2011.

[57] Adam Marcus, Eugene Wu, Samuel Madden, and Robert C. Miller. Crowdsourced databases: Query processing with people. In *CIDR 2011*, pages 211–214. www.cidrdb.org, January 2011.

[58] Winter Mason and Duncan J. Watts. Financial incentives and the "performance of crowds". In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '09, pages 77–85, New York, NY, USA, 2009. ACM.

[59] Paul Milgrom and John Roberts. Economics, organization and management, 1992.

[60] Patrick Minder and Abraham Bernstein. How to translate a book within an hour: towards general purpose programmable human computers with crowdlang. In *WebScience 2012*, pages 209–212, Evanston, IL, USA, June 2012. ACM.

[61] Sabine Niebuhr and Daniel Kerkow. Captivating patterns - A first validation. In *Second International Conference on Persuasive Technology, PERSUASIVE 2007, Palo Alto, CA, USA, 2007*, pages 48–54, 2007.

[62] Stefanie Nowak and Stefan Rüger. How reliable are annotations via crowdsourcing: a study about inter-annotator agreement for multi-label image annotation. In *Proceedings of the international conference on Multimedia information retrieval*, MIR '10, pages 557–566, New York, NY, USA, 2010. ACM.

[63] Aditya G. Parameswaran and Neoklis Polyzotis. Answering queries using humans, algorithms and databases. In *CIDR 2011*, pages 160–166, Asilomar, CA, USA, January 2011.

[64] Hyunjung Park, Richard Pang, Aditya G. Parameswaran, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: A system for declarative crowdsourcing. *PVLDB*, 5(12):1990–1993, 2012.

[65] Al Mamunur Rashid, Kimberly S. Ling, Regina D. Tassone, Paul Resnick, Robert E. Kraut, and John Riedl. Motivating participation by displaying the value of contribution. In *Conference on Human Factors in Computing Systems, CHI 2006, Montréal, Canada, April 22-27*, pages 955–958, 2006.

[66] Jakob Rogstadius, Vassilis Kostakos, Aniket Kittur, Boris Smus, Jim Laredo, and Maja Vukovic. An assessment of intrinsic and extrinsic motivation on task performance in crowdsourcing markets. In Lada A. Adamic, Ricardo A. Baeza-Yates, and Scott Counts, editors, *ICWSM*. The AAAI Press, 2011.

[67] K. Sato, R. Hashimoto, M. Yoshino, R. Shinkuma, and T. Takahashi. Incentive mechanism considering variety of user cost in p2p content sharing. In *IEEE Global Telecommunications Conference (GLOBECOM) 2008*, pages 1–5, Nov 2008.

[68] Ognjen Scekic, Hong Linh Truong, and Schahram Dustdar. Incentives and rewarding in social computing. *Commun. ACM*, 56(6):72–82, 2013.

[69] Aaron D. Shaw, John J. Horton, and Daniel L. Chen. Designing incentives for inexpert human raters. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, CSCW '11, pages 275–284, New York, NY, USA, 2011. ACM.

[70] Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 614–622, New York, NY, USA, 2008. ACM.

[71] Penelope Sweetser and Peta Wyeth. Gameflow: a model for evaluating player enjoyment in games. *Computers in Entertainment*, 3(3):3, 2005.

[72] Fujio Toriumi, Ken Ishida, and Kenichiro Ishii. Encouragement methods for small social network services. In *2008 IEEE / WIC / ACM International Conference on Web Intelligence, WI 2008, Sydney*, pages 84–90, 2008.

[73] Matteo Venanzi, John Guiver, Gabriella Kazai, Pushmeet Kohli, and Milad Shokouhi. Community-based bayesian aggregation models for crowdsourcing. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 155–164, New York, NY, USA, 2014. ACM.

[74] Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. Max algorithms in crowdsourcing environments. In *WWW '12*, pages 989–998, New York, NY, USA, 2012. ACM.

[75] Molly McLure Wasko and Samer Faraj. Why should I share? examining social capital and knowledge contribution in electronic networks of practice. *MIS Quarterly*, 29(1):35–57, 2005.

[76] Jennifer Widom and Stefano Ceri. *Active Database Systems*. Morgan Kaufmann, 1996.

[77] Ming Yin, Yiling Chen, and Yu-An Sun. The effects of performance-contingent financial incentives in online labor markets. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*, 2013.

[78] Ming Yin, Yiling Chen, and Yu-An Sun. Monetary interventions in crowdsourcing task switching. In *Proceedings of the Seconf AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2014, November 2-4, 2014, Pittsburgh, Pennsylvania, USA*, 2014.

[79] Kazufumi Yogo, Ryoichi Shinkuma, Taku Konishi, Satoko Itaya, Shinichi Doi, Keiji Yamada, and Tatsuro Takahashi. Incentive-rewarding mechanism to stimulate activities in social networking services. *Int. Journal of Network Management*, 22(1):1–11, 2012.

[80] Chen Jason Zhang, Lei Chen, and Yongxin Tong. Mac: A probabilistic framework for query answering with machine-crowd collaboration. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 11–20, New York, NY, USA, 2014. ACM.