A Model-Driven Approach for Crowdsourcing Search

Alessandro Bozzon, Marco Brambilla, Andrea Mauri
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
{name.surname}@polimi.it

ABSTRACT

Even though search systems are very efficient in retrieving world-wide information, they can not capture some peculiar aspects and features of user needs, such as subjective opinions and recommendations, or information that require local or domain specific expertise. In this kind of scenario, the human opinion provided by an expert or knowledgeable user can be more useful than any factual information retrieved by a search engine.

In this paper we propose a model-driven approach for the specification of crowd-search tasks, i.e. activities where real people – in real time – take part to the generalized search process that involve search engines. In particular we define two models: the "Query Task Model", representing the meta-model of the query that is submitted to the crowd and the associated answers; and the "User Interaction Model", which shows how the user can interact with the query model to fulfill her needs. Our solution allows for a top-down design approach, from the crowd-search task design, down to the crowd answering system design. Our approach also grants automatic code generation thus leading to quick prototyping of search applications based on human responses collected over social networking or crowdsourcing platforms.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search Retrieval—*Search Process*

Keywords

crowdsourcing, social network, model driven development.

1. INTRODUCTION

While search systems are superior machines to get world-wide information, people tend to put more trust in people than in automated responses. That is why often users seek for opinions collected within friends and expert/local communities for taking an informed decision about significant issues. Other users' opinions can ultimately determine our decisions. While in the past people could rely on opinions given by close friends on local or general topics, the change in the social connections in our society makes users increasingly rely on online social interaction to complete and validate the results of their search activities. People often search for human help in between canonical web search steps: they first query a search system, then they ask for an opinion on the result, maybe they also ask suggestion on the query term. We define this trend as *crowd-searching*.

In current Web systems, the crowd-search activity, i.e. looking for opinion from friends or experts, is detached from the original search process, and is often carried out through different social networking platforms and technologies. Moreover, people manages different applications, different virtual identities and maybe also different devices: they send email, ask on Twitter, Facebook or other social network, or ask to friend and people they know.

Recent works (see section 4) on crowd-based search focus on simple and atomic task, while crowd-sourced search involve a wide range of scenario, from trivial decisions, like choosing where going to eat at dinner, to more serious things like organizing a travel or even buying a house. Thus the user need a way to manage and control the whole process, from the creation of the query, the selection of the target to the gathering of the results.

In this paper we propose a model-driven, platform independent, approach to design Web applications that support crowd-sourced search. We define a top-down design approach, as sketched in Figure 1 which applies model-driven engineering (MDE) techniques for the specification of the crowd-sourced information collection task, its splitting and refinement, and its mapping to the Web user interaction

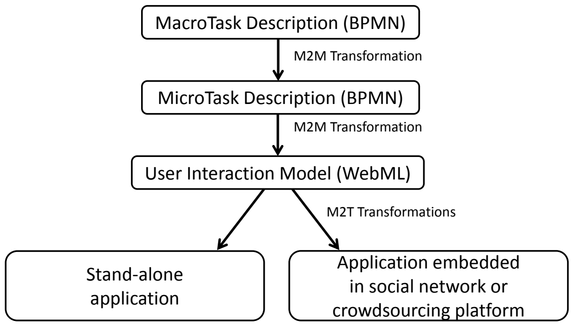

MacroTask Description (BPMN)
M2M Transformation
MicroTask Description (BPMN)
M2M Transformation
User Interaction Model (WebML)
M2T Transformations
Stand-alone application
Application embedded in social network or crowdsourcing platform

Figure 1: Overview of our approach.

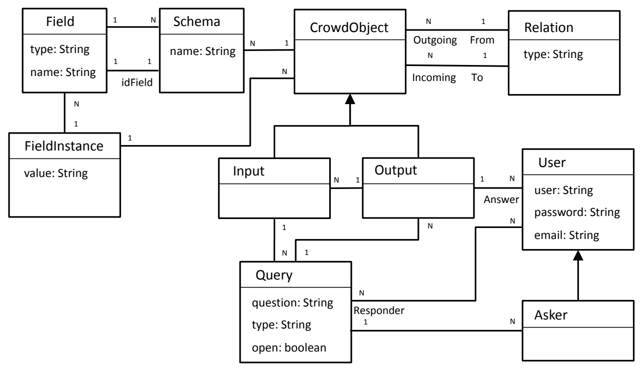


Figure 2: The query task meta-model.

specification. The approach starts from the task description and applies model-to-model transformations to build the detailed task definitions (described e.g. in BPMN) and then the platform independent user interaction model (described in the domain-specific language WebML[3, 9]). Then the final application is automatically generated by means of a model-to-text code generation transformation.

The main ingredients that participate to our contribution are: 1) a metamodel of the crowd-sourced question; and 2) the models of the user interfaces needed for defining the questions and for responding. In this short paper we focus on the aspects related to the model-driven design of the crowd-search user interactions, spanning from the question definition to the engagement, dissemination, and ending in the response submission and collection. On the other side, we consider the task refinement and redesign problem as outside the scope of this short work.

The paper is organized as follows. Section 2 and Section 3 respectively describe our search task meta-model and user interaction model; Section 4 summarizes the related works for both the crowd-sourcing and the model-driven fields; and finally, Section 5 concludes.

2. TASK MODEL

The starting point of our MDE approach is the query task model. Figure 2 shows the query task meta-model to which every query task should conform. The main element is the *Query* submitted by a *User*. The *Query* is defined by a *Question*, written in natural language, and a list of *CrowdObjects*, i.e. information structured according to a given schema.[1]

A question includes a set of *Input CrowdObjects*, i.e., a set of data in the user's question upon which the responder can apply his response. For example, if the user wants to collect opinions about some restaurants in Lyon, the *Input CrowdObject* instances comprise the restaurants subject to the comparison. The input object can be either inserted manually at query creation time by the user or extracted from a previous search (both canonical or crowd-based) step. The model of these objects is defined by the *Schema* element. Input objects are not mandatory for the creation of a query, as a user can create an open question. However, we always assume the presence of a *Schema*.

[1]To ease the discussion, we assume that information is structured in relations; however, other formats (e.g. semi-structured, graph, etc.) are also suitable.

The *type* of the query defines how a user can answer to the question. These have been classified in a taxonomy [6] comprising among others the following task types:

- **Like**: the user answers the query by voting ("liking") one or more of the query inputs;
- **Comment**: the user answers the query by writing a comment on one or more of the query inputs;
- **Add**: the user answers the query by adding one or more new instances of *Output CrowdObject*.

Finally, the *Query* is also related with a set of *Output CrowdObjects*, representing the answers to the question submitted by the crowd.

Users of a crowd-search task can be classified into two categories: *askers* and *responders*. The former is the user using the platform and creating questions to be submitted to the crowd, while the latter is a user involved in the query answering process using the social network or crowdsourcing platform.

Relation represents associations that can exist between *CrowdObjects*. These relations can be either an *Input-Input* relation or a *Output-Input* relation. They are created when a query is split into sub-queries and depend on the kind of splitting pattern that is applied. Indeed, starting from the design of the coarse-grained task, one can refine its description by structuring its activities according to known crowd-interaction patterns (ie.g., find-fix-verify [5], map-reduce [11], or Turkomatic guidelines [12]).

Input-Input relations occur when the initial set of inputs is partitioned across different instances of the same query, to reduce the workload of the responder. For example, if the original query would ask the responder to order one hundred restaurants, it can be useful to split the task into subtask of ten restaurants each, to be assign to different responders. In this case the task performed by each responder is the same, but it is applied on different sets of objects. The initial set can be is therefore partitioned into the different query instances, according to different strategies (e.g. , in a uniform way or according some properties of the input instances). The input of the new query instances are thus mapped to the inputs of the original query, according e.g. to a map-reduce pattern [11].

Output-Input relations occurs when the task requested by the author of the query is complex or difficult, or if the result require some kind of validation, which therefore requires organizing the task into a sequence of subtasks [5]. In this case the query is composed by several heterogeneous tasks, and each user performs a particular one (e.g., according to a find-fix-verify or similar pattern [5]). Hence, the output of the first subtask becomes the input for another query subtask, and so on, thus generating an output-input mapping.

3. USER INTERACTION MODEL

The user interaction model describes the interface and navigation aspects of the crowdsearch application. Starting from the query task model, possibly split in a complex pattern of microtasks, a model transformation can lead to a coarse *user interaction model*, which in turn can be manually refined by the designer. The user interaction must cover three fundamental phases of the crowdsearch process:

- the submission of the question (performed by the asker);

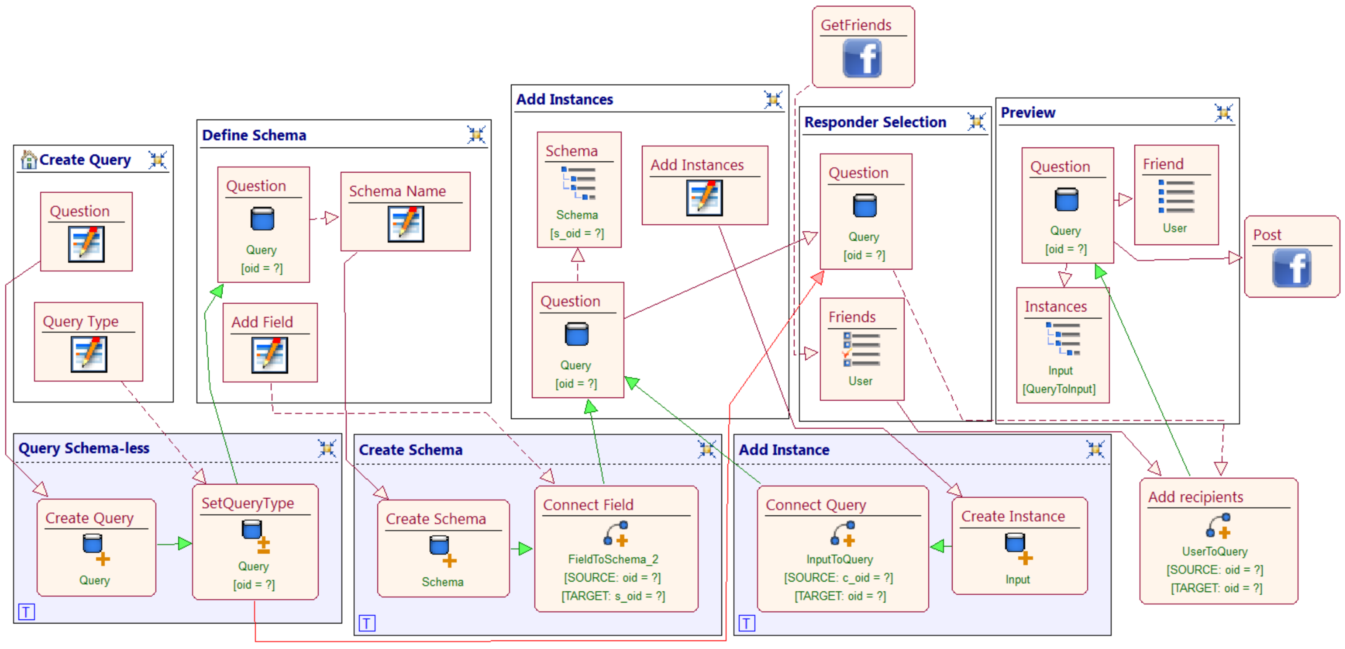


Figure 3: User interaction model for creating a query.

- the collection of the responses (performed by the responder);
- and the analysis of the results (available to the asker for getting insights).

At the current stage, our research has identified the interaction patterns relevant for each phase, considering the various options of deployment platform, task type, and macro-task splitting pattern. For space reasons, in this section we report one possible outcome of the user interaction design, in case of simple query task and of deployment on the Facebook social networking platform. We describe the phases of query creation and of query answering, according to the WebML notation [3].

3.1 Query creation

Figure 3 shows the user interaction model for creating and submitting a query, according to the WebML notation. In the *Create Query* page, the user specifies the textual question (e.g., "What's the best museum to visit in Milan?") and sets the query type (e.g., "Like", "Add", and so on). The user can also choose the type "open question", thus assuming that no items are needed in input for the responder to select/like and so on. In both cases, an *Query* instance is created, and its type is set. If the query does not have inputs, then the user is directly brought to the *Responder Selection* page.

If, on the other hand, the user chooses to build a structured question with inputs, then he is redirected to the "Define schema" page, where the asker can create a schema for inputs by assigning a general name to the input type and by defining its attributes in terms of name and type. By submitting the form, the application creates a new instance for the *Schema* entity and its associated *Fields*. The asker is brought to the *Add Instance* page, where he can add input objects following the schema previously defined. The specified instances of the *Input* are created and linked to the query.

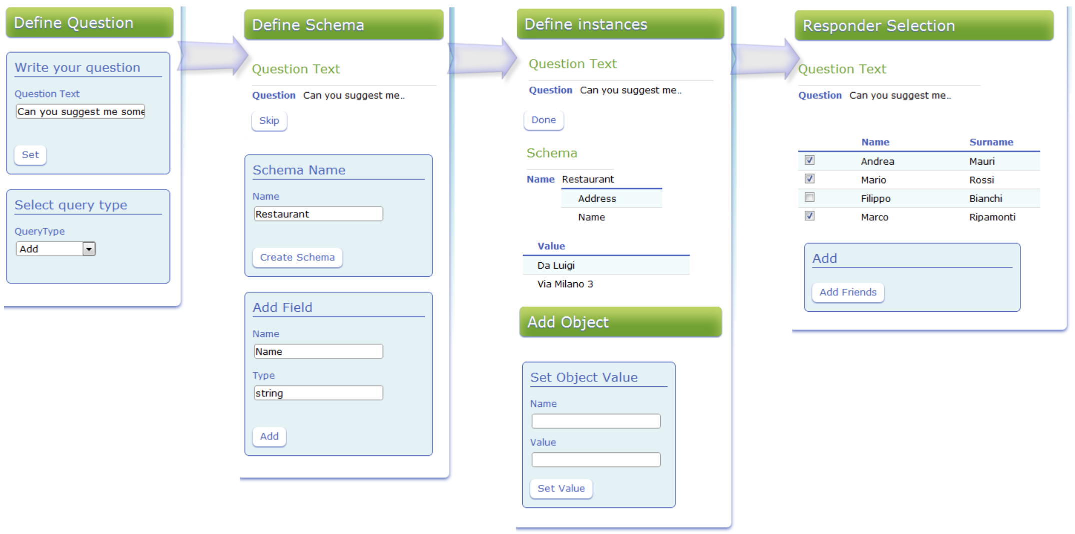
Finally, in the *Responder Selection* page the asker can select the responders to the query: the list of possible responders is retrieved from the social network or crowdsourcing platform (in this example, the *GetFriend* component collects the friends from the Facebook platform). The user can select the responders through the "Friends" multi-selection list in the page. Eventually, after viewing a preview of the created question, the user can post the query on the social platform.

Figure 4 shows a compressed view of the web pages that are produced starting from the user interaction models described in Figure 3, thanks to the code generation facilities of WebRatio. The structure and content of the pages can be easily recognized and mapped to the corresponding model elements. In this particular example the user wants to know some good restaurants in Milan. Hence she defines the question "Can you suggest me some good restaurants in Milan?" and selects the "Add" query type. Then she creates the schema the input instances of the question must conform to. In the instance list she adds a restaurant she already knows. Finally she selects the recipients of the question from the list of her friends extracted from Facebook

3.2 Answering to a query

Figure 5 depicts the WebML model for the query answering activity, performed by a responder based on the query structure defined by the asker. When accessing the application through the *Responder Dashboard* page, the responder is presented with a list of questions to answer. By clicking on a question, he is brought to the *Details* page, where he can provide his answer. The page shows the question text, plus the set of defined input instances (*Input* component in the *Details* page).

Depending on the type of the question (defined by the asker defined during the query creation phase), different



Define Question

Write your question

Question Text: Can you suggest me some

Set

Select query type

QueryType: Add

Define Schema

Question Text

Question: Can you suggest me..

Skip

Schema Name

Name: Restaurant

Create Schema

Add Field

Name: Name

Type: string

Add

Define instances

Question Text

Question: Can you suggest me..

Done

Schema

Name: Restaurant — Address, Name

Value

Da Luigi

Via Milano 3

Add Object

Set Object Value

Name:

Value:

Set Value

Responder Selection

Question Text

Question: Can you suggest me..

	Name	Surname
✓	Andrea	Mauri
✓	Mario	Rossi
☐	Filippo	Bianchi
✓	Marco	Ripamonti

Add

Add Friends

Figure 4: Rendering of the Web pages implementing the query creation phase, as generated by WebRatio.

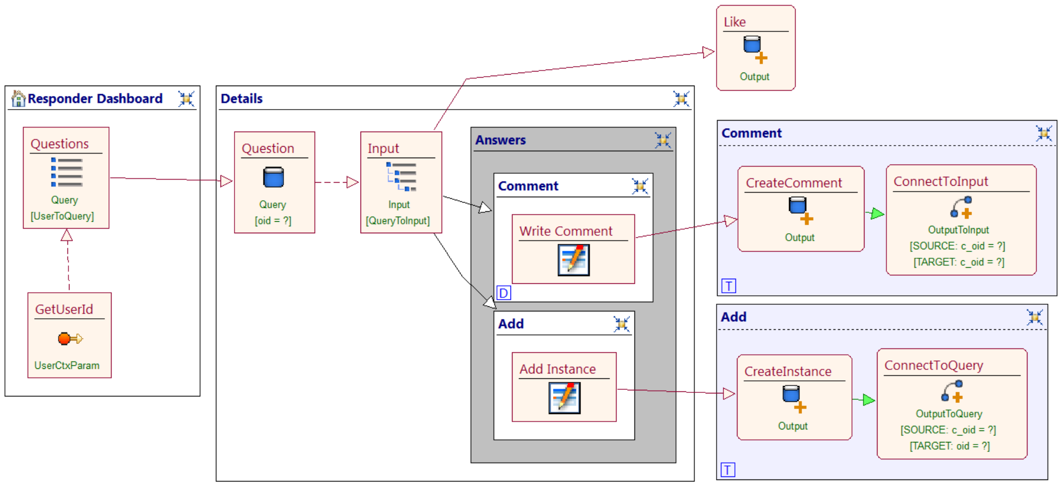


Figure 5: Hypertext model for answering to a query.

concrete user interfaces can be shown: in the case of a "Like" question, the responder simply selects the preferred instances in the *Input* list; as a consequence, a set of Output objects are created corresponding to the "likes" of the user.

In the case of "Comment" or "Add" question, the user is shown a form to respectively write a comment or add a new instance to the list. In the case of "Comment" questions, an *Output* object with the comment schema (i.e. a single textual field) is created. The "Add" case is more noteworthy, as the *Output* objects will present a schema equivalent to the *Input* ones, so to add the new object instances to the list of input object of the query.

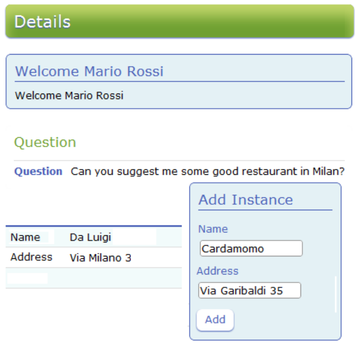
Figure 6 shows the compressed view of the "Details" page built from the user interaction model described in Figure 5. Continuing the previous example, in this page the responder of the question can add additional restaurants he knows.

4. RELATED WORKS

This work falls into the broad field of human computation, i.e., the discipline that aims to use human knowledge to fulfill tasks that are difficult or even impossible for a machine. For example, human computation studies have been done about using crowd's knowledge for image recognition [15], to answer ambiguous queries[6] or to refine incomplete data [10, 14]. The platforms most adopted for exploiting human knowledge and skills are based on crowdsourcing (the most prominent example being Amazon Mechanical Turk [1]). However, other ways of collecting human intelligence can be exploited, such as social networks.

A very important aspect of a crowd-based search is the quality of the results and the response time, therefore several works have addressed the problem of understanding how design features (for example the cost of the task) impacts on these results metrics [13, 4].

The novelty aspects of our approach with respect to the existing works include: independence with respect to the crowdsourcing platform (in particular, we allow to exploit indifferently a social network or a crowdsourcing marketplace



Welcome Mario Rossi

Welcome Mario Rossi

Question

Question Can you suggest me some good restaurant in Milan?

Name	Da Luigi
Address	Via Milano 3

Add Instance

Name: Cardamomo

Address: Via Garibaldi 35

Figure 6: Rendering of the query answering Web page, as generated by WebRatio.

of choice); model-driven design of tasks and user interactions; model-transformation based approach that partly automates the generation of some models, thus reducing the cost of designing new applications; and possibility of manually or automatically choosing the responders to a query task. Our work can be seen as an extended social question answering approach (as applied in Quora and other well known platforms), where the asker has greater flexibility in defining and sharing his questions. Our work addresses the problem of defining crowdsourcing tasks at the modeling level, while existing approaches and tools typically allow for a programming approach to the problem (e.g., see TurkIt [2]).

Our work is based on general purpose model-driven techniques and on our previous work on Web application design [9], on mapping business processes to user interaction models [8], as well as on the preliminary results presented in the CrowdSearcher approach [6]. From the implementation perspective, we rely on the WebRatio toolsuite [7], which provides code generation facilities for WebML models.

5. CONCLUSIONS AND FUTURE WORKS

In this paper we presented a model-driven approach for crowdsourcing responses to questions. We defined a metamodel of the query taks and a user interaction model for building and answering to a query. We apply model-driven techniques to the design of the various aspects of the query tasks and to the transformations among them.

Ongoing activities are addressing the problems of task splitting and automatic model transformations, so as to implement a model-driven approach to the design of the tasks, considering the structured crowdsourcing patterns identified in literature. For the future we plan to extend the coverage of the deployment to several social and crowdsourcing platforms and integration of the potential responders base from several platforms at a time.

ACKNOWLEDGMENTS

This research is partially supported by the Search Computing (SeCo) project, funded by European Research Council, under the IDEAS Advanced Grants program; by the Cubrik Project, an IP funded within the EC 7FP; and by the BPM4People SME Capacities project. We thank all the projects' contributors.

6. REFERENCES

[1] Amazon mechanical turk https://www.mturk.com.

[2] Turkit http://groups.csail.mit.edu/uid/turkit/.

[3] Webml http://www.webml.org.

[4] D. Ariely, U. Gneezy, G. Loewenstein, and N. Mazar. Large Stakes and Big Mistakes. *Review of Economic Studies*, 75:1–19, 2009.

[5] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. 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.

[6] A. Bozzon, M. Brambilla, and S. Ceri. Answering search queries with crowdsearcher. In *Proceedings of the World Wide Web conference (WWW 2012)*, page in print, 2012.

[7] M. Brambilla, S. Butti, and P. Fraternali. Webratio bpm: A tool for designing and deploying business processes on the web. In B. Benatallah, F. Casati, G. Kappel, and G. Rossi, editors, *ICWE*, volume 6189 of *Lecture Notes in Computer Science*, pages 415–429. Springer, 2010.

[8] M. Brambilla, S. Ceri, P. Fraternali, and I. Manolescu. Process modeling in web applications. *ACM Trans. Softw. Eng. Methodol.*, 15(4):360–409, 2006.

[9] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing data-intensive Web applications*. Morgan Kaufmann, USA, 2003.

[10] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 61–72, New York, NY, USA, June 2011. ACM.

[11] A. Kittur, B. Smus, and R. Kraut. CrowdForge: crowdsourcing complex work. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, CHI EA '11, pages 1801–1806, New York, NY, USA, 2011. ACM.

[12] A. P. Kulkarni, M. Can, and B. Hartmann. Turkomatic: automatic recursive task and workflow design for mechanical turk. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, CHI EA '11, pages 2053–2058, New York, NY, USA, 2011. ACM.

[13] W. Mason and D. 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.

[14] A. Parameswaran and N. Polyzotis. Answering queries using humans, algorithms and databases. In *Conference on Inovative Data Systems Research (CIDR 2011)*. Stanford InfoLab, January 2011.

[15] T. Yan, V. Kumar, and D. Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 77–90, New York, NY, USA, 2010. ACM.