Modeling, Enacting and Integrating Custom Crowdsourcing Processes

STEFANO TRANQUILLINI, FLORIAN DANIEL, PAVEL KUCHERBAEV, FABIO CASATI

University of Trento, Italy

Abstract—**Crowdsourcing is the outsourcing of a unit of work to a crowd of people via an open call for contributions. Thanks to the availability of online crowdsourcing platforms, such as Amazon Mechanical Turk or CrowdFlower, the practice has experienced a tremendous growth over the last few years and demonstrated its viability in a variety of different fields, such as data collection and analysis or human computation. Yet, it is also increasingly struggling with the inherent limitations of these platforms: each platform has its own *logic* of how to crowdsource work (e.g., marketplace or contest), there is only very little support for *structured* work (work that requires the coordination of multiple tasks), and it is hard to *integrate* crowdsourced tasks into state-of-the-art business process management (BPM) or information systems.**

We attack these three shortcomings by (i) developing a *flexible crowdsourcing platform* (we call it Crowd Computer, CC) that allows one to program custom crowdsourcing logics for individual as well as structured tasks, (ii) devising a BPMN-based *modeling language* that allows one to program the CC intuitively, (iii) equipping the language with a dedicated *visual editor*, and (iv) implementing the CC on top of *standard BPM technology* that can easily be integrated into existing software and processes. We demonstrate the effectiveness of the approach with a case study on the crowd-based mining of mashup model patterns.

1. Introduction

Crowdsourcing (CS) is a relatively new approach to execute work that requires human capabilities. Howe [1], who coined the term, defines *crowdsourcing* generically as "the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call." In principle, work could therefore be outsourced in a variety of different ways, e.g., by temporarily recruiting volunteers to help complete a given job or by distributing questionnaires that people fill voluntarily. However, the environment that most contributed to the emergence and success of the practice is the Web, and we therefore specifically focus on crowdsourcing in the context of the Web and on work that is crowdsourced with the help of so-called *crowdsourcing platforms*. These are on-line brokers of work (in the form of web applications/services) that mediate between the *crowdsourcer* who offers work and the *workers* who perform work in exchange for a possible *reward*. The latter form the *crowd*. Prominent examples of crowdsourcing platforms

are Amazon Mechanical Turk (https://www.mturk.com), CrowdFlower (http://www.crowdflower.com/), or 99designs (http://99designs.com). The power of these platforms is their workforce, the crowd, which is potentially large, always available, and can be requested on demand, which makes workforce in crowdsourcing similar to computing power or data storage capability in cloud computing [2].

Hossfeld2011 statistically analyzed the growth of the crowdsourcing platform Microworkers from May 2009 to October 2010 and identified a square or a logistic growth model for its crowd. Concretely, according to a recent white paper by [3], the enterprise crowsourcing market experienced a 75% growth in 2011, after a 53% growth in 2010, and was expected to even double in 2012. The people engaged as workers in crowdsourcing platforms span the whole globe, and their number increased by 100% in 2011 and was expected to grow even more in the years to follow. This is confirmed by the recent first Annual Survey among enterprise leaders and entrepreneurs by the 2014 Global Crowdsourcing Pulsecheck [4] that reports that about 16% of respondents think that crowdsourcing will grow "substantially more than current growth," 38% say it will grow "more than current growth," 41% say it will grow "same as current growth," while only 5% think it will grow "less then current growth." Likewise, the number and type of available crowdsourcing platforms has enormously grown over the last years, summing up to hundreds of platforms as of today (http://www.crowdsourcing.org/directory) and covering areas like Internet services, media and entertainment, technology, manufacturing, financial services, travel and hospitality, and so on. The key benefits crowdsourcers identify are productivity, flexibility and scalability, cost savings, predictable costs, and better time to market [3].

Typical tasks outsourced on today's crowdsourcing platforms are so-called *micro-tasks* like tagging pictures or translating receipts and *creative tasks* like designing a logo or implementing a piece of software. Each platform approaches these tasks following its very own *tactic*, i.e., organization of work – depending on the nature of the task. For instance, micro-tasks with small rewards do not require any direct interaction between the crowdsourcer and the worker; creative tasks with higher rewards may instead require a direct negotiation among the two. Each platform has therefore its own way of describing tasks, publishing tasks, selecting workers, assigning tasks, collecting contributions, assessing quality, rewarding work, etc. The tactic, thus, defines how the platform manages one individual crowd assignment, that is, an *atomic* or indivisible piece of work performed by one

worker in one transaction.

Composite tasks that require the coordination of multiple individual tasks (assignments of different types of work) are typically not supported. We call these composite tasks *crowdsourcing processes*, since they require the coordination of multiple individual atomic tasks. Enacting crowdsourcing processes therefore requires the implementation of ad-hoc logic [5]. This logic may comprise the specification of a *control flow* that describes in which order tasks are executed, a *data flow* that describes how data is passed among tasks, additional *machine tasks* that are executed by machines instead of by the crowd (e.g., to aggregate data produced by the crowd), and similar. These requirements make the crowdsourcing of composite tasks highly process-driven [6], [7], [8], [9] – hence, the name "crowdsourcing process." Platforms like Turkit [10] or Jabberwocky [11] do provide some support for structured crowdsourcing processes to be deployed on top of existing crowdcourcing platforms, typically Mechnical Turk. Yet, they suffer from the inherent limitations described above and are only *hardly integrable* with legacy systems, e.g., to enable the inclusion of crowdsourced tasks into common business processes or applications.

These limitations put the healthy growth and further spreading of crowdsourcing at *risk*, in that they load the crowdsourcer with unnecessary coordination overhead, prevent scalability, lower flexibility, and threaten cost and time savings if more complex, structured work is to be crowdsourced.

In order to overcome these limitations, we propose the following *contributions*:

- A flexible crowdsourcing platform called Crowd Computer (CC) that allows one to program custom crowdsourcing logics for individual as well as structured tasks (an extension of [9]);
- A BPMN-based modeling language equipped with a dedicated visual editor that together allow one to program the CC;
- A publicly available, open-source implementation of the CC on top of standard BPM technology that can easily be integrated into existing software and processes and a respective, automated model compiler; and
- A concrete case study that demonstrates the applicability and effectiveness of the approach in the context of crowd-based mining of mashup model patterns.

Next, we introduce crowdsourcing and our problem statement, then we derive requirements and explain how we approach the problem (Section 3). In Section 4 we present the Crowd Computer, while Sections 5 and 6 develop the modeling language for crowdsourcing tactics and processes. In Section 7 we describe the implementation of our prototype, which we evaluate in Section 8 with the use case. We discuss limitations and related works in Sections 9 and 10 and then anticipate some future work.

Figure 1. The high-level steps of crowdsourcing and the respective actors

2. Crowdsourcing: Concepts and State of the Art

Crowdsourcing is a young, yet already complex practice, especially as for what regards the different ways work can be outsourced and harvested. In the following, we conceptualize the necessary background and define the problem we approach in this article.

2.1. Crowdsourcing Tasks

Crowdsourcing a task using a platform typically involves the steps illustrated in Figure 1 (not all steps are mandatory): The crowdsourcer *publishes* a description of the task (the work) to be performed, which the crowd can inspect and possibly express interest for. In this step, the crowdsourcer typically also defines the reward workers will get for performing the task and how many answers he would like to collect from the crowd. Not everybody of the crowd may, however, be eligible to perform a given task, either because the task requires specific capabilities (e.g., language skills) or because the workers should satisfy given properties (e.g., only female workers). Deciding which workers are allowed to perform a task is commonly called *pre-selection*, and it may be done either by the crowdsourcer manually or by the platform automatically (e.g., via questionnaires). Once workers are enabled to perform a task, the platform creates as many *task instances* as necessary to process all available input data items and/or to collect the expected number of answers. Upon completion of a task instance (or a set thereof), the crowdsourcer may inspect the collected answers and *validate* the respective correctness or quality. The crowdsourcer typically *rewards* only work that passes the possible check and is of sufficient quality.

2.2. Crowdsourcing Tactics

Depending on the acceptance criteria by both the crowdsourcer and the worker to enter a mutual business relationship (after all, this is what crowdsourcing is about), different *negotiation models* may be adopted to crowdsource a piece of work. For simple tasks (e.g., tagging a photo), it is usually not worth to start a dedicated negotiation process; more complex tasks (e.g., designing a logo or developing a piece of software), instead, may justify a process in which crowdsourcer and worker commonly agree on either the quality of the delivered work or its reward. Since it is the crowdsourcer who starts the crowdsourcing process and approaches the crowd, we call these negotiation models *crowdsourcing tactics*. Three major tactics have emerged so far (Figure 2 illustrates the relationship workers–tasks–task instances):

(a) *Marketplace*: The marketplace tactic [12] targets so-called micro-tasks of limited complexity, such as tagging a picture or translating a piece of text, for which the

crowdsourcer typically (but not mandatorily) requires a large number of answers. Usually, the acceptance criteria by the crowdsourcer for this kind of tasks are simple and clear, e.g., all answers are accepted (e.g., subjective evaluations or opinions) or only answers that pass a given correctness check (e.g., a given answer is similar to answers of other workers; a worker answered correct to randomly injected tasks with known answers). Rewards for micro-tasks commonly range from nothing (workers perform tasks for fun or glory), to few cents or dollars, without any margin for negotiation. If workers find the offer fair, they perform the task, otherwise they skip it. Prominent examples of crowdsourcing platforms that implement the marketplace tactic are Amazon Mechanical Turk (https://www.mturk.com), Microworkers (http://microworkers.com), and CrowdFlower (http://crowdflower.com).

(b) *Contest*: The contest tactic [13] is particularly suitable to creative tasks for which the crowdsourcer knows the budget he is willing to spend, while he does not have clear criteria to decide which work to accept. Designing a logo or the layout of a web page are examples of tasks that fall into this category. In order to enable the crowdsourcer to clarify his criteria, this tactic invites workers to conceive a solution to a task and to participate with it in a contest. Once a given number of contributions or a deadline is reached, the crowdsourcer can inspect all contributions and choose the solution he likes most, thereby electing the winner of the contest (there could be multiple winners). Only the winner gets rewarded. Examples of crowdsourcing platforms that implement the contest tactic are 99designs (http://99designs.com), InnoCentive (http://www.innocentive.com), and IdeaScale (http://ideascale.com).

(c) *Auction*: The auction tactic [14] targets tasks for which the crowdsourcer has relatively clear acceptance criteria, but for which he is not able to estimate a reward. Coding a piece of software is an example of this kind of task. An auction allows the crowdsourcer to publish his requirements and to let workers to express the reward for which they are willing to perform the task. Typically, but not mandatorily (this depends on the adopted auction model), the worker with the lowest offer gets assigned the task and is payed accordingly upon delivery of the agreed on work. An auction can thus be seen as a combination of a contest (to win the auction) and a marketplace task with pre-defined worker assignment (to perform the task). An example of auction-based crowdsourcing platform is Freelancer (http://www.freelancer.com), which allows programmers to bid for the implementation of software projects.

The latter two tactics aim at producing one result that satisfies the crowdsourcer's need. The marketplace tactic, instead, most of the times aims at producing a large number of results that jointly satisfy the crowdsourcer's need. For instance, the quality of a translation is higher the more workers contribute to it.

n this article, we do not focus on the analysis of which tactic is most effective for which task type (see for instance [15]). We rather aim to enable the flexible design of tactics, which in turn enables answering more efficiently these and other research questions. We specifically do so by focusing on how to aggregate results and coordinate workers, which is typically out of the scope of crowdsourcing state-of-the-art platforms. We think that support for different tactics (the three above plus custom ones) and for aggregating results and coordinating workers is crucial for the crowdsourcing of work that is complex, as described next.

2.3. Crowdsourcing Processes

Figure 3 shows an example of how to iteratively crowdsource the recognition of a line of text using micro-tasks until the last worker has no doubts left. As illustrated by the model, translating the text may be more complex than a single task: it may involve multiple instances of the text recognition task and, possibly, different workers. It furthermore requires passing data from one task instance to another, so as to incrementally improve the translation. We call such kind of structuring of multiple crowd tasks and task instances, in order to achieve a common goal, a *crowdsourcing process*.

Crowdsourcing processes are not supported by the crowdsourcing platforms referenced above. The emergence of programming frameworks and higher-level platforms built *on top* of these crowdsourcing platforms (most notably, Mechanical Turk) and extending them with basic process management features, however, evince the need for the automation of these kinds of crowdsourcing processes [10], [11], [17], [18], [5], [8].

The state-of-the-art frameworks/platforms aim to facilitate the crowdsourcing of individual tasks that require splitting and the coordination of split tasks (e.g., to collect separate feedback for a large number of photos), typically inside an own, proprietary environment. There is however a set of areas that may benefit from processes that are more advanced than these, such as:

- *Product design*: Early feedback to new products is crucial to success. Integrating crowd tasks for the collection of feedback, acceptance studies or testing into production processes may represent a significant competitive advantage.
- *Social marketing*: Marketing campaigns are increasingly conducted online. The integration of crowdsourcing into common marketing processes may allow organizations to boost and monitor their social presence.

Figure 4. A crowdsourcing process involving different actors (humans, machines and the crowd) and possibly different crowdsourcing tactics

- *Idea management*: Increasingly, organizations engage the crowd in the ideation of new products or services. Common social networks do not provide adequate support for this, and idea management systems may be too rigid. Custom crowdsourcing processes may make the difference.
- *e-Democracy*: Crowdsourcing may enable the participation of the civil society to politics. How to involve society (e.g., via voting, promoting petitions or similar) is as crucial as election laws are. Each party may have its own preferences and goals, i.e., crowdsourcing processes.
- *Human computation*: Despite the increasing computing power of machines, there are still tasks that only humans can solve, e.g., telling whether a portrait photo is beautiful or not. Advanced crowdsourcing processes enable the flexible integration of both humans and machines, unleashing the computing power of both.

What is missing to bring CS to these areas is (i) support for crowdsourcing processes that are integrated with common business process management practices [19] able to seamlessly bring together the crowd, individual actors and the machine, and (ii) support for crowd tasks with different crowdsourcing tactics inside a same process, depending on the specific needs of the crowdsourcer. The three tactics described above are just the most prominent ones emerged so far. They are typically hard-coded inside their crowdsourcing platforms, and each platform has its own tactic with proprietary pre-selection, quality assessment and rewarding logics that require a significant amount of manual labor by the crowdsourcer. It is not possible to freely choose and fine-tune how to negotiate a task with the crowd.

The most pragmatic approach to the problem so far is the one by CrowdFlower, which has a set of dedicated CrowdFlower employees working collaboratively with its biggest customers to process data for their most complex jobs/processes. The recent Crowd-Flower Labs initiative (http://www.crowdflower.com/blog/introducing-crowdflower-labs) eventually aims to bring some level of automation to key customers.

The process in Figure 4 illustrates an example of a crowdsourcing process. It models a logo evaluation procedure where the crowd judges a logo that the crowdsourcer (e.g., a design studio) wants to be assessed. The first two tasks model the upload and evaluation of the logo, and are respectively a *human task* and a *crowd task*. The third task in the process is a *script/machine task* to compute the average of the votes which are given by the crowd. The process is closed by a *human task* where the crowdsourcer inspects the results of the assessment, the list of the votes, and their average. Human tasks are "typical 'workflow' tasks where a human performer performs the task with the assistance of a software application" [16]. Crowd tasks are performed voluntarily by workers of the crowd. The crowd task in the model internally adopts a custom tactic defined by the crowdsourcer that resembles a common marketplace tactic, e.g., without validating results but simply paying all participating workers. The script/machine task is a standard BPMN component that contains a script written by the modeler and later executed by the BPMN engine.

2.4. Problem Statement

The problem we approach in this paper is the design of a model-driven development and execution environment for the implementation of (i) advanced crowdsourcing processes and (ii) custom crowdsourcing tactics. The work does not focus on any specific application domain (for instance, Hobfeld2014 provide a good overview of crowdsourcing practices in the domain of multimedia quality evaluation); the work rather aims to provide an enabling technology for the development of flexible, highly tailored crowdsourcing instruments for all kinds of domains. Also, we do not further elaborate on how to most effectively describe tasks or on how to fine-tune rewards, so as to maximize crowd participation or quality. These are aspects that very strongly depend on the specific task to be crowdsourced, and good studies of the topic already exist [20], [21], [22].

Next, we summarize the set of core requirements we identify and explain how we tackle the problem.

3. Modeling and enacting advanced crowdsourcing processes

Supporting the modeling and enactment of crowdsourcing processes is a complex task, especially if the aim is to cater for both crowdsourcing processes and crowdsourcing tactics in an integrated fashion and with a similar level of abstraction. We assume that the crowdsourcer interested in this kind of CS support has working background in business process modeling and management, basic knowledge of crowdsourcing, and the skills necessary to develop own Web pages for data collection. For simplicity, we use the term "crowdsourcer" to refer to both individuals and groups of collaborating individuals with these skills and that want to crowdsource work.

Experience has shown that crowdsourcing any task that is not as trivial as filling a questionnaire or asking workers to like a Facebook page requires the implementation of custom task logics and designs, typically in the form of Web pages to which workers can be forwarded from within crowdsourcing platforms. We therefore assume that CS tasks are implemented by the crowdsourcer and that the CS platform (the Crowd Computer) focuses its attention on the coordination of work.

In line with this choice, we further assume that the Crowd Computer does not manage data, as such are directly managed by the CS task pages. The CC concentrates on the management of metadata, such as references to the actual data and process runtime data. This allows us to keep the CC

lightweight and the crowdsourcer not to loose control of the data. The alternative would be to also host the actual data on the CC. This would however require the availability of considerable amounts of data storage (expensive) and may imply slow task deployments. More importantly, if the CC does not store any potentially sensible data, e.g., personal or protected data, the crowdsourcer has a higher trust in the platform, and we do not have to worry about the respective legal aspects.

3.1. Requirements

Given these assumptions and a thorough analysis of own CS processes, we derive the following requirements for the implementation of advanced CS processes:

R1 *Crowd tasks.* The crowdsourcer must be able to properly describe tasks and to link them to external task pages.

R2 *Metadata exchange.* CS task pages must be enabled to exchange metadata with the CC, so as to allow the CC to coordinate tasks and to propagate data.

R3 *Crowdsourcing tactics.* The tactics used to crowdsource a given task may differ from task to task. The crowdsourcer must therefore be able to design custom tactics, including custom quality assessment and rewarding logics.

R4 *Human tasks.* These are used when a task has to be executed by a designated human actor (not the crowd), such as the crowdsourcer or an external expert. Supporting human tasks is thus necessary to allow arbitrary human actors to participate in a CS process, e.g., to validate task outputs.

R5 *Machine tasks.* Similarly, it is necessary to support machine tasks that enable the integration of computations performed by a machine, e.g., an operation to compute the average of a series of data extracted from crowd-provided data.

R6 *Control flow.* Processes are composed of a set of tasks that need to be coordinated. It is necessary to be able to specify the order in which tasks are executed and possible decision points that allow one to split and merge the execution flow.

R7 *Data flow.* Tasks may consume data as input and produce data as output. It is thus necessary to enable the crowdsourcer to define clearly which data are produced and consumed by which task and to specify suitable data propagation logics.

R8 *Data management.* CS typically produces large amounts of data. Propagating data among tasks with different data constraints (e.g., show only 3 photos out of a given set of photos) requires being able to suitably cut, slice, merge and format data.

Concretely, enabling a crowdsourcer to crowdsource advanced CS processes therefore asks for the design and implementation of:

R9 *Modeling language.* The model-driven design of CS processes with the above features requires the conception of a formalism that allows the crowdsourcer to model own CS processes (BPMN4Crowd).

Figure 5. High-level architecture for the design and enactment of flexible crowdsourcing processes.

R10 *Modeling editor.* In order to turn the modeling language into an instrument that can also be used in practice, it is necessary to equip the language with a suitable graphical editor that allows the crowdsourcer to model CS processes.

R11 *Runtime environment.* The execution of CS processes then requires the implementation of a runtime environment that must be able to cater for the above features. We call this environment Crowd Computer (CC).

R12 *Deployment.* Turning a high-level process model into a running process requires the implementation of a dedicated model compiler and support for the automatic deployment of the generated artifacts.

3.2. Approach

Figure 5 illustrates the architecture we propose to approach the above requirements. It is divided into three parts: design, deployment and execution.

The *design* of CS processes starts from the design of CS tasks. These can be implemented by the crowdsourcer as regular Web pages using his/her preferred *Web development environment* (R1). For the exchange of metadata between task pages and the CC, a dedicated JavaScript *task manager API* implements the necessary interface (R2). For the modeling of CS processes and tactics, we specifically extend BPMN (Business Process and Modeling notation) [16]. The rationale of this choice is that BPMN already satisfies some of the requirements we outlined above: it supports human tasks (R4) and machine tasks (R5), control flow (R6) and data flow (R7) constructs, and its native extensibility allows us to implement custom task types for CS tasks (R1) and data management operations (R8), which are not supported by the language. This extended version of BPMN represents BPMN4Crowd (R9), which we accompany with a dedicated *BPMN4Crowd editor* for the visual design of CS processes and tactics (R10).

BPMN4Crowd is specifically tailored to the needs of crowdsourcing (the crowd, CS tasks, quality assessment, rewarding, data transformations) and aims to abstract them with intuitive constructs. In [9] we proposed a structured approach for the modeling of CS processes, which we refine in this article. Specifically, BPMN4Crowd distinguishes three conceptual modeling layers that foster the separation of concerns:

- *CS processes.* These are the highest level of abstraction and represent the main processes to be automated. The crowdsourcer models the control flow (R6), data flow (R7), CS tasks (R1), human tasks (R4), machine tasks (R5), and data transformation tasks (R8) that compose the CS process and configures the tactics of CS tasks (R3) and the exchange of metadata (R2).
- *CS tactics.* The second level of abstraction focuses on the tactics (R3). The crowdsourcer decides how to

approach the crowd and how to manage crowd tasks. Tactic models express interactions with the APIs of the Crowd Computer (R11).

- *Validation and rewarding logics.* The third and lowest level of abstraction hosts the models of the different quality assessment and rewarding processes that can be re-used when designing tactics (R3).

In this article, we specifically aim to ease the design of these kinds of processes. Tactics and validation/rewarding logics (R3) are particular processes that are not easy to model. We allow the crowdsourcer to develop own tactics and logics, but we also provide a set of pre-defined tactics and logics that can easily be reused via suitable configurations of the CS processes. Low-level models serve as libraries to higher levels. We opted for a consistent approach to express logic throughout the whole platform (models) and to allow crowdsourcers to implement their very own validation logics. This acknowledges that we would not be able to pre-define all possible validation logics and, at the same time, enhances the reusability of BPMN4Crowd processes.

For the *execution* of crowdsourcing processes we provide the Crowd Computer (R11), which internally uses a standard *BPMN engine* for process execution plus a set of *Crowd Computer APIs* for the management of the crowdsourcing aspects not supported by the engine, e.g., metadata management, crowd management, quality assessment. The CC comes with a *front-end* that can be used by the crowd to discover and perform tasks. The crowdsourcer can start and stop tasks, assign them, approve or reject results, reward workers, and is able to keep track of work that is assigned, done or pending, and of the performances of each worker. The *compiler and deployer* automatically translate the models into instructions for the CC (R12).

4. The crowd computer

Crowdsourcing work requires a crowdsourcing platform to advertise, assign, perform and reward work, i.e., CS tasks. With existing platforms, it is not possible to design own tactics for tasks, and crowdsourcers have to live with predefined choices (e.g., the type of payment method) and limited customization capabilities, and they are required to use different platforms if they want to crowdsource work using different tactics (e.g., inside a CS process). The *Crowd Computer* (CC) aims to overcome these limitations by (i) singling out as APIs all the basic features needed to crowdsource work (task, crowd, quality and reward management), (ii) not hardcoding any own CS tactic that would anyway limit somehow someone, and (iii) enabling the crowdsourcer to assemble own tactics coordinating API calls and CS processes on top.

Figure 6 illustrates the internals of the CC. The approach to enable the flexible configuration of custom CS tactics is to use a *BPMN engine* that can be programmed with process models. These latter are stored in a dedicated *process repository* and may contain tactics, validation and CS process definitions. The tactics rule the low-level interaction with the internal APIs, while the validation and CS processes define possible complex CS logics on top. The APIs maintain internal *metadata* about tactics and processes in execution. The *CC front-end* acts as interface toward the crowd and the crowdsourcer. It embeds the *crowd task pages* used to collect results from workers and has a management UI for the crowdsourcer.

Figure 6. Functional architecture of the Crowd Computer.

Running a CS process in the CC therefore requires the availability in the process repository of one process model for the CS process, one process model for each tactic used in the CS process, one process model for each task validation logic used in the tactics. All models are instantiated hierarchically by the BPMN engine, starting from the CS process model, which is started via the CC front-end or via the BPMN engine. The task, crowd, quality and reward manager APIs are called from within the CS process by suitable machine tasks. The four APIs all work on an integrated metadata repository, providing for the synchronization of the respective crowdsourcing operations. Each crowd task inside the CS process causes the BPMN engine to instantiate a new process implementing the respective tactic, which, in turn, may instantiate a process for the validation of task outcomes. The instantiation of a tactic process causes the CC front-end to publish the respective task. This allows workers to pick tasks and to perform them through the embedding of the external task pages. Submitting work through a task pages notifies the task manager API and BPMN engine of the completion and progresses the state of the engine and the metadata repository. A CS process terminates when all internally instantiated sub-processes have completed and now further tasks are to be processed.

We discuss the internal metadata model and APIs next; then we explain the structure and logic of crowd task pages for the CC. The following sections explain how to implement tactics and CS processes.

4.1. Metadata Model

To support the integration of APIs and the correct execution of tactics and processes, the CC implements the metadata model in Figure 7. The model distinguishes the description of tasks, tactics and processes (instruction register) from their execution (status register). The purpose of the model is to focus on metadata and not on data, that is, the CC does not necessarily manage actual data as produced by the workers during task execution. It only keeps track of respective metadata, such as the keys that identify answers by workers inside the crowd task data maintained by the crowdsourcer.

The core element of the *instruction register* is the *TaskDefinition*, which contains all the information necessary to describe a crowd task, e.g., name, description, type (human/machine/crowd). Each task is part of a *Process* owned by a *User*. Users have a profile composed of *UserProfileParameters* (used for pre-selection) and are connected

Figure 7. Metadata model underlying the execution of tasks, tactics and processes in the Crowd Computer.

to task definitions through a *UserRole*, which tells which role a user plays in which process (e.g., for human tasks). Task definitions comprehend the *ParameterTypes* of their configuration and runtime parameters (e.g., for the invocation of machine tasks or for control flow decisions) and the *DataObjectTypes* and *DataPropertyTypes* of the inputs and outputs consumed/produced by the task execution. Data objects store references to the external crowd task data repository and enable data splitting and merging inside the CC. Data properties enable the association of descriptive information to data objects and enable, e.g., the definition of custom quality controls and the correlation of task instances if multiple instances work on a same object.

The *status register* contains the runtime data. The *Task* represents a task instance, it contains information about the task status, and is associated to a user (e.g., a worker). It is also associated with the set of *Parameters* and *DataObjects* carrying concrete input and output data, as well as with the *DataProperties* describing the produced data objects. The *RewardPayment* stores information of the reward that is given to workers, which can be used with different *payment_services*.

4.2. Programmable APIs

The metadata modeled in Figure 7 is created and consumed by the processes running in the BPMN engine (tactics, validation and CS processes), mediated by the internal APIs of the CC. That is, the CC can be "programmed" via process models coordinating the invocation of API operations representing the "instruction set" of the CC. The goal of the CC is to keep the instruction set focused to the core crowdsourcing aspects and, hence, simple, manageable and efficient. The four APIs are (see Figure 6):

- *Task manager:* It comprises a set of operations for task life cycle management and data propagation among tasks. Operations are instantiating tasks, assigning tasks to workers, canceling tasks in execution, re-running them, enacting machine tasks, etc. Each task can be instantiated multiple times, so as to collect the amount of data the task is required to produce as output. The API parses the task definitions with their configuration parameters, runtime parameters, and input and output data object specifications, and manages the respective runtime values.
- *Crowd manager:* It is in charge of human resource management and worker pre-selection. It comprises a set of operations for the management of users, such as resolving user roles, pre-selecting potential workers, tracking which user executed which task, sending direct invitations to people, ensuring separation of duties, etc.
- *Quality manager:* Typically, but not mandatorily, only work that passes a quality assessment is rewarded. The

Excerpt of most important operations of the CC APIs.

API	Operation	Input	Output	Description
Task	create	Description, task page URL, number of instances, deadline, validation strategy, reward, reward strategy, preselection condition	task id	Creates a task in the C
	start	task id	-	Starts a task and make
	stop	task id	-	Stops a task and make Pending feedback for cepted.
	createInstance	task id	task instance id	Creates an instance of it available to workers
	stopInstance	task instance id	-	Stops the specified ins
	assignInstance	task instance id, user id	-	Assigns a task instanc
	storeResult	task instance id, data	-	Stores the data for the
	updateInstance	task instance id, data	-	Updates the status of
	updateInstances	task instance ids (array), data	-	Updates the status of
Crowd	preselect	user id, task id	true/false	Runs the pre-selection check if he/she is all task.
Quality	set	task instance id, data	-	Assigns the quality le ment to the selected i
Reward	give	user id, reward	-	Assigns a reward to a

Note:Each API implements additional operations that are not shown here. In this table, we concentrate on those operations that are used to model the processes we present later in the paper.

Figure 8. Implementation logic of task pages to be included into and managed by the Crowd Computer.

Quality manager provides for the necessary management of quality metadata for crowd tasks. It allows one to (i) represent quality as a normalized value in a range from 0 (min) to 10 (max), (ii) to set a minimum threshold level below which work is rejected and not rewarded, and (iii) to assign quality assessments to task instances at runtime. How to set threshold levels and assess quality, i.e., the semantics of quality, is decided by the crowdsourcer at design time and assessed via custom human/machine or CS tasks at runtime, respectively.

- *Reward manager:* It provides for payment management and is in charge of keeping track of which task instances have been rewarded, i.e., paid, and how. Each task may have an associated reward and payment service, and payments may occur for individual task instances, bundles of task instances or similar. The CC does not impose any reward logic; these can be specified ad hoc or instantiated from reusable model templates. The CC does also not impose any concrete payment platform (e.g., PayPal, VISA); such can be chosen and plugged in by the crowdsourcer. Again, the semantics of the *Reward_amount* property of tasks is decided by the crowdsourcer.

In Table 4.2, we summarize the most important operations accessible via these APIs that we will use throughout this paper. For presentation purpose, we are not able to discuss all operations here, so we refer the reader to http://apidocs.crowdcomputer.org for the complete details.

4.3. Crowd Task Pages

The front-end of the CC is implemented as a regular Web application. In order for the front-end to be able to visualize a task page, which is also a Web application, it must be published online and embeddable into an iframe inside the front-end's own UI.

In the most simple case a task page is a static HTML Web form hosted online. For more complex task pages with own application logic, the application logic should be integrated with that of the CC to enable the exchange of data and/or metadata, as described in Figure 8. To support this integration, the task pages must contain a *task manager client* (a JavaScript library in the current implementation). This client allows the task page to communicate with the *task manager API* of the embedding CC front-end. Specifically: When a task is instantiated and its UI is embedded into the iframe, the *task manager API* loads from the *metadata store* the *data identifiers* of the input data objects to be processed by the task page (if inputs are to be processed) and sends them to the *task manager client*. The task page receives the identifiers, loads the corresponding data from its internal *data store*, and displays them to the worker. Upon the completion of a worker's feedback, the task page stores the results in its *data store* and sends the respective *result identifiers* back to *task manager API*, which, in turn, stores them into the *metadata store* and marks the instance as executed.

5. Modeling crowdsourcing tactics

In the introduction of this article, we stated that a crowd task is atomic or indivisible, referring to the perspective of the worker who executes the crowdsourced work. Yet, it is important to note that enacting a crowd task in practice is not an atomic action executed only by a single actor. It rather is a composition of different tasks possibly executed by multiple actors, namely the worker, the crowdsourcer, and the platform. We call this composition the tactic, and it models how to crowdsource a given task.

Throughout this article, we introduce *BPMN4Crowd*, an extension of BPMN (Business Processes Model and Notation) specifically tailored to the needs of crowdsourcing, and show how to develop custom tactics. First, we however provide a brief summary of the basis of BPMN.

5.1. Background: BPMN

BPMN [16] is a standard for business process modeling that provides a graphical, flowcharts-based notation to specify business process diagrams. Processes are used to model the interactions and business operations that actors have to perform to achieve a given goal. The BPMN notation is composed of four classes of components: (i) flow objects; (ii) connecting objects; (iii) swimlanes; and (iv) artifacts.

Figure 9 illustrates an example of a BPMN process. The model shows a process in which an insurance company opens a new case for a hypothetical user. Inside the *pool* company, there are two *lanes* that represent two actors. The first is the employee who opens the case and validates the requirements. The second lane models the signature system, which stores the signature given by the user. The second pool models the user's actions: he waits for a notification telling him to sign a document, signs it, and sends the signature back to the signature system. When the signature is stored and the requirements validated the process ends.

This small examples helps us to introduce the BPMN constructs that are used in this paper, concretely the following classes: *Flow objects* contain the main elements to create a process: events, activities and gateways. *Events* affect the control flow, they have a cause and an impact and are used to *start* or *end* a process and to intercept or trigger *intermediate* events; their shape is a circle. *Activities* are used to model the work to be performed, i.e., the *tasks*. There are various types of tasks, among others *human* tasks, executed by a person, or *machine* tasks, executed e.g. by a web service; the shape of tasks is a rounded rectangle with an icon in the top-left corner that identifies the type of task. *Gateways* represent decision points, such as *xor* or *and* conditions; their shape is a diamond. The group of *connecting objects* contains elements that represent *sequences*, *messages* and *data flows* by means of arrows. The sequence (control) flow is modeled with a solid line; the message flow, which models a message exchanged among process participants, is a dashed arrow; the association, used to associate data and artifacts with a task (thus the data flow), is a dotted arrow. *Swim lanes* (short *lanes*) separate the activities of different process participants (also called actors). *Pools* group lanes/participants into organizations, e.g., a company or institution. *Artifacts* are additional elements of the diagram that are used to add context (e.g., a descriptive comment). The most important element of this category is the *data object*, which shows how and which data is consumed or produced by the tasks of the process.

5.2. Basic Structure

Modeling a CS tactic requires complying with a set of conventions that guarantee that the CC is provided with all the information it requires to correctly publish, assign, collect, assess, and reward work. We summarize these conventions in the *basic tactic structure* (a template) depicted in Figure 10. Note that we mark CC API invocations with a CC logo in the top left corner; technically, these tasks correspond to common machine tasks, but for a better readability of the models, we explicitly distinguish those

tasks that interact with the CC-internal APIs from those that interact with humans, external services, or the crowd.

The process diagram is articulated into four pools, an input data artifact, and a minimum set of tasks that invoke the operations of the CC APIs. The pools correspond to the BPMN engine orchestrating the tactic, the CC front-end mediating between the tactic and the crowd, the task UIs (crowd task pages) for performing work, and the validation logics. The task input data carries the input data for the task to be crowdsourced. The five tasks serve the following purpose:

- *Pick task* (no API call need, optional) receives the identifier of the worker that decides to work on a given task listed in the Crowd Computer front-end. The construct is a shorthand for the specification and configuration of a receive message event. Skilled modelers can directly use any of receive message events of BPMN (e.g., message start event or message receive event).
- *Create instance* (`Task.createInstance` and `Task.assignInstance`, mandatory) uses the task metadata from the CC's internal data store to create an instance of the task and to assign it either to the worker identified via a prior Pick task or by automatically assigning a worker (e.g, via email).
- *Receive result* (`Task.storeResult`, mandatory) represents the actual execution of the task instance. It interacts with the Task UI and manages the data identifiers of the task inputs and outputs (metadata) as explained above.
- *Validate result* (`Quality.set`, optional) assigns a quality level to a data object produced by the crowd. The task internally invokes an external validation process (custom or chosen from a library) that invokes the `Quality.set` API.
- *Reward result* (`Reward.give`, optional) assigns rewards to workers. Configurable rewarding logics (functions of the quality assessment values) decide for each worker whether to give or reject the reward and possibly hand the reward out.

It is important to note that the basic tactic structure is *not executable* as is. It rather lists the minimum (and typical) ingredients that are necessary to design a correct tactic. It is clear that this basic structure needs to be extended and turned into a correct, executable process to be applicable in practice.

5.3. Tactics Models

In the following, we show what it means to develop an executable tactic. We model examples of the most important tactics, namely marketplace, contest and bid [12], [13], [14] described in Section 2.2. In addition, we also model a mailing list tactic, which can be useful in all those crowdsourcing scenarios that require targeting a precise group of people, for example, for user studies.

This set of tactic models is not meant to be complete; it rather is a set of examples of how own, custom tactics

can be implemented by the crowdsourcer. All tactics can be reused as they are or extended and adapted to new needs. We recall that a crowdsourcing process may contain multiple crowd tasks, while each crowd task may have its own tactic. This means that within one crowdsourcing process multiple crowd tasks with different tactics may coexist, yet each crowd task has exactly one tactic.

5.3.1. Marketplace. The marketplace [12] is one of the most common tactics for crowdsourcing work. Figure 11 shows a process that represents a possible implementation. The model expresses the perspective of the BPMN engine inside the CC. The process starts with the CC loading the task definition stored in the metadata store. This operation interrogates the `Task` API. The task definition is used to create task instances, which are managed by the *multi-instance subprocess* in the figure. In the subprocess, the first operation creates the task instance, then the subprocess waits for a worker to accept the task. Upon selection by a worker, the CC tries to assign the instance by first performing the possible pre-selection test (part of the Assign instance task): if the worker meets the requirements, the instance is started and assigned to the worker, otherwise the instance is released and the process goes back waiting for another worker. If the instance is assigned, the CC waits until the worker submits the result, which updates the task instance information through the `Task` API. In the case of a timeout (if a worker does not submit results within a pre-defined time interval), the CC marks the task instance as expired, and the subprocess terminates. If the task instance is instead successfully completed, the process starts the validation task, which is another subprocess. Then, the process rewards workers using the `Reward` API. We describe examples of validation processes and rewarding logics later.

The subprocess that manages the task instances has a timer that is triggered if instantiated tasks are not accepted or completed within a given time window. The timer triggers a task that marks all the remaining instances as expired. Only if all instances are completed or expired the process (and the CS task) ends.

This is only one possible implementation of the marketplace tactic. Many others exist. For example, we can define a *minimal marketplace* as a marketplace with a single instance and without validation and reward. An implementation of this tactic needs to have a task to retrieve the task definition, a task to create an instance (without the sub-process), a task to assign the instance, and a task to receive the results. This corresponds to the minimal set of mandatory tasks of our basic tactic structure.

5.3.2. Contest. The contest tactic [13] makes workers compete with their work for a reward. In Figure 12, we model the process of a typical contest tactic. The process is divided

Figure 12. Process diagram of a possible contest tactic. The main process manages the overall flow, the subprocess manages the instantiation of tasks and and the submission of results.

Figure 13. Process diagram of a sealed first-price auction tactic.

into two parts: the main process that is in charge of starting the task, validating results, and rewarding workers and a sub-process that manages the instantiation and execution of task instances. The subprocess is non-interrupting, which means that workers can submit results in parallel until the expiration of a deadline. Once results are collected, the process starts a validation subprocess to decide the winner(s) and then rewards the winner(s). Differently from the marketplace tactic, where the validation and reward logic are instantiated for each task instance, in a contest they are executed only once for all the instances at the same time.

The Collect results subprocess models the collection of workers' answers. It sets a timer that, after a predefined amount of time (defined by the crowdsourcer), ends the sub-process and closes the contest; this implements the deadline of the contest. Answers from workers are collected only as long as the contest is open. This behavior is modeled using an *event subprocess* (the dashed box) with a non-interrupting start message (the dashed start message) that can be triggered many times in parallel by different actors. The subprocess is thus executed every time an acceptance message arrives from a worker.

Also for the contest, various implementations exist. For instance, we can again imagine a *minimal contest* tactic, in which we remove the rewarding task. The other three tasks cannot be removed from the main process, as they are all mandatory. The logic of the Collect results subprocess is rather standard, and one cannot modify much. What a crowdsourcer may want to change is the logic used to decide the winner. In a minimal contest, this logic could be picking randomly a worker or selecting the first worker who submitted something. In more complex and real scenarios, this logic may require the crowdsourcer or an external expert to select the winner, or it may ask the crowd to rank the results, or it may use a machine task to compute the winner.

5.3.3. Auction. The auction tactic [14] is fundamentally different from the previous two. While in the marketplace and contest the reward is specified at the beginning, in the auction the reward is the result of a process to be run. In crowdsourcing, auctions are usually executed in a reverse fashion, in which the winner is the worker who offers to perform a task at less money (the bid) then the others. The (reverse) *English* auction model allows workers to bid against each others for a fixed amount of time; then the lowest bid wins. In the (reverse) *Dutch* auction, the crowdsourcer specifies an initial reward and goes on raising it by a small amount until the first worker accepts the contract; this worker wins the auction. This type of auctions requires a significant amount of time to participate in the

Figure 14. Process diagram of the mailing list tactic with two subprocesses.

bid, time that is not paid and that may discourage workers from participating. Another common model is the (reverse) *sealed first-price* auction, in which the requester specifies the maximum amount of reward he is willing to pay, and the first worker that bids less than this value wins. This tactic is, for example, often used by platforms with professional workers, such as programmers or freelancers.

In Figure 13 we provide an example of process diagram for a sealed first-price auction. The process shares common parts with the contest and the marketplace: From the contest it borrows the possibility to have multiple workers competing with their bids (Check bids). The respective subprocess is therefore implemented similarly to the one of the contest tactic, with the difference that now it is a signal (the triangle) that stops the subprocess execution. The signal is emitted as soon as a worker bids less than the threshold and the system creates an instance of the task and assigns it to the winning worker. The main process then proceeds similarly to the marketplace tactic, yet managing only a single task instance (the one of the winner of the auction). The process ends with the usual validation of the work and the payment of the agreed-on reward based on the quality assessment.

We already explained that many other possible auction processes can be conceived. As for the *minimal auction* tactic, what can be eliminated is the Review result task, assuming the worker performs satisfactorily. The reward task is mandatory, since the tactic is based on a negotiation of the reward. The bid logic (the subprocess) is most open to variations; what we modeled here is one of the simplest logics.

5.3.4. Mailing List. This tactic is not a traditional crowd-sourcing tactic, yet it is a common and very effective way of soliciting people to participate in some effort and to get work done (e.g., questionnaires or scientific studies). The logic of this tactic is the opposite of the other tactics, in that it actively pushes tasks to workers, while in the others it's the workers who pull tasks. Some platforms implement similar approaches, e.g., Mechanical Turk allows the crowdsourcer to assign a task to a specific person.

In Figure 14, we model the process of a possible mailing list tactic. Task instances are created automatically and assigned to selected workers in advance (Assign Tasks subprocess). The assignment keeps track of which user is assigned which instance, information that may be useful for validation and reward. Then the platform waits for workers' feedback, up to a given timeout. Each time a worker submits work, the system stores it, validates it and rewards the worker. Generally all the workers are rewarded, if a reward is offered at all. If a worker is not willing to participate, the system marks the respective instances as expired after the timeout has expired.

As usual, different variations of this tactic may exist. For instance, we could invite people also anonymously, e.g., by

allowing the crowdsourcer to share a link to the task that can be forwarded via email, independently of the CC. For the *minimal mailing list* tactic it is enough to remove the validation and rewarding tasks.

5.3.5. Custom Tactics. The number of possible tactics to crowsource a task, that is, the number of possible negotiation models bringing together workers and crowdsourcers, is unlimited. The above tactics represent the most common ones emerged so far. Other tactics may feature moderator-based approaches, contests with groups of winners, or tactics in which the reward depends on the assessment by a control group or on majority decisions [15]. More advanced tactics may feature a two-stage approach [23], [24] in the assignment of tasks to workers that, for example, use one task to assess the preparation/suitability of workers and another task for the actual task of interest; access to the second task is granted only to workers considered reliable enough in the first task (pseudo-reliable workers).

Sophisticated negotiation models may require complex implementations, raising the doubt whether these are still tactics or already crowdsourcing processes. Technically, it is fully up to the crowdsourcer to decide how much complexity to put into the tactic and how much of it to keep outside, that is, in the crowdsourcing process. For the sake of reusability across different crowdsourcing processes, however, the more of the negotiation logic is inside the tactic, the better. Our recommendation is to aim at tactics that allow one, at the CS process level, to focus exclusively on the integration of crowd tasks into bigger logics, possibly containing other crowd tasks, without having to focus on how to actually crowdsource work (selection of workers, quality control, rewarding).

5.4. Validation and Rewarding Logics

The tactic models described previously neglect the details of the last two aspects of crowdsourcing, that is, validation and rewarding. Given their reusability across tactics, the respective logics can be designed independently of the tactics and associated to them later, e.g., via simple configuration parameters. As highlighted in the basic tactic structure, validation (quality assessment) logic can be as complex as full-fledged CS processes with different actors involved; the rewarding logic is, instead, more of a simple configuration of how to give a reward, once quality has been assessed. For both, the CC implements a set of pre-defined options, which we describe here. Custom validation logics can be implemented similarly by the crowdsourcer.

5.4.1. Validation Processes. Validation processes have the goal of producing a quality assessment for a given data object produced by the crowd. They are invoked by the Validate Result task of the tactics, which also stores the assessment values in the CC's metadata store upon the completion of the validation processes. The most common quality assessment logics are (see Figure 15):

Figure 15. Example process diagrams of four validation processes involving different actors. All processes produce a quality assessment value in output to be stored in the CC's internal metadata store.

(a) *Expert validation:* This validation involves the crowdsourcer himself/herself or another, external expert. The process consists in a simple human task that asks the expert to assess the data object and produces the assessment value as output. If the assessment is positive, the work is accepted, otherwise it is rejected.

(b) *Marking:* This approach asks the crowd to mark (rank) the work of a given worker. The output is based on the average mark given by a set of workers: if such exceeds a given threshold, the work is evaluated positively, otherwise negatively.

(c) *Gold data:* This validation is based on so-called gold questions (whose answers are known in advance) that can be checked automatically by a suitable machine task [25]. Checking the answers to gold questions can be used to estimate quality without analyzing the actual data objects. This validation is straightforward and does not require additional time or human activities. The percentage of correct gold answers is compared with the threshold (decided by the crowdsourcer) to decide whether to accept or reject the whole data.

(d) *Agreement:* This validation requires the agreement on the quality of a data object by at least two out of three judges, where judging is again crowdsourced (CS tasks). If two workers immediately agree, the agreed-on value (true or false) is chosen as output of the process. Otherwise, a third worker is asked to judge, and his/her judgment is used as output, since it for sure agrees with one of the previous two.

Various other logics can be conceived and used to assess quality. It suffices to model a process like the ones in Figure 15 (the next section provides the necessary modeling constructs and conventions), which take a data object as input and produce a respective quality assessment as output (recall Figure 10). This latter is stored as metadata and used to decide who to reward.

In this respect, it is important to understand the relationship between validation logics, tactics and task types. The crowdsourcer chooses validation logics and tactics depending on: (i) the task type, (ii) the desired quality level of results, and (iii) the available resources to spend. The use of gold data can, for instance, guarantee a good level of quality of marketplace tasks, where it is possible to automatically evaluate results against trusted data (e.g., sentiment analysis with only possible answers "positive", "neutral" and "negative"). More quality-critical tasks (e.g., nudity detection in images uploaded on kids websites) may require the use of an agreement logic with redundancy of answers (higher cost). Gold data and agreement can also be applied together, in order to keep only agreed answers from a pool of workers checked against trusted data. Marking works for tasks where it is not easy to evaluate results automatically and instead a subjective evaluation is needed (e.g., voting for proposed

ideas). Expert validation usually takes place in contest and auction tactics for creative tasks (e.g., logo creation). Also marking and expert validation can be applied together in contests, e.g., using markers to create a pool of top-k best results and an expert only to select among the top ones to pick the winner.

5.4.2. Rewarding Logics. Rewards are important in crowdsourcing, since they motivate workers to perform tasks. The most used rewarding logics are:

- *All/none.* This is the most trivial logic. All the workers are paid (or none of them), independently of whether the work is satisfactory or not. This choice is not used often, but it can be executed automatically by a platform if no counter-instruction is given within a predefined time window.
- *Upon validation.* The payment upon validation is perhaps the most used logic. Only the works that pass the validation step are rewarded. This type of strategy works well with tactics where workers are rewarded only if the result is satisfactory.
- *The best.* The reward is given only to the best among all the workers submitting work. This logic is generally adopted in tactics where workers compete for the reward, thus send results for a same task where only one is accepted as correct.

In addition, these reward logics can be complemented with other strategies, such as:

- *Bonus.* The reward is fixed and specified when a task is created. However, occasionally workers may provide astonishing results, and a bonus reward may be given to selected workers to increment their overall reward.
- *Milestones.* Some tasks can require a long-lasting collaboration between the worker and the crowdsourcer, e.g., writing a software application, which can be split into milestones. In this case, also the reward can be split into milestones. Before starting the task, the crowdsourcer and workers agree on the milestones and deadlines, and at each milestone the worker receives the corresponding reward.

With this set of reward logics it is possible to configure most of the possible scenarios that a crowdsourcer may face. All logics make use of the quality metadata of the CC and its configuration of reward payment services.

All these rewarding logics are examples of logics already implemented in existing platform: The *all/none* and *upon validation* logics are mostly implemented in marketplace platforms, such as Mechanical Turk and CrowdFlower. The former can be executed automatically by the platforms, the latter only upon an explicit judgment expressed by the crowdsourcer. The *the best* logic is used in contest-based platforms, such as 99designs. The *milestones* logic is used in auction platforms, such as vWorkers, and the *bonus* logic is a crosscutting logic supported in several different platforms.

Figure 16. Crowdsourcing-specific notation for data objects and data transformations.

6. Modeling Crowd Tasks and Data Transformations

With the CC and the tactics, we now have almost all the ingredients needed for the development of advanced CS processes. What we still need is clarifying how to use CS tasks in practice inside CS processes and how to propagate and transform data.

6.1. Crowd Tasks

The core construct needed for the modeling of CS processes is the *crowd task* (recall Figure 1), which represents the work to be crowdsourced. The construct does not differ from a standard BPMN task only by its crowd icon, but also by the fact that it actually represents a process on its own (the process of the chosen tactic) that must be properly configured before execution. A crowd task definition therefore comes with a set of parameters for crowd-related information:

- *Description:* short instructions that allow workers to understand and choose tasks
- *Task page URL:* the CS task page implementing the user interface of the task
- *Deadline:* the date and time after which the task expires
- *Tactic:* the tactic process to be used as logic for the execution of the task
- *Tactic parameters:* free input of custom parameters defined inside the tactic models (e.g., `NumberOfInstances = 10`)
- *Reward:* the amount of the reward for the task (e.g., 10 dollars)
- *Reward strategy:* the strategy of the Reward Worker task (e.g., the best)
- *Pre-selection condition:* Boolean expression over user profile parameters (e.g., `User.Gender = 'Woman' AND User.Language = 'English'`)

Crowdsourcing and orchestrating work inside a CS process is thus as simple as setting these properties for each crowd task of the process. However, crowdsourcing work via multiple crowd tasks may ask for proper data transformation logics.

6.2. Data Transformations

Crowd tasks usually ask workers either to produce some output that can be inspected (e.g., photos or text) without input from the crowdsourcer or to produce some output (e.g., tags or translations) given a dataset provided as input (e.g., photos or text). It is also common that consumed and produced datasets are large and that crowd tasks work with different input/output dataset sizes. For example, one task

Figure 17. The different uses of the dataset splitting operator of BPMN4Crowd.

Figure 18. The CS process of the photo contest scenario modeled with BPMN4Crowd.

may ask workers to upload *one* photo, while another task may ask them to compare *two* photos or to select the best out of *three* given photos. Input dataset sizes are usually fixed, output dataset sizes may also be variable (e.g., a task may allow workers to upload a generic set of photos). The mismatch that may arise between output and input dataset sizes asks for proper data management operations.

BPMN has native constructs to specify data objects and associations of data objects with tasks, with the association arrows specifying the data flow of the process. Data objects model individual objects (see Figure 9) or collections of objects (if marked with the multi-instance label, that is, three vertical lines). In order to model collections of collections, in Figure 16(a) we define the *set of sets* data object. This allows us to model, for example, 10 sets of 3 photos each to be crowdsourced using 10 instances of a given crowd task. Properly splitting, merging and filtering datasets further requires a set of dedicated operators; Figure 16 shows their constructs.

6.2.1. Splitting Data Sets. The *Split Data* operation allows splitting datasets into subsets. It takes as input a set of data objects and produces as output a set of sets. The operation can be configured to use different splitting logics, as illustrated in Figure 17:

- *SplitWith(n,r?):* this function allows the crowdsourcer to specify the size of the output sets; the number of produced sets depends on the size of the input dataset. The optional parameter r specifies redundancy: if set, consecutive datasets will contain r similar data objects. The operation is, for instance, used when the crowdsourcer wants to assign to each task instance a precise number of input data objects without worrying about how many task instances need to be created.
- *SplitInto(n,r?):* this function splits the input dataset into n subsets of similar size. The optional parameter r specifies redundancy: if set, consecutive datasets will contain r similar data objects. The operation is used when the crowdsourcer wants to control how many task instances will be created (one instance per subset).
- *Combination(n):* this function generates all the possible combinations of size n from a given dataset in input. Combinations are redundant in that they share repeated elements. They provide for maximum redundancy with the minimum number of output datasets.

6.2.2. Merging Data Sets. The opposite of splitting datasets into subsets is merging a set of subsets into an integrated dataset. BPMN4Crowd uses the *Merge Data* operation for this purpose. The merging logic is unique and works at the data object level (it takes all the elements without distinctions), and no further configuration is required. It takes as input a set of sets and produces a set of items. The

operation can be used to recompose the results of different task instances into a unique set of results.

6.2.3. Filtering Data Objects. Finally, the *Filter Data* operation allows the crowdsourcer to filter data objects inside datasets. Filter conditions can be specified over the metadata (e.g., data properties or quality assessments) associated with the data objects in the CC's metadata store. Only data items that match the filter condition pass the filter. For example, a filter can be used to filter out those data objects that have been evaluated positively by the respective quality assessment process.

6.2.4. Modeling Example. We are now ready to refine the initial logo assessment example described in Figure 4. Figure 18 illustrates the BPMN4Crowd model of the process with the necessary data transformation operations in place. We recall that the crowd task (Evaluate logo) asks workers to assess the quality of a logo, which, as a result, produces a collection of marks. We assume that for this purpose we implement a task page that shows the logo together with a form with a set of HTML radio buttons that allow the worker to express his/her mark. Each task instance thus requires the logo as input and produces one mark as output. The logo is supplied by the requester that uploads it in the first task. In order to compute the average, we merge the marks into a unique list (Merge results task) and we use a machine task (executed internally by the process engine, Compute avg votes) for the math. The average value can then be inspected by the crowdsourcer via the process interface (last human task). The tactic of the crowd task does not require the crowdsourcer to validate results, and automatically gives for example USD 0.05 to all workers.

7. Implementation

In order to ease the creation and execution of BPMN4Crowd crowdsourcing processes, we implemented three tools:

- A *BPMN4Crowd editor* to support the graphical design of crowdsourcing processes (http://editor.crowdcomputer.org); the editor is based on the Activiti BPMN modeling tool (http://www.activiti.org).
- A *BPMN4Crowd compiler* to transform process models into executable BPMN processes and configurations for the CC, as well as a *process deployer* able to deploy crowdsourcing processes onto the CC (http://compiler.crowdcomputer.org).
- An *extended BPMN engine* (based on the Activiti engine) that contains the library of CC APIs that enable the communication between the engine and the CC (http://engine.crowdcomputer.org).

Next we present how we implemented the three tools.

Figure 19. The visual BPMN4Crowd editor in action. The figure highlights the BPMN4Crowd palette with new modeling constructs and the properties window for the setting of task parameters

Figure 20. The transformation of the processes before and after the compilation step.

7.1. BPMN4Crowd Editor

Given the intrinsic complexity of process editors, for the implementation of the BPMN4Crowd editor we opted for the extension of the well-known Activiti Designer, a BPMN plugin for Eclipse (http://www.eclipse.org). The tool supports the implementation of custom tasks, while it lacks support for other custom artifacts, such as new data object constructs, and it does not implement all features of the standard BPMN specification, for example it does not support the non-interrupting event sub-processes (useful for the tactics), nor the message end event (useful for the validation processes). For this reason, the current implementation of the BPMN4Crowd editor provides some of the BPMN4Crowd modeling features differently than presented before: Extended data objects (sets and sets of sets) are not specified using data object artifacts connected to tasks, but instead we use task parameters to declare data objects and to specify data flows. Crowdsourcing tactics and quality validation logics can be modeled as independent processes that can be assigned to crowd tasks and configured via suitable task parameters.

With the BPMN4Crowd editor a crowdsourcer can easily create own crowdsourcing processes. Figure 19 shows a screenshot of the editor. In the right part there is the palette that contains the crowd tasks (A), data transformation tasks (B), and CC API tasks to define the tactic (C), and to define the validation process (D). The crowd tasks (A) enable the reuse of predefined tactics (the ones presented in Section 5.3) and of custom tactics; TurkTask also allows posting tasks on Mechanical Turk. We also added a machine task, different from standard BPMN, to ease the integration of external Web services with the CC. The machine task can be used to *get* data that are used in the process (load data) or to *post* data to execute external services. Each data transformation task (B) implements a specific data operation (e.g., merge or split). The current implementation also supports a prototypical implementation of split and merge operations on metadata structures. For the CC API tasks (C and D), we implemented a set of dedicated tasks that wrap specific API calls. In (C) we have all the tasks that can be used to model a custom task. In (D) the modeler has the tasks that are used to call and model a validation process. The Quality set task allows invoking the `Quality.set` CC operation directly from within a validation process.

The central part the BPMN4Crowd editor features the canvas where the process is modeled. The crowdsourcer can drag and drop tasks from the palette and compose his/her own crowdsourcing process. The view in Figure 19 is the process model for the logo assessment example we introduced in Figure 4. For each data transformation or crowd task, the crowdsourcer can specify a set of configuration parameters in the property tab in the lower part of the editor.

The parameters are the ones already specified for crowd tasks (Section 6.1), plus additional fields that are used to specify input and output data objects as well as others that refer to the configuration of the tactic, such as the tactic process and reward parameters:

(a) *Description of the task.* This is the text that is presented to the worker when he/she starts the task. It should describe what the worker has to do to complete the task.

(b) *Task duration.* This parameter specifies for how long the task will be active. After this time the task will automatically be stopped by the system (deadline).

(c) *Page URL.* This is the parameter that specifies the URL of the task page created by the crowdsourcer. The task page implements the interface of the crowd tasks.

(d) *Tactic process.* Since each custom crowd task has its own tactic process, the crowdsourcer can specify here the process name of the tactic process to adopt.

(e) *Tactic parameters.* Here the crowdsourcer can pass configuration data from the CS process to the tactic process.

(f) *Reward.* This parameter specifies the quantity of the reward to be given to workers whose work is evaluated of sufficient quality.

(g) *Reward platform.* The CC has a plugin interface able to support various types of payments. With this parameter the crowdsourcer is able to decide which platform, e.g., PayPal, to use to reward workers

(h) *Input data name.* This parameter is used to specify the name of the data object to read data from as input.

(i) *Output data name.* This parameter specifies the name of the data object to write output data to.

These implementation conventions assure the full expressive power of BPMN4Crowd as introduced in this article.

7.2. BPMN4Crowd Compiler and Deployer

BPMN4Crowd contains both standard BPMN constructs and instructions for the CC APIs and task pages. This makes the language not immediately executable by a standard BPMN engine. The compiler therefore takes as input a crowdsourcing process, rewrites parts of it, and creates a zip file that contains executable BPMN processes (the crowdsourcing process and all its validation processes) that can be deployed on the engine.

The compiler modifies the process in five key aspects, graphically illustrated in Figure 20 showing the differences between compiled and non-compiled processes: Crowd tasks, Pick tasks, Receive tasks, Validation tasks, and the end event in tactic models. The compiler adds a receive message after each crowd task to receive a notification of task completion from the CC front-end upon termination of the respective tactic process. A similar compilation is repeated

for all the validation processes linked by the crowd tasks of the process, adding a receive task after each Validation task. For the internals of the tactic processes, the compiler converts the Pick and Receive tasks into message receive tasks, if these are not yet present (this is, for example, used by the tactics in Figures 12 and 13), and adds a Task Finish task before the end event of tactic models (a system operation that closes the tactic).

Once the compilation is finished, the crowdsourcer can take the zip file and upload it to the deployer that unzips the file, extracts crowd-related information, creates the necessary data structures and metadata to handle the execution of crowd tasks, and deploys the process on the BPMN engine. Then, the process can be executed.

7.3. Extended BPMN Engine

Upon deployment of a process, the start message of the process is sent by the deployer to the BPMN engine to start execution. Standard BPMN constructs are processed according to their standard semantics. To handle the execution of a crowd task on the CC, we extended the engine with additional logic (Java classes). For every crowd task, there is a Java class that sends to the Task Manager API of the CC the task configuration parameters (specified in the process model) and runtime data objects, such as the output of a previous task. The Task Manager receives the data and instantiates the corresponding crowd task by instantiating the respective tactic process. For each new crowd task, the Task Manager creates an own tactic instance and makes the task available to workers. Workers are then able to execute the task instances and to send results. Workers' results are sent via the CC API to the Task Manager that updates the metadata of the corresponding instances. Afterwards, the Task Manager executes the validation process. This process is again implemented as a BPMN process. After the validation, the Task Manager executes the reward procedure. This is a function of the Reward Manager that executes the logic specified by the crowdsourcer and updates the metadata relative to rewarding, giving the reward to the workers. Once the tactic execution is completed, the Task Manager gives back the execution control to the BPMN engine, sending also the task metadata resulting from the execution.

Data tasks are executed in a similar fashion. We created dedicated APIs for each operation (we do not show these in the Figure 6 because it is a mere implementation choice). The Data API implements the operations of the previously discussed functions. Their execution is straightforward: the APIs accept as input data, execute the selected operation, and return the result to the BPMN engine in the form of a message that contains the metadata results.

8. Case Study

In the following, we discuss the benefits and limitations of BPMN4Crowd and the described CC prototype if leveraged on for the implementation of a concrete crowdsourcing process we implemented at the University of Trento in the context of another research project: crowd-based pattern mining.

8.1. Goal and Requirements

Harvesting knowledge from large datasets, i.e., data mining, is a domain where computers generally outperform humans, especially if the dataset to be analyzed is large. If the dataset is instead small and the knowledge to be identified is complex (such as model patterns), humans may outperform machines. Starting from this observation, in [26] and [27] we started investigating whether it is possible to use the crowd to mine patterns from small datasets; we specifically focused on the domain of mashup model patterns to feed our pattern recommender Baya [28], [29] implemented for the data mashup tool Yahoo! Pipes (https://pipes.yahoo.com).

Mashups [30] are composite Web applications developed starting from reusable data, application logic and/or user interfaces sourced from the Web. Mashup tools, such as Yahoo! Pipes, aim to ease the development of mashups, most commonly via model-driven development paradigms that produce models that can be parsed or compiled for execution. Models hence reflect mashup development knowledge, and model patterns (reusable, modular model fragments) may help share and reuse such knowledge. A typical pattern (see for example Figure 22) contains a set of components (e.g., RSS feeds or data processing logics), connectors (propagating data among components), selections, and configurations (e.g., data inputs). The video of how Baya works in practice (https://www.youtube.com/watch?v=AL0i4ONCUmQ) provides a good feeling of the kind of models we target and of their benefit in the development of data mashups (mashups processing data sourced from the Web).

The experiments carried out were designed, implemented and coordinated manually, while as a matter of fact, the methodology followed can be seen as a crowdsourcing process involving human actors (researchers), machine tasks (e.g., to compute metrics), and a set of crowd tasks. In the following we show how the CC and BPMN4Crowd can be used to implement the same experiments as a CS process.

We summarize the *requirements* of the study as follows:

- *Crowd algorithms:* We would like to compare three different "crowd algorithms" for pattern mining, that is, three task implementations asking workers to identify patterns with different levels of complexity and information available:
  - *Naive* shows individual models to workers and asks them to identify pieces of reusable knowledge, that is, patterns;
  - *Random3* presents three different models and asks to identify patterns that recur in at least two of the models;
  - *ChooseN* allows workers themselves to choose N models from a set of 10 given models and asks them to identify patterns inside the N models.

- *Research question 1:* After running the three algorithms, we would like to compare the respective patterns produced by computing a set of simple metrics, such as the number of patterns produced, the average size of the patterns, and the cost per pattern. This to be able to choose the "best" algorithm (if any) and set of patterns for the next step of the study.
- *Quality assessment:* Data mining commonly requires an expert to validate discovered knowledge. In the context of mashup model pattern, we are specially interested in studying the understandability, usefulness, reusability and novelty of patterns, and we would like to study whether this validation can be crowdsourced as well by implementing two different validation "algorithms" to be compared:
  - *Expert assessment* is done manually by ourselves (experts of the field). This assessment serves as ground truth for the comparison.
  - *Crowd assessment* is done by the workers and represents the object of the study. The task is the same as the one for the experts.
- *Research question 2:* Given the two assessments, we would like to take a decision on the applicability of crowdsourcing for the validation of mined model patterns.

The crowdsourcing logics of the two dependent studies can be abstracted as a configurable process:

1) *Load mashup models and initialize dataset*: to ease the work of the crowd, we use screen shots of the models as rendered in Yahoo! Pipes as dataset.
2) *Partition dataset and map to tasks*: the three mining tasks require 1, 3 and 10 models as input per task instance; the validation tasks require 1/3 of the dataset per expert (three researchers are involved) and 1 model per crowd task instance.
3) *Deploy tasks:* the tasks must be deployed on the CC and made available to the crowd and to the experts.
4) *Manage task execution and collect patterns*: tasks must be executed by the crowd and experts and patterns must be collected by the CC.
5) *Integrate results*: results from workers must be integrated into an output dataset, possibly filtering out results of workers with poor performance (based on control questions).

For presentation purpose, we keep the requirements and descriptions here simple; the complete details can be inspected in the references cited in this section. For instance, we omit the details of the control questions we use to assess the performance of workers or the input fields we add to the tasks to collect additional metadata.

8.2. Process Model

Figure 21 illustrates a possible BPMN4Crowd process model implementing the whole study in an integrated fashion. The process starts with loading the screen shots of the models (also called "pipes" in Yahoo! Pipes) from a

Figure 21. The crowdsourcing process of the crowd-based pattern mining scenario. All except one crowd tasks use a marketplace tactic; the Expert Assessment task uses a mailing list tactic for pre-defined experts. In line with our prototype implementation, we omit the Data Merge task after the crowd tasks.

Figure 22. Task design for the selection, description and rating of mashup model patterns.

pre-filled repository. Then the process proceeds with three parallel branches, one for each crowd mining approach to study. The Naive branch splits the dataset into subsets of size 1 using the SplitWith(n) operator; the Random3 and ChooseN branches do the same with subset sizes of 3 and 10. Then, the actual mining is crowdsourced via three marketplace crowd tasks, each followed by a Filter operator dropping patterns that have less than 2 components selected, are not connected, and do not have all input fields filled. Then all collected patterns used by a researcher to compute a set of metrics (number of patterns, average pattern size, cost per pattern) that, in turn, are used by a researcher to choose the algorithm with the best performance.

Starting from the set of patterns of the chosen algorithm, the process proceeds with the study of whether the crowd can be used to validate patterns. It splits the patterns into 3 equal subsets to be validated by the three experts and subsets of size 1 for the workers. The process models the involvement of the experts as a crowd task, in that it uses a mailing list tactic to assign the tasks to them. The crowd assessment is instead again performed using a marketplace tactic. The process ends with a researcher comparing the results manually and taking a final decision on the research questions.

8.3. Implementation

Figure 22 and Figure 23 are screen shots of the task pages implemented to crowdsource the Naive pattern identification task and the pattern validation tasks (for both experts and the crowd), respectively; the screen shots of the Random3 and ChooseN tasks can be found in [26].

Given these task implementations, the modeling of the process in Figure 21 inside the BPMN4Crowd editor is straightforward. We replicated the logic within the editor, configuring each task according to the conventions explained in the previous section. The resulting process model is shown in Figure 24, in which, for better readability, we

Figure 23. Task design for the validation of collected mashup model patterns. We omit the initial description, gold questions for worker assessment, and additional inputs.

Figure 24. Implementation and configuration of the crowdsourcing process of Figure 21 in the BPMN4Crowd editor. Annotations correspond to task configurations and make them explicit in the model.

graphically annotate tasks with the parameters that must be specified as task properties in the editor. To model the process we decided to use the crowd task with predefined tactics. The figure shows that the implemented editor and runtime environment support all the necessary features described previously in this article.

Figure 19 shows the editor with the first part of the process in the modeling canvas along with the details of the parameter settings for the Naive crowd task. Specifically, the Naive crowd task uses the marketplace tactic illustrated in Figure 11 with an evaluation of the results based on gold data, as presented in Figure 15(c). The reward of the task is 0.50$ upon successful validation. The description of the task tells the goal and the rules of the task. The deadline is set to 1 month. The page URL, the interface of the task, points to the page shown in Figure 22. The input data object is set to 'Individual pipes" and the output data object is set to "Patterns N". The other crowd tasks are configured similarly. The figure also annotates data transformation tasks with their configurations. For example, in the first branch, the Split task is configured to divide the set of models into sets of 1 element each.

All crowd tasks are implemented with standard Web technologies (HTML, CSS, JavaScript and Java on the server side) and deployed on a Web server hosted by Amazon Web Services. Mashup models are identified (the metadata exchanged between the task pages and the CC) using the file names of the models as stored on the Web server. The human and machine tasks are standard BPMN constructs (assigned to ourselves and an initial pattern loading Web service) and, as such, are managed directly by the Activiti BPMN engine, which can be deployed on a common Web server. Activiti natively exposes a set of API that the CC can use to deploy, execute, and control the process.

8.4. Analysis

The outlined implementation shows how BPMN4Crowd eases both the design and the conduct of the described study. For the actual study we describe in [26], we however did not yet have the CC and BPMN4Crowd and, therefore, had to implement everything from scratch and by hand (so far, we only studied the first part of the algorithm, that is, the comparison of the three pattern mining tasks). For instance: We developed three separated web applications, one for each pattern mining task, each comprising an own data store, data management logic, task UI, data-task instance assignment logic, and pattern validation logic. Figure 22 is the screen shot of the original task page used. We manually configured the database underlying the applications and implemented the logic to load the pipes dataset both in JSON (for all pipes extracted of the "most popular pipes" category) and PNG (for the sample used to crowdsource the task). We implemented the algorithms (one for each application) to split the dataset as needed and manually mapped each partition to a task instance. We manually created as many task instances as we had data partitions, so as to have a task instance, thus a worker, for each partition. As concrete crowdsourcing

platform we used CrowdFlower (www.crowdflower.com), which features a marketplace tactic with gold data for woker assessment. For each crowd task, we created a task on CrowdFlower that contained the questionnaire with the gold questions, to evaluate workers' expertises, plus a link to our web applications (the task pages) where workers could perform the pattern identification tasks. Gold data were used as discriminant to accept or reject results, filter out bad results, and reward workers. The filtering was done manually: data were downloaded from CrowdFlower and from our web applications, matched and filtered according to the criteria outlined above. In the same filtering operation we also checked if the patterns met the requirements of what we called "good patterns" (minimum two components, connected model, meaningful text inputs), eliminating the ones that did not. The three crowdsourcing experiments were run sequentially, first Naive, then Random3, then ChooseN. When all three experiments were completed and all data integrated (manually), we compared the results and decided which was the best approach to mine model patterns with the crowd. For curiosity, it turned out that the Naive approach produced most patterns, with good variability in terms of pattern sizes, and at the lowest overall cost (after applying all checks) [26]. In [27] we further compare the performance of Naive against that of an automated mining algorithm, obtaining similar results.

The BPMN4Crowd process model in Figure 24 shows well how much of the described manual effort can instead be automated with the CC. The two approaches are both effective, yet they require a significantly different amount of time for their development. Of course, the implementations of the individual task pages must still be done manually by the crowdsourcer. However, the respective web applications no longer need to take care of managing task instances and data-task instance assignments, as this is now done by the CC. What is needed, instead, is the exchange of the respective metadata with the CC, which is however a minor effort compared to the full implementation of the data management and task instantiation logics. Given the previous implementation of these task pages, their reuse in the CC is straightforward. The Load Data task that fetches the pipes models from the local repository can be implemented as common web service.

Given these implementations, the use of the CC and of BPMN4Crowd allows one to automate the management and execution of 14 tasks out of the 17 tasks in Figure 24 (next to the control flow and propagation of data): the machine task to load data is executed automatically by calling the respective web service; the 8 data transformation tasks are done inside the CC, and the 5 crowd tasks (including the expert assessment task) are deployed, executed and integrated automatically. What is left as pure manual work are the 3 human tasks, which require the direct involvement of the researchers (in addition to the involvement into the expert evaluation, which is however part of the study design and not attributable to the coordination of the study). In addition, the availability of the model of the process and the possibility

to easily change parameters of tasks or to re-configure the model, enables the researchers to fine-tune the process and to re-run experiments without spending additional time for coding or coordinating tasks.

9. Comparative Analysis and Limitations

The target users of BPMN4Crowd are process modelers that want to crowdsource work in a fashion that is integrated with their BPM practices. Today, BPM typically involves the design and execution of process models expressed in BPMN. The possibility to intermix tasks executed by a crowd with tasks executed by other actors, such as the crowdsourcer himself/herself or a web service, by means of an extension of BPMN makes the creation of integrated and structured crowdsourcing processes feasible. Compared to state-of-the-art crowdsourcing platforms, the binomial CC/BPMN4Crowd provides for this integration in a natural fashion. We thus consider the approach described in this article particularly effective and powerful in all those cases where the outlined kind of integration is needed.

In this respect, it is important to realize that advanced crowdsourcing scenarios are generally not trivial and require good domain knowledge by the process modeler as well as working knowledge of crowdsourcing. The CC and BPMN4Crowd both enable new features (e.g., tactics) and simplify their use (e.g., via suitable abstractions and repositories of reusable models); however, the intrinsic, conceptual complexity of advanced CS processes cannot be eliminated. This is true also for other crowdsourcing approaches that target similar composite CS processes.

In the Figure 25 we compare the CC and BPMN4Crowd with a basic marketplace crowdsourcing platform (Mechanical Turk) and three other platforms for the management of complex CS processes (TurKit, CrowdForge and Crowd-Searcher – see the next section for detailed descriptions). The requirement for the selection of these platforms is either their free accessibility online (Mechanical Turk, Crowd-Searcher) or the availability of their source code (TurKit, CrowdForge). The comparison is based on a selection of features that impact the complexity of developing CS processes; we group the features into task support, control flow support, data management support, and development support features. The more features are supported by a platform, the more the platform eases the development of processes that require these features and the less manual effort is needed.

The CC is the only platform that supports human tasks, that is, integration with conventional BPM practice, while CrowdSearcher comes with support for tasks per- formed on social networks (Facebook). The CC is the only platform that supports custom crowdsourcing tactics, and, thanks to the use of BPMN, it supports all control flow constructs. It also supports custom validation logics, while the other

approaches typically support pre-defined validation logics, if any. One distinguishing feature is that the CC manages metadata, not actual data, and the crowdsourcer decides where to store the data on his own. This choice reduces the amount of data transferred over the network and allows the crowdsourcer to protect data that are sensitive (e.g., images with nudity) or subject to local regulations (e.g., healthcare data of citizens in some countries must be stored on country-local servers).

The previous comparison shows that for simple or individual crowd tasks that do not require custom logics or that do not have special data management requirements, the CC can be used but it may result more complex than necessary. To crowdsource a single crowd task, a conventional CS platform (e.g., Mechanical Turk) may be easier to use and manage. To crowdsource a crowd task that is of pure data processing nature, a dedicated instrument like CrowdForge or CrowdDB [31] may be more efficient. Yet, already if the crowd task is to be integrated with existing legacy business processes, the CC may again be more efficient.

The choice of extending an existing process modeling language and related technologies has a threefold benefit (next to the native support for the control flow features): (i) it can foster the adoption of BPMN4Crowd, since BPMN is widely known and adopted to model business processes; (ii) people that already use BPMN can easily integrate and extend their processes with tasks executed by the crowd; and (iii) the implementation of the necessary runtime environment can rely on existing, robust infrastructure services (e.g., the BPMN engine). People that are not yet familiar with business process modeling will have to first get acquainted with that, then BPMN4Crowd requires learning only few additional constructs for crowdsourcing; the resulting complexity is only slightly bigger than that of pure BPMN.

Of course, the choice of BPMN as a starting point for the modeling of crowdsourcing processes also comes with a cost: the need to stick to the abstractions and conventions of the language. This implies that not all development aspects are represented graphically and instead may require the joint use of modeling constructs and configuration parameters. This is, for instance, the case of our crowd tasks that come with a set of parameters to configure the URL of the task page, the validation process to be used, etc. Since this is common practice in BPMN, we however confident that the benefits and abstractions of the language outweigh this apparent limitation.

With the CC, we propose a new type of "crowdsourcing platform" that comes more in the form of a library of basic crowdsourcing APIs and less as a self-sufficient platform. Only the implementation of suitable CS tactics brings the CC into life. Yet, the current shortcoming of the CC is that it does not yet have its own crowd – of course, this is also due to the fact that it is still a research prototype, more than a commercial tool. This of course limits the possibility to test the execution of crowdsourcing processes with real workers. To overcome this problem and enable the crowdsourcer to be immediately productive, we started integrating the CC with existing platforms, such as Amazon Mechanical Turk, and

to enable posting tasks on external platforms. The prototype implementation shown in Figure 19, for instance, features a TurkTask task type. While this provides access to an existing crowd, it also introduces a limitation: it is not possible to implement own tactics for this kind of tasks, as these are hard-coded inside the target platforms.

Regarding the implementation of crowd tasks, our choice is to ask the crowdsourcer to create his/her own UIs for tasks via separated web pages that can be included in the CC front-end. On the one hand, this gives the crowdsourcer the freedom to create and fine-tune UIs that suite best the tasks to be crowdsourced (this is a crucial aspect for the success with crowdsourcing) and full control over the published and collected data. On the other hand, this requires some programming effort to create the task pages and to set up the exchange of metadata between the pages and the CC. This may hinder the adoption of the CC by less skilled crowdsourcers. To support also this kind of crowdsourcers, we aim to develop a dedicated, hosted tool that helps crowdsourcers to create task UIs and provides a set of ready-to-use page templates that can easily be configured and reused inside CS processes.

Finally, the focus of this work was specifically on crowdsourcing processes, that is, on the control flow perspective of advanced crowdsourcing scenarios and less on declarative aspects like advanced worker selection policies. The current implementation provides for worker selection based on workers' profile properties. It is however not yet able to express more complex constraints, such as the four eyes principle ("the performer of one task cannot participate in a validation task of its own work"). Enforcing this kind of constraints requires maintaining not only worker profile information but also the history of the task instances and the specific data items they worked on. The metadata model implemented by the CC already accommodates this kind of tracking, the necessary API support still needs to be implemented.

10. Related work

The work presented in this article proposes a language and a platform to create and enact crowdsourcing processes that are composed of crowd, data, machine, and human tasks. In literature, similar solutions have been proposed, interpreting the same problem along three different lines of thought: *procedural programming*, *parallel computing*, and *process modeling*.

10.1. Procedural Programming Approaches

The procedural programming approaches mostly extend existing programming languages or define new languages able to cover crowdsourcing aspects. For instance, Turkit [32] is a programming language based on JavaScript that adds support for human computation. Turkit uses Amazon Mechanical Turk as platform to execute human tasks. Programmers can write software applications that use both human and machine computations. Automan [33] is another

system for human computation that integrates crowd tasks into the Scala.

These works abstract crowdsourcing logic at a programming language level. Programming languages allow crowdsourcers to create flexible crowdsourcing logics, yet they neglect support for tactic definitions and configurations. As we showed in our case study, implementing a crowdsourcing process is generally not an easy task, even with the help from dedicated frameworks. Crowdsourcers that rely on programming languages are forced to code all aspects of their crowdsourcing process, a task that is not trivial and that makes it harder to maintain and evolve processes. Also the use of procedural programming is therefore limited to people that are programmers and is out of the reach of people without the necessary software development skills

10.2. Parallel Computing Approaches

The parallel computing approaches interpret a crowd task as a complex process that can be decomposed into a set of smaller and simpler tasks that can be executed in parallel. Crowdforge [18] and Turkomatic [5] adapt, for example, the Map-Reduce approach [17] to crowdsource complex work. Both frameworks model tasks as a set of *split* and *recombine* tasks executed by workers. Workers have the possibility to solve a task or to split it in smaller tasks (sub-tasks), in which case the workers are in charge of recombining the solutions of the sub-tasks into a unique solution. Jabberwocky [11] implements a similar map-reduce approach. Jabberwocky is composed of three different pieces: Dormouse provides operations to interact with machines and humans; ManReduce, similar to Crowdforge and Turkomatic, implements the Map-Reduce algorithm using Dormouse as workers; Dog, a scripting language that can be used to specify the details of applications (e.g., defining users or task goals).

The works of this type give one the possibility to bring together human and machine computations in a single application. Compared to our approach, these systems focus on crowdsourcing processes composed of parallel executions of tasks, which can be implemented as well with BPMN4Crowd, but they neglect support for more complex, generic process logics. Most of the discussed approaches require some level of scripting expertise, which is an abstraction that may not be suitable for crowdsourcers that are business analyst or that are not programming experts.

10.3. Process Modeling Approaches

The process modeling approaches are most similar to the interpretation we follow in this article. CrowdWeaver [8] is, for instance, a process modeling tool built on top of Crowd-Flower. The CrowdWeaver system offers a visual tool, with a graphical modeling notation, to create and execute data-driven processes bringing together machine and crowd tasks, the former providing data transformation capabilities. The system abstracts the native operations and logic of Crowd-Flower; it does not support crowd tactics, other crowd-

sourcing platforms, or the integration with legacy business processes. CrowdLang [6], [34] is a model-driven language for the modeling of generic human computations. Similarly to BPMN4Crowd, it allows one to express data and control flows and to describe application logics as tasks executed by the crowd or machines. CrowdLang also supports operations for the management of data. Its modeling notation features workflow modeling constructs like rounded rectangles for tasks and diamonds for conditions, but it also introduces additional constructs, such as the decision, which is modeled with a circle shape. More importantly, each task instance in CrowdLang must be modeled as an individual task, thus crowdsourcers must create a task in the model for each task instance they need. This may result in a huge number of similar tasks and is not very efficient from a modeling perspective. [35] proposes methods and tools for designing crowd-based workflows as interacting tasks and also propose a set of typical workflow patterns that help crowdsourcers implement their crowdsourcing processes (BPMN4Crowd is also powerful in expressing reusable logics and patterns). The proposed modeling language is accompanied by a runtime environment called CrowdSearcher that supports deploying tasks on both crowdsourcing platforms and social networks and monitoring crowdsourcing processes at runtime. The environment also provides explicit support for data storage and processing.

The three approaches are suitable to model crowdsourcing processes. However, they do not provide crowdsourcers with the flexibility to also define own tactics. Also, they all feature an own, proprietary modeling notation, which may not be straightforward to crowdsourcers. We instead extend BPMN and support the integration with most of the state-of-the-art BPM tools, which makes modeling more accessible to people that already know BPMN and integration with legacy processes easier.

Many researchers [36], [37], [38], [39] have highlighted the benefits of the integration of systems that manage human- and machine-based work with BPM or workflow management in crowdsourcing. This integration can be achieved in various ways, next to the one we describe in this article. For example, Khazankin12 present an approach to an adaptable execution of business processes on top of a crowdsourcing platform. Its main peculiarity is the ability to determine optimal task publishing procedures, adapting them to modeled process deadlines and minimizing the reward cost. The approach optimizes process execution, a feature the CC could integrate in the future and benefit from. Skopik11 use Human-Provided Services (HPS) [40], which abstract human capabilities as web services, easing the interaction with people and the integration with SOA systems. The supported processes use crowd tasks that are executed by people taken from a social network. Schall2012 extend BPEL4People [41], an extension of the web service orchestration language BPEL [42], with parameters to specify requirements specific to crowdsourcing, such as user skills and deadlines. Both approaches tackle the problem of crowdsourcing by abstracting worker capabilities as configurations of tasks or services.

In a more general context, *human computation*, i.e., the use of human capabilities to solve problems, has recently been investigated and adopted as a way to solve tasks that machines cannot solve. Crowdsourcing as proposed by [1] is one possible example of human computation. Another example of human computation is *social computing*, which studies and leverages on social human behaviors facilitated by computers (examples are blogs and wikis) [43]. This line of thought has inspired also a practice called *social BPM*, a recent trend of research that fuses social interactions as enabled by social software and business process management [44], [45]. BPM is a multi-faceted domain and social capabilities have been used, for instance, to improve the design of processes [46] or to enable the coordination and collaboration of multiple actors during process execution [47]. Then there are social BPM approaches that extend business process languages, similar to our approach, however to leverage on the capabilities of people acting in generic social networks, rather than in dedicated crowdsourcing platforms. BPM4People [48], [49] proposes, for instance, a set of extensions of BPMN that enable the modeling and deployment of process-aware, social interactions over social networks, such as the collection of votes or comments. BPM4People supports a social computing paradigm in which work is mostly executed implicitly, and actors may not be aware of taking part in a task or work. Our focus is instead specifically on crowdsourcing platforms, where tasks are defined explicitly and possibly rewarded and where actors participate consciously either to offer or to execute work.

11. Conclusion

This article advances the state of the art on crowdsourcing along three core directions: First, it establishes *crowdsourcing processes* as processes that involve individual actors, machines and workers as first-class development concern, going beyond the conventional focus on crowd tasks only. Second, it proposes an alternative interpretation of crowdsourcing platform, the Crowd Computer, that comes as a set of crowdsourcing-specific APIs and allows the crowdsourcer to implement *custom crowdsourcing tactics* for the crowdsourcing of individual tasks. Third, it fosters the *integration* of crowdsourcing capabilities with state-of-the-art business process management practices and legacy systems. Crowdsourcing processes allow the crowdsourcer to specify advanced crowdsourcing and process logics at a level of abstraction that has proven suitable for the specification and coordination of tasks, i.e., business processes. Crowdsourcing tactics provide the crowdsourcer with unprecedented control over how work is advertised, assigned, executed, integrated, evaluated, and rewarded. The use of off-the-shelf BPM technology finally makes the two crowdsourcing instruments available as a service that can be integrated flexibly into existing software.

The tangible contributions of the article are (i) a prototype implementation of the Crowd Computer available as open source software, (ii) an extension of BPMN,

BPMN4Crowd, specifically tailored to the needs of crowd-sourcing, (iii) a set of reusable tactic process models, (iv) a set of reusable validation process models, (v) a visual editor for the modeling of BPMN4Crowd processes, and (vi) a case study that discusses the benefits and shortcomings of the approach as a whole in the context of a concrete human computation scenario.

In our future work, we plan to provide an online repository where crowdsourcers can search for, retrieve and share tactics, validation processes, and crowdsourcing processes to foster knowledge reuse and exchange. Similarly, we are considering implementing a hosted instrument for the visual design of crowd task pages, starting from a set of pre-defined templates for recurrent task types, e.g., questionnaires, comparisons, voting tasks, text translations, etc. We also intend to use the CC and BPMN4Crowd ourselves for the design of other crowdsourcing experiments, such as the case study described in this article.

References

[1] J. Howe, *Crowdsourcing: why the power of the crowd is driving the future of business*, 1st ed. New York, NY, USA: Crown Publishing Group, 2008.

[2] C. Baun, M. Kunze, J. Nimis, and S. Tai, *Cloud computing - web-based dynamic it services*. Springer, 2011.

[3] Massolution, "The crowd in the cloud: exploring the future of outsourcing," White Paper, January 2013.

[4] Crowdsourcing Week, "2014 Global Crowdsourcing Pulsecheck: 1st Annual Survey Topline Results," http://www.slideshare.net/crowdsourcingweek/2014-global-crowdsourcing-pulsecheck-1st-annual-survey-topline-results, August 2014.

[5] A. Kulkarni, M. Can, and B. Hartmann, "Collaboratively crowdsourcing workflows with turkomatic," ser. CSCW '12. New York, NY, USA: ACM, 2012, pp. 1003–1012.

[6] P. Minder and A. Bernstein, "Crowdlang - first steps towards programmable human computers for general computation," in *Human computation*, 2011.

[7] D. Schall, B. Satzger, and H. Psaier, "Crowdsourcing tasks to social networks in bpel4people," *WWW*, pp. 1–32, 2012.

[8] A. Kittur, S. Khamkar, P. André, and R. Kraut, "Crowdweaver: visually managing complex crowd work," in *CSCW '12*. New York, NY, USA: ACM, 2012, pp. 1033–1036.

[9] P. Kucherbaev, S. Tranquillini, F. Daniel, F. Casati, M. Marchese, M. Brambilla, and P. Fraternali, "Business processes for the crowd computer," in *Business Process Management workshops*, ser. Lecture Notes in Business Information Processing, M. Rosa and P. Soffer, Eds. Berlin Heidelberg: Springer, 2013, pp. 256–267.

[10] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller, "Turkit: human computation algorithms on mechanical turk," in *UIST '10*. New York, NY, USA: ACM, 2010, pp. 57–66.

[11] S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar, "The jabber-wocky programming environment for structured social computing," in *UIST'11*, 2011, pp. 53–64.

[12] P. G. Ipeirotis, "Analyzing the amazon mechanical turk marketplace," *XRDS*, vol. 17, no. 2, pp. 16–21, Dec. 2010.

[13] R. Cavallo and S. Jain, "Efficient Crowdsourcing Contests," in *Proceedings of AAMAS 2012 - Volume 2*, 2012, pp. 677–686.

[14] B. Satzger, H. Psaier, D. Schall, and S. Dustdar, "Auction-based crowdsourcing supporting skill management," *Inf. Syst.*, vol. 38, no. 4, pp. 547–560, Jun. 2013.

[15] M. Hirth, T. Hoßfeld, and P. Tran-Gia, "Analyzing costs and accuracy of validation mechanisms for crowdsourcing platforms," *Mathematical and Computer Modelling*, vol. 57, no. 11-12, pp. 2918–2932, June 2013.

[16] O. M. G. (OMG), "Business process model and notation (bpmn) version 2.0," 2011.

[17] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[18] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut, "Crowdforge: crowdsourcing complex work," in *UIST'11*, 2011, pp. 43–52.

[19] M. Weske, *Business process management: concepts, languages, architectures*. Springer, 2007.

[20] P. G. Ipeirotis, F. Provost, and J. Wang, "Quality management on amazon mechanical turk," in *HCOMP '10*. New York, NY, USA: ACM, 2010, pp. 64–67.

[21] S. Dow, A. Kulkarni, S. Klemmer, and B. Hartmann, "Shepherding the crowd yields better work," in *CSCW 2012*, 2012, pp. 1013–1022.

[22] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H. R. Motahari-Nezhad, E. Bertino, and S. Dustdar, "Quality control in crowdsourcing systems: issues and directions," *IEEE IC*, vol. 17, no. 2, pp. 76–81, 2013.

[23] M. Soleymani and M. Larson, "Crowdsourcing for affective annotation of video: Development of a viewer-reported boredom corpus," in *ACM SIGIR 2010 Workshop on Crowdsourcing for Search Evaluation (CSE 2010)*, 2010, pp. 4–8.

[24] T. Hoßfeld, C. Keimel, M. Hirth, B. Gardlo, J. Habigt, K. Diepold, and P. Tran-Gia, "Best Practices for QoE Crowdtesting: QoE Assessment With Crowdsourcing," *IEEE Transactions on Multimedia*, pp. 541–558, 2014.

[25] D. Oleson, A. Sorokin, G. P. Laughlin, V. Hester, J. Le, and L. Biewald, "Programmatic gold: targeted and scalable quality assurance in crowdsourcing," in *Human computation*, 2011.

[26] C. Rodriguez, E. Zaupa, F. Daniel, and F. Casati, "Crowd-based pattern mining - on the crowdsourcing of reusable knowledge identification from mashup models," UNITN, Deliverable, 2013.

[27] C. Rodriguez, F. Daniel, and F. Casati, "Crowd-based mining of reusable process model patterns," in *BPM 2014*, 2014, p. In print.

[28] S. Roy Chowdhury, C. Rodríguez, F. Daniel, and F. Casati, "Baya: assisted mashup development as a service," in *WWW 2012 companion*. New York, NY, USA: ACM, 2012, pp. 409–412.

[29] S. Roy Chowdhury, F. Daniel, and F. Casati, "Efficient, interactive recommendation of mashup composition knowledge," in *ICSOC'11*, 2011, pp. 374–388.

[30] F. Daniel and M. Matera, *Mashups: Concepts, Models and Architectures*. Springer, 2014.

[31] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "CrowdDB: Answering Queries with Crowdsourcing," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, 2011, pp. 61–72.

[32] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller, "Exploring iterative and parallel human computation processes," in *HCOMP '10*. New York, NY, USA: ACM, 2010, pp. 68–76.

[33] D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor, "Automan: a platform for integrating human-based and digital computation," *SIGPLAN Not.*, vol. 47, no. 10, pp. 639–654, Oct. 2012.

[34] P. Minder and A. Bernstein, "Crowdlang: a programming language for the systematic exploration of human computation systems," in *Social informatics*, ser. Lecture Notes in Computer Science, K. Aberer, A. Flache, W. Jager, L. Liu, J. Tang, and C. Guéret, Eds. Springer Berlin Heidelberg, 2012, vol. 7710, pp. 124–137.

[35] A. Bozzon, M. Brambilla, S. Ceri, A. Mauri, and R. Volonterio, "Pattern-based specification of crowdsourcing applications," in *ICWE 2014*. Springer, 2014, pp. 218–235.

[36] S. Curran, K. Feeney, R. Schaler, and D. Lewis, "The management of crowdsourcing in business processes," in *Integrated network management-workshops, 2009.*, 2009, pp. 77–78.

[37] M. Vukovic, "Crowdsourcing for enterprises," in *2009 world conference on Services*, july 2009, pp. 686 –692.

[38] G. La Vecchia and A. Cisternino, "Collaborative workforce, business process crowdsourcing as an alternative of bpo," in *ICWE'10*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 425–430.

[39] A. Kittur, J. V. Nickerson, M. Bernstein, E. Gerber, A. Shaw, J. Zimmerman, M. Lease, and J. Horton, "The future of crowd work," in *CSCW '13*. New York, NY, USA: ACM, 2013, pp. 1301–1318.

[40] D. Schall, H.-L. Truong, and S. Dustdar, "Unifying human and software services in web-scale collaborations," *IEEE Internet Computing*, vol. 12, no. 3, pp. 62–68, May 2008.

[41] Active Endpoints, Adobe, BEA, IBM, Oracle, SAP, "WS-BPEL extension for people (BPEL4People) version 1.0," Tech. Rep., June 2007.

[42] D. Jordan and J. Evdemon, "Web services business process execution language version 2.0," OASIS, OASIS Standard, Apr. 2007, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[43] A. J. Quinn and B. B. Bederson, "Human computation: a survey and taxonomy of a growing field," in *CHI '11*. New York, NY, USA: ACM, 2011, pp. 1403–1412.

[44] P. Johannesson, B. Andersson, and P. Wohed, "Business process management with social software systems – a new paradigm for work organisation," in *BPM workshops*, 2009, pp. 659–665.

[45] S. Erol, M. Granitzer, S. Happ, S. Jantunen, B. Jennings, P. Johannesson, A. Koschmider, S. Nurcan, D. Rossi, and R. Schmidt, "Combining bpm and social software: contradiction or chance?" *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 22, no. 6-7, pp. 449–476, Oct. 2010.

[46] A. Koschmider, M. Song, and H. A. Reijers, "Social software for business process modeling." *JIT*, vol. 25, no. 3, pp. 308–322, 2010.

[47] F. Dengler, A. Koschmider, A. Oberweis, and H. Zhang, "Social software for coordination of collaborative process activities," in *BPM workshops*, 2011, pp. 396–407.

[48] M. Brambilla, P. Fraternali, and C. Vaca, "A notation for supporting social business process modeling," in *Business Process Model and Notation*, vol. 95. Springer Berlin Heidelberg, 2011, pp. 88–102.

[49] M. Brambilla, P. Fraternali, and C. K. Vaca Ruiz, "Combining social web and bpm for improving enterprise performances: the bpm4people approach to social bpm," in *WWW 2012*, 2012, pp. 223–226.