Adaptive and Interoperable Crowdsourcing

Marco Brambilla, Stefano Ceri, Andrea Mauri, Riccardo Volonterio

Politecnico di Milano. Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)

Piazza Leonardo da Vinci, 32. 20133 Milano, Italy

{marco.brambilla, stefano.ceri, andrea.mauri, riccardo.volonterio}@polimi.it

ABSTRACT

Crowd-based computing is an increasingly popular paradigm for building Web applications, which uses the collective strength of human actors for performing tasks which are most suited to humans than to computers. Interaction with the crowds was originally confined to specifically designed crowdsourcing platforms, such as Amazon Mechanical Turk; more recently, crowd-based computing has been reconsidered and extended, targeting social networks such as Facebook, Twitter or LinkedIn, or including basic and direct interaction mechanisms, such as routing personal emails or tweets.

This paper is focused on interoperability and adaptation of crowd-based applications, i.e. the ability of supporting multi-platform applications and of adapting them in reaction to events; we specifically support dynamic interoperability, i.e. the ability to modify the execution platforms while the application is ongoing, as a reaction to crowd behavior, which is hardly predictable. We show how interoperability control can be specified at a high, declarative level and then implemented using active rules, thereby obtaining answers from crowds engaged in different communities; we show the effect on precision, delay and cost.

KEYWORDS

Crowdsourcing, Adaptivity, Interoperability, Web applications, human factors, active rules, social network, workflow, process, content annotation, adaptation, modeling, datamart.

1. INTRODUCTION

A number of emerging crowd-based applications cover a variety of scenarios, including opinion mining, localized information gathering, marketing campaigns, expert response gathering, and so on. The common aspect of these applications is the interaction between the requestor (who poses questions), the system (which organizes a response collection campaign), and the wide set of responders (who are in charge of

providing answers, despite being typically unknown to the requestor). Crowdsourcing platforms such as Amazon Mechanical Turk are a natural environment for deploying such applications, since they support the assignment to humans of simple and repeated tasks, such as translation, proofing, content tagging and items classification, by combining human contribution and automatic analysis of results [1]. However, also other popular community-based and social networking platforms can be used for collecting responses, such as vertical community-based building systems (e.g., FourSquare or Yelp) or general-purpose social networks (e.g., Facebook or Twitter) [1].

In the various platforms, crowds take part to social computations both for monetary rewards and for non-monetary motivations, such as public recognition, fun, or genuine will of sharing knowledge [2]. To get the best possible results, we propose that crowd-based applications should dynamically adapt and take advantage of the diversity of these platforms. In particular, the same application can be deployed over multiple platforms and dynamically adapt based on how the crowd behaves, so as to exploit the peculiar features of each platform and thus get the best quality of results, while minimizing cost or execution time. In static crowdsourcing planning, just a few options are available for optimizing executions and results.

The main focus of this paper is the **adaptive, dynamic interoperability of crowd-based applications**, which includes both the possibility of statically determining the target crowds and crowd-based systems for each application, and of dynamically adapting them by taking into account how the crowd behaves in responding to task assignments. Interoperability is guaranteed by the use of a high-level, platform-independent model, presented in previous work [3,4,14], that guarantees that tasks of given kinds can be deployed to different platforms and that the same objects (and sometimes even the same performers) can be involved in executing the same application over multiple platforms. Enabling dynamic interoperability for crowd-based applications is new, and opens important opportunities, including tuning the cost of crowdsourcing campaigns, or choosing the most appropriate social community for performing given tasks, or excluding from the computations platforms which exhibit poor performance. In this work we discuss how dynamic adaptation techniques, already in use in other fields such as workflow management and information systems, can be tailored and applied to the peculiar context of crowdsourcing.

This paper is organized as follows: Section 2 discusses previous and related work and Section 3 introduces the running example. Then, Section 4 dwells into application interoperability by illustrating how interoperability can be defined by two kinds of active rules, respectively at a high level of abstractions (supported by a user-friendly design tool) and at a low level of abstraction (covering a broader spectrum of alternatives); it also illustrates how high-level rules are translated into low-level rules. Section 5 presents some experimental results, and Section 6 concludes.

2. Previous and related work

In order to make this paper self-contained, we summarize our previous work from [3,4]. We assume that a crowd-based application consists of several **tasks**; each task receives as input a list of **objects** (e.g., photos, texts, but also arbitrarily complex objects, with a **schema**) and asks the users to perform one or more **operations** upon them, which belong to a predefined set of abstract **operation types.** Examples of operation types from [3] are *Tag*, for assigning free keywords to items; *Classify*, for assigning each item to one or more classes; and *Sort* for ordering them. For instance, a task may consist in choosing the most interesting photos out of a collection, writing a caption for them, and then adding some tags. Note that this model is platform-independent. The progressive specification of crowdsourcing applications consists of the following phases:

1. **Task design** - deciding how a task is assembled as a set of operation types;
2. **Object design** - defining the set of objects;
3. **Workplan design** - Defining how a task is split into micro-tasks, and how objects are assigned to them;
4. **Platform design –** Defining the platforms that will be involved in executing micro-tasks and how performers are invited to perform micro-tasks in each platform.

Note that several micro-task answers might be required to decide the results for each data object (e.g., by majority agreement); each micro-task can be executed on different platforms. A performer may be registered on several platforms (with different accounts) and can be part of several communities.

Most crowdsourcing systems only provide limited and predefined controls; in contrast, our approach provides fine-level, powerful and flexible controls through active rules which are formally defined in [4] and whose properties (e.g., termination) can be proven in the context of a well-organized computational framework. Control design consists of four activities:

1. **Object control** is concerned with deciding when and how responses should be generated for each object.
2. **Performer control** is concerned with deciding how per- formers should be dynamically selected or rejected, on the basis of their performance.
3. **Task control** is concerned with completing a task or re-planning task execution.
4. **Adaptation and interoperability control** is concerned with determining which platforms should be involved with each tasks and specifically how micro-tasks can be dynamically adapted or reallocated.

In previous work, we focused on object, performer, task control [4,14]. In this paper, we focus on

adaptation and interoperability control, which are crucial to taking full advantage of multi-platform crowd applications.

Crowd programming approaches usually rely on imperative programming models to specify the interaction with crowdsourcing services; one interesting exception is given by CrowdWeaver, a system for visually manage complex crowd work [5]. AutoMan [6] is integrating human computations with Scala, providing a rich variety of options for adaptive quality control, although expressed within a low-level programming style. Important contributions addressing adaptation within specific aspects of crowdsourcing applications were given, e.g. by adaptive methods for determining cost-time tradeoffs in finding top-k objects out of a large dataset [7] or by dynamically deciding strategies based on result quality [8]. Recent works propose approaches for human computation which are based on high level abstractions, sometimes of declarative nature, including Qurk [13], CrowdDB [10], DeCo [11], CrowdLang [12] also interleave declaratively specified computations with crowd-based computations.

In designing our system, we have been inspired by several applications of human computation. Among them, [9] compares seven strategies for improving the quality and diversity of worker-generated results and [15] compares some alternatives for involving Mechanical Turk users in terms of their cost and quality.

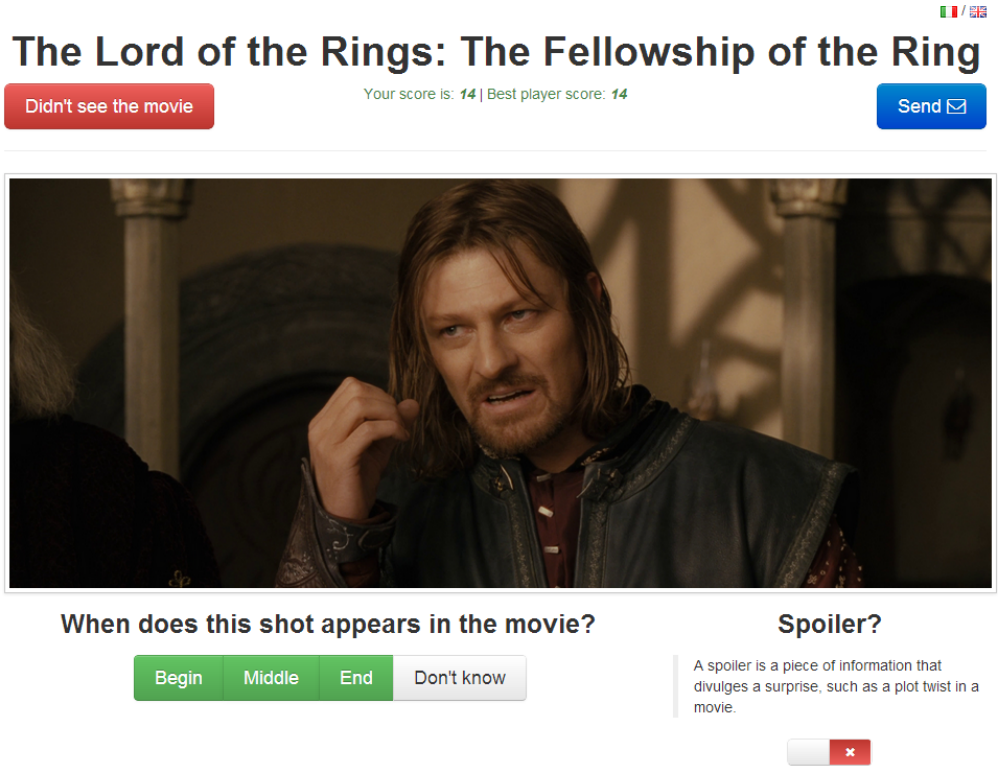

Figure 1: Customized UI used in the experiment; users can select the screenshot timeframe and whether it is a spoiler or not.

3. Running example

Given a set of still images taken from a movie, we asked performers to classify the image as belonging to

the beginning, middle or final part of the movie. Furthermore, we asked to explicitly say if the image could be a spoiler for the movie, i.e., an anticipation of the plot that ruins the enjoyment of the movie. Given the peculiarity of the task, we let the performers declare that they had not seen the movie or that they did not remember the specific scene, if this is the case. The experimental setting was as follows:

- **Dataset:** We selected 16 popular movies and we captured 20 still images from each. For time positioning, we recorded the timestamp of the capture, we split the movie in three slots and we automatically assigned each capture to the corresponding slot. We 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 consisted of evaluating one image. A customized, double language UI (Italian and English) is presented within Crowdsearcher (as shown in Figure 1). Results are accepted when an agreement between 5 performers is reached independently on the number of executions.

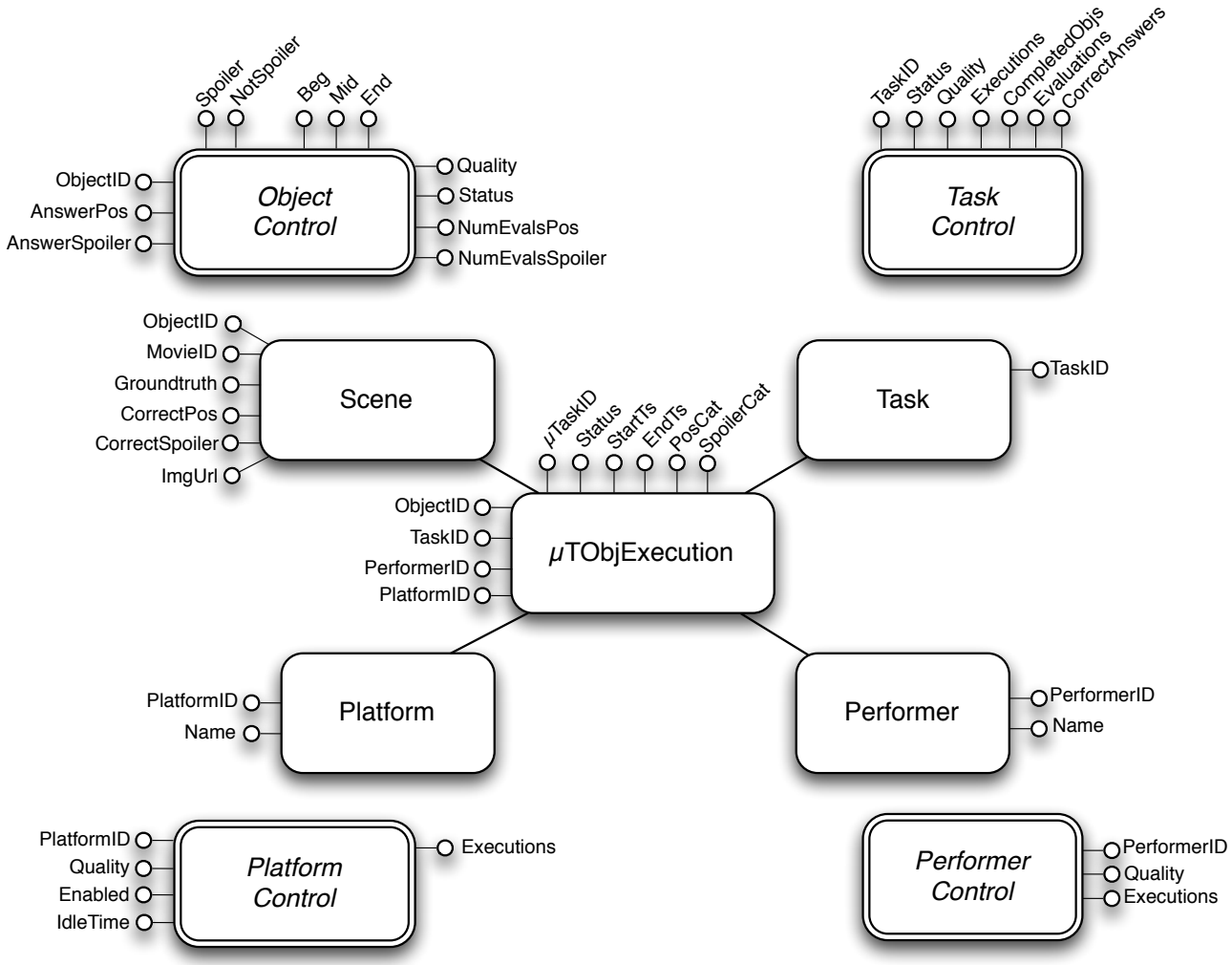

Figure 2: Control Mart example.

For enabling crowd control, we rely on control marts. A **control mart** is a data structure for monitoring the execution of a crowdsourcing task. Control marts are analogous to data marts used for data warehousing, as they exhibit a central entity representing the relevant facts associated with various entities representing control dimensions.

Figure 2 shows the control mart for our scenario. 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 control; such construction was introduced in [4] relative to the first three controls, and is hereby extended for platform control. The *microTObjExecution* entity stores the information about time of execution and the response given by the performer for a specific object (in this case, the position of the image in the movie and the spoiler flag). Each instance is connected to one task, one performer, one object and one platform; thus, it is identified by the tuple of identifiers of those entities. Note that each *microTask* is associated with several instances of *microTObjExecution*, one for each object associated with the *microTask*. Note also that some scenes belong to the *groundtruth* and as such are associated to a correct position and spoiler classification; these scenes are used for quality control. *ObjectControl*, *PerformerControl*, *TaskControl* and *PlatformControl* are convenience objects that describe aggregate properties whose values are updated by control rules during execution. In particular, the *PlatformControl* entity has attributes describing the number of executions and their precision; the *Enabled* flag indicates if the platform is currently enabled.

4. Adaptation rules

Interoperable crowd-based applications are deployed over multiple crowdsourcing platforms; executions can be enabled or disabled independently on each platform. We call **mapping** the assignment of executions to platforms. **Adaptation** is any change of the mapping; it occurs at a given, instantaneous time, called **adaptation time**, when the **adaptation condition** is checked and evaluated as true. It may require **replanning** (the process of generating new micro-tasks) and **reinvitation** (the process of generating new invitations for those micro-tasks). Changes of mappings and re-planning or re-invitation are dynamic changes of the application's configuration, captured by event-driven active rules.

We have designed two kinds of active rules, respectively at high and low-level level of abstraction. High-level rules are produced by a user-friendly design tool and capture the most typical design situations, without aiming at exhaustively capturing all cases. Low-level rules can express arbitrary conditions and actions upon the control mart. In both cases, these active rules follow the Event-Condition-Action paradigm. An active rule is triggered by an **event;** the rule's **condition** is a predicate that must be satisfied in order for the action to be executed; the rule's **actions** change the content of the control mart and/or invokes functions (such as "migrate").

We decided to use a rule based approach because: most of the rules can be automatically generated by the system, rule definition applies separation of concerns and thus changes in the requirements can be addressed by modify specific rules while preserving the rest; rules can be programmed to support arbitrarily complex controls [4].

4.1 High-level Rules

High-level rules apply to Objects, Tasks, Performers, and Platforms; each of them is associated with simple parameters, such as Number of executions and Quality, which are computed by formulas taking into account the current answers (e.g., for a classification operation, the number of answers that are in agreement relative to the total number of 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). Then, conditions are Boolean expressions of simple predicates upon these parameters; actions depend on each specific object and include: REDO or CANCEL for an object, RE-INVITE, RE-PLAN for a task, BAN for a performer, MIGRATE for a platform. For instance, the following high-level rule HR1 calls for the ban of a given performer:

HR1: **ON** PERFORMER **WHEN** QUALITY<50%
DO BAN

Adaptation supported by high-level rules also covers re-inviting and re-planning a task, or migrating it from a platform to another, as a whole. Rule HR2 calling for the migration of execution platform is:

HR2: **ON** OBJECT **WHEN** EXECUTIONS > 500 AND QUALITY < 60%
DO MIGRATE(AMT)

Figure 3 shows the interface that our tool offers to designers for expressing a rule that migrates an application from the current platform (a social network) to Amazon Mechanical Turk after 10 minutes of idle time.[1]

[1] The tool can be inspected at the URL: http://demo.search-computing.com/cs-demo/ . A demonstration video is available too: https://www.youtube.com/watch?v=9ZAQCqAfwzA

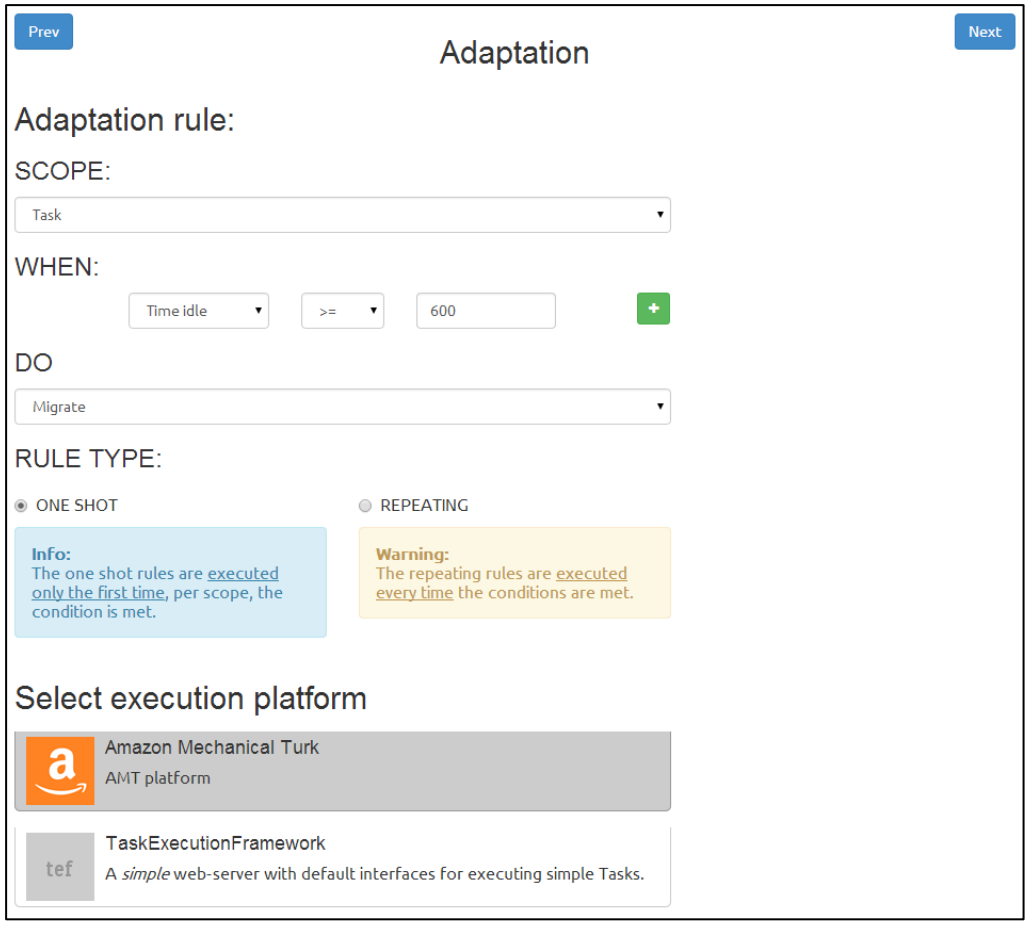

Figure 3 – Example of high-level adaptation rule (platform migration) supported by the tool

4.2 Low-Level Rules

High-level rules have a limited expressive power, due to their simplified syntax; low-level rules are instead capable of expressing all the reactive behaviors supported by control marts. We recall from [4] that:

- Rules can be triggered by classical data updates and by absolute or periodic events.
- Rules are at row-level granularity. Variables NEW and OLD denote the before and after state of each row.
- Special selector formulas are used to express sub-queries synthetically; thus,
  TABLE[<predicate>].ATTRIBUTE extracts the same values as
  SELECT ATTRIBUTE FROM TABLE WHERE <predicate>.
- The rule language includes functions (e.g., for re-planning of micro-tasks or re-invitation of performers).

The process of generation of low-level rules from high-level rules is as follows. Several rules exist in the background to compute the current values of the control entities, which contain aggregate properties about objects, tasks, performers, and platforms. Thus, entity-specific parameters (e.g. object's executions

and quality) discussed in the previous section are updated at each micro-task execution; in particular, parameters such as quality are defined on the basis of the specific method for guaranteeing the convergence of computations, e.g. majority voting for classification tasks. For instance, the following background rules LR1, LR2 and LR3 contribute to computing the parameters required by rule HR2. LR1 tracks the answers that classify a scene as spoiler, LR2 closes the state of an object when it has 5 agreeing classifications as either spoiler or non-spoiler, LR3 computes the quality of closed objects based on the answers which agree with the gold truth. Of course many other rules exist (e.g. for classifying scenes relative to their position, for counting executions by object and by task) and the rules below have some simplifications (e.g. closed objects must also be classified for their position in the movie).

```
LR1 e: UPDATE FOR M_T_O_EXECUTION[SpoilerCat]
       c: NEW.SpoilerCat ==  'Spoiler'
       a: SET Object_CTRL[oid == NEW.oid].Spoiler += 1

LR2: e: UPDATE FOR Object_CTRL
        c: (NEW.Spoiler== 5) or (NEW.NotSpoiler == 5)
        a: SET Scene[oid==NEW.oid].CorrectSpoiler = NEW.AnswerSpoiler,
           SET Task_CTRL[tid==NEW.tid].CompObj += 1
           SET Object_CTRL[oid==NEW.oid].Status = 'closed'

LR3: e: UPDATE FOR Task_CTRL[CorrectAnswers]
        a: SET Task_CTRL[TaskID == NEW.TaskID].Quality =  Task_CTRL[TaskID ==
           NEW.TaskID].CorrectAnswers /  Task_CTRL[TaskID == NEW.TaskID].Evaluations
```

Each high-level rule is translated into one or more low-level rules, which is added to background rules and possibly to other custom adaptation rules. Rule HR2 is translated by rule LR4, which is activated when quality is below threshold and executions are above threshold, and calls for a migration of the task:

```
LR4: e: UPDATE FOR Task_CTRL
        c: NEW.PlatformID='Facebook' and NEW.Quality < 0.6 and NEW.Executions > 500
        a: migrate(NEW.TaskID,'AMT')
```

4.3 Switchover options

Platform-level adaptation requires a **switch-over**, which involves detached and joining systems: **detached systems** participate to the application prior to the adaptation and become not involved afterwards. A switch-over can be continuous, instantaneous, or reset:

- With a **Continuous Switch-Over**, results of initiated tasks of detached systems contribute to the application's outcome, even if they are produced after the switch over. This strategy should be used if the results up to now are acceptable.

- With an **Instantaneous Switch-Over**, results of initiated tasks of detached systems produced before the adaptation are considered, while results after the adaptation are disregarded. This strategy has a bigger impact on the quality of the results (bad performer from the detached system can not participate anymore) but can negatively influence the completion time of the campaign.

- With a **Reset Switch-Over**, all the results from detached systems are disregarded. It can increase the quality of the results if the detached system was performing badly and can hurt the completion time of the task depending on the speed of the new platform.

By using these options, the contribution of detached systems over results can be reduced or eliminated.

High-level rules perform a continuous switch over; instantaneous switch-over and reset switch-over can be programmed by custom low-level adaptation rules.

Rule LR5 implements a reset switchover which occurs when the detached system is Facebook and a specific object has already all counters Beg, Mid, and End greater than 1 (as an indication that it has been poorly classified); the action sets to zero the Beg, Mid, and End counters (re-initialization of other attribute values is omitted for brevity), sets all microtask object executions for that object to invalid, then replans the object to AMT and invites performers for the new micro-tasks.

```
LR5:  e: UPDATE for Object_CTRL
        c: NEW.Beg >= 1 AND
           NEW.Mid >= 1 AND
           NEW.End >= 1 AND
           Platform [ObjectID == NEW.ObjectID].Name = 'Facebook'
        a: SET NEW.Beg == 0,
           SET NEW.Mid == 0,
           SET NEW.End == 0,
           FOREACH e IN M_T_O_EXECUTION [ObjectID == NEW.ObjectID] SET e.Status = 'invalid',
            replan(NEW.ObjectID,'AMT'),
            reinvite(NEW.ObjectID,'AMT')
```

5. Experiments

We considered 4 platforms for inviting performers (email, Twitter, Facebook, and Amazon Mechanical Turk); we redirected all the performers to the customized UI shown in Figure 1. The first round of invitations was sent to the mailing lists of students of 8 courses; subsequently, we engaged people on Twitter, then Facebook, and finally we posted HITS on AMT, with a payment of 1 cent per execution. The change of platform was performed using a continuous switch-over every 2 days. As the diagram shows a

plateau well before each switch-over, each platform-specific portion of the diagram can be considered as baseline of execution without adaptation. We closed evaluations when agreement of five votes was reached both for the positioning in time (beginning, mid, end) and for spoiler labeling (yes, no).

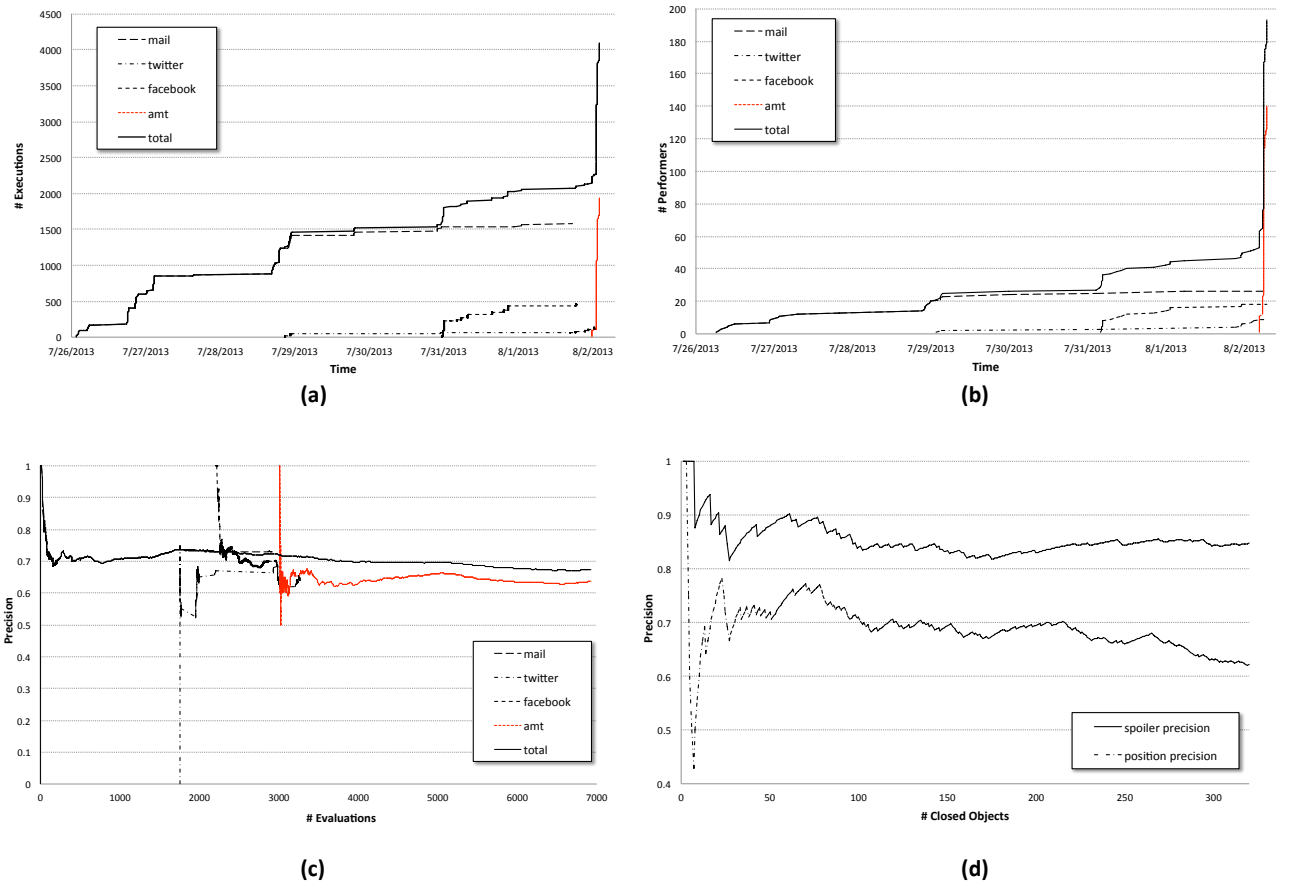

Figure 4: number of executions (a) and performers (b) by platforms; Precision of evaluations by platform (c); and precision on closed objects after majority is applied (d).

Figure 4 shows the number of executions (a) and of performers (b) for each platform. The graphs show substantial participation of people invited via email, invitations to friends using Twitter were less popular, while invitations using Facebook were more popular – this indicates that Facebook-based social networks may be suggested for performing tasks which are suited for mobilizing friends rather than work colleagues. We also noticed some spontaneous re-invitations, as some tasks were executed by people not included within the initial invitation lists (thus, some invited people in turn invited other people). When we turned to a crowdsourcing platform, performers on AMT closed the experiment in few hours.

Figure 4 (c) shows the aggregate precision of performers 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. 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.

This effect can be seen also on Figure 4 (d), which shows the precision of the results after applying the majority, by keeping the two operations (scene positioning and spoiler detection) separate. The system's precision (of Fig. 5) is computed after the agreement on five evaluations, hence it is higher than precision by worker (of Fig. 4); 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 does not change much with time – hence, workers on AMT platform perform more or less like all other users; while the precision of image positioning reduces with time, both because the last scenes to be closed are most difficult to classify and because AMT users become more numerous and they exhibit a lower individual precision.

6. Conclusions

Achieving platform interoperability is a big step forward for crowd-based applications, as each platform provides specific qualities. On the other hand, instrumenting and programming applications over multiple platforms is hard, as each platform provides APIs which are only suitable for imperative programming. Crowdsearcher supports a declarative style for describing interoperability requirements and a rule-based, flexible implementation on top of well-designed data structures to implement such requirements.

References

[1] E. Law and L. von Ahn. Human Computation. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2011.

[2] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. Commun. ACM, 54(4):86–96, Apr. 2011.

[3] A. Bozzon, M. Brambilla, and S. Ceri. Answering search queries with crowdsearcher. In 21st Int.l Conf. on World Wide Web 2012, WWW '12, pages 1009–1018. ACM, 2012.

[4] A. Bozzon, M. Brambilla, S. Ceri, and A. Mauri. Reactive crowdsourcing. In 22nd World Wide Web Conf., WWW '13, pages 153–164, 2013.

[5] D. W. Barowy, C. Cutsinger, E.D. Berger, A. McGregor. AutoMan: a platform for integrating human-based and digital computation. Proceedings of the OOPSLA 2012, 639-654, ACM.

[6] A. Kittur, S. Khamkar, P. André, R. Kraut. CrowdWeaver. Visually managing complex crowd work. CSCW 2012, 1033-1036.

[7] A. Das Sarma, A. Parameswaran, H. Garcia-Molina, A. Halevy. Crowd-Powered Find Algorithms. Proc. ICDE 2014, 964-975.

[8] A. Marcus, E. Wu, D. Karger, S. Madden, R. Miller, Human-Powered Sorts and Join, PVLDB 5:0, 2011.

[9] W. Willett, J. Heer, and M. Agrawala. Strategies for crowdsourcing social data analysis. In SIG-CHI Conf. on Human Factors in comp. sys., pages 227–236. ACM, 2012.

[10] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In ACM SIGMOD 2011, pages 61–72. ACM, 2011.

[11] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: A system for declarative crowdsourcing. PVLDB, 5(12):1990–1993, 2012

[12] P. Minder and A. 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.

[13] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In CIDR 2011, pages 211–214. www.cidrdb.org, Jan. 2011.

[14] A. Bozzon, M. Brambilla, S. Ceri, A. Mauri, R. Volonterio . Pattern-Based Specification of Crowdsourcing Applications. Web Engineering, Springer LNCS Vol. 8541, pages 218-235, 2014.

[15] A. Kittur, E. H. Chi, and B. Suh. Crowdsourcing user studies with Mechanical Turk. In SIG-CHI Conf. on Human factors in comp. sys., pages 453–456. ACM, 2008.

Short Bios

Stefano Ceri is professor of Database Systems at the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) of Politecnico di Milano. He was visiting professor at the Computer Science Department of Stanford University (1983-1990), chairman of the Computer Science Section of DEIB (1992-2004), chairman of LaureaOnLine in Computer Engineering (2004-2008), director of Alta Scuola Politecnica (ASP) of Politecnico di Milano and Politecnico di Torino (2010-2013), co-founder (2001) and shareholder of WebRatio (www.webratio.com). He wrote over 250 journal and conference articles. He is co-editor in chief of the book series “Data Centric Systems and Applications” (Springer-Verlag). He is author of three US Patents and ten books in English. He is the recipient of the ACM-SIGMOD “Edward T. Codd Innovation Award” (New York, June 26, 2013), an ACM Fellow and member of the Academia Europaea. His research interests include database technologies, web engineering methods, crowdsourcing and genomic data analysis. Contact him at Stefano.Ceri@polimi.it

Marco Brambilla is assistant professor at the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) of Politecnico di Milano. He graduated cum laude in 2001 and got his Ph.D. in Information Engineering in 2004. He collaborated as application analyst in several industrial projects; among others, he worked with Acer Europe, Cisco System (San José, CA, USA), and WebRatio. He has been visiting researcher at UCSD, Cisco San José, and Dauphine University Paris. He has been PC chair of ICWE 2012 in Berlin and organizer of several conference tracks and workshop. His research interests include theoretical, experimental, and methodological aspects related to Web modeling methodologies, Web engineering, information seeking and crowdsourcing. He is shareholder and scientific advisor of WebRatio. He is the leader of the standardization initiative of IFML within OMG. He is coauthor of 3 international books and of more than 150 scientific papers. Contact him at Marco.Brambilla@polimi.it.

Andrea Mauri is PhD student at the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) of Politecnico di Milano.
His main research topics are Crowdsourcing and Human Computation, with a particular focus on studying new methodologies for developing crowd and social based applications. He has a master's degree in computer science from Politecnico di Milano. Contact him at Andrea.Mauri@polimi.it.

Riccardo Volonterio is Research Assistant at the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) of Politecnico di Milano.
His research interests include Crowdsourcing and Human Computation. He has a master's degree in computer science from Politecnico di Milano. Contact him at Riccardo.Volonterio@polimi.it.

Mailing information of all authors:

Dipartimento di Elettronica, Informazione e Bioingegneria

Politecnico di Milano

Piazza L. Da Vinci 32

20133 Milano, Italy

Stefano Ceri

Phone: 02 2399 3532

Fax: 02 2399 3411

Email: marco.brambilla@polimi.it

Marco Brambilla

Phone: 02 2399 7621

Fax: 02 2399 7321

Email: marco.brambilla@polimi.it

Andrea Mauri

Phone: 02 2399 7620

Fax: 02 2399 7321

Email: andrea.mauri@polimi.it

Riccardo Volonterio

Phone: 02 2399 7620

Fax: 02 2399 7321

Email: riccardo.volonterio@polimi.it