BPMN Task Instance Streaming for Efficient Micro-Task Crowdsourcing Processes

Stefano Tranquillini[1], Florian Daniel[1,2], Pavel Kucherbaev[1], and Fabio Casati[1]

[1] University of Trento – DISI, Via Sommarive 9, I-38123, Povo (TN), Italy
{tranquillini,daniel,kucherbaev,casati}@disi.unitn.it
[2] Tomsk Polytechnic University, Belinskya Street 30, 634050, Tomsk, Russia

Abstract. The Business Process Model and Notation (BPMN) is a standard for modeling and executing business processes with human or machine tasks. The semantics of tasks is usually discrete: a task has exactly one start event and one end event; for multi-instance tasks, all instances must complete before an end event is emitted. We propose a new task type and streaming connector for crowdsourcing able to run hundreds or thousands of micro-task instances in parallel. The two constructs provide for task streaming semantics that is new to BPMN, enable the modeling and efficient enactment of complex crowdsourcing scenarios, and are applicable also beyond the special case of crowdsourcing. We implement the necessary design and runtime support on top of CrowdFlower, demonstrate the viability of the approach via a case study, and report on a set of runtime performance experiments.

Keywords: Crowdsourcing processes, task instance streaming, BPMN

1 Introduction

BPMN [15] is the most representative example of the state of the art in business process modeling. Its core modeling constructs are tasks and control flow connectors. Both constructs follow semantics that stem from their roots in office automation: tasks are *atomic*. They express indivisible pieces of work that have a well-defined start and end, and do not provide insight into what is going on inside a task while in execution. In the basic setting, one task corresponds to one runtime instance of the task. However, the notation also supports *multi-instance tasks* that allow the execution of multiple runtime instances either in parallel or in sequence. State-of-the-art engines commonly implement multi-instance tasks following the same atomic start/end semantics of basic tasks (the end event fires only when all instances have completed), although the BPMN specification [15] also envisions intermediate instance termination events for complex behavior definitions (p. 432).

There are however modeling scenarios that would benefit from more transparency, in order to be executed more efficiently. This is, for instance, the case of processes that run multiple instances of tasks in parallel. An extreme example is *crowdsourcing*, that is, the outsourcing of a unit of work to a crowd of people

via an open call for contributions [8]. Thanks to the availability of crowdsourcing platforms, such as Amazon Mechanical Turk (`https://www.mturk.com`) or CrowdFlower (`http://www.crowdflower.com`), the practice has experienced a tremendous growth over the last years and demonstrated its viability in different fields, such as data collection and analysis or human computation – all practices that leverage on *micro-tasks*, which are tasks that ask workers to complete simple assignments (e.g., label an image or translate a sentence) in exchange for an optional reward (e.g., few cents or dollars). The power of crowdsourcing is represented by the crowd, which may be huge and span the World, and its ability to process even thousands of task instances in short time in parallel.

However, not all types of work can easily be boiled down to simple micro-tasks, most platforms still require significant amounts of manual work and configuration, and there is only very limited support for structured work, that is, work that requires the integration of different tasks and multiple actors, such as machines, individuals and the crowd. We call these kinds of structured work *crowdsourcing processes*, since they require the coordination of multiple tasks, actors and operations inside an integrated execution logic [17].

Crowdsourcing processes therefore represent a problem where business process management (BPM) is expected to excel. The modeling and efficient enactment of crowdsourcing process is however still not well supported [11]. In particular, BPMN does not provide the right means to model processes that are as simple as, for example, asking the crowd to upload a thousand images in one task and then to label them in another task. The labeling task would start only once all images have been uploaded, not benefiting from the evident parallelization opportunities of the scenario. The tokens of Petri Nets [18] would allow one to deal with this kind of dynamic state, but BPMN does not support tokens.

In [17], we proposed BPMN4Crowd, a BPMN extension for the modeling of crowdsourcing processes that can be executed on our own crowdsourcing platform, the Crowd Computer; the approach uses the standard task termination semantics of BPMN. In this paper, we instead study the problem of micro-task parallelization in generic BPMN engines, making the following contributions:

- An extension of BPMN with a *new task and connector type* that provides full support for the streaming of outputs of completed micro-task instances to subsequent micro-task instances without requiring an overall task end event.
- An implementation of a *runtime environment* for crowdsourcing processes with micro-task instance streaming. The environment is distributed over a BPMN engine for the coordination of work, a state-of-the-art crowdsourcing platform for the micro-tasks, and an intermediate middleware.
- An implementation of a visual *design environment* with support for the extended BPMN modeling notation and the translation of extended models into standard BPMN for the engine and configuration instructions for the crowdsourcing platform and the middleware.
- A demonstration of the viability of the approach via a concrete crowdsourcing *case study* complemented by a *performance analysis* reporting on the execution time improvements that can be achieved.

2 Crowdsourcing processes

2.1 Scenario: transcription of receipts

The reimbursement of a business trip, such as the attendance of a scientific conference, is subject to the documentation of the incurred expenses. This documentation are the receipts that can be scanned and transcribed for the processing of the reimbursement. Transcribing a receipt is a small task that can be crowdsourced at low cost and with fast response times.

Let's imagine we would like to support the following crowdsourcing process: The admin reimbursing the travel expenses initiates the process by feeding it with the receipts (e.g., 40) collected from traveling employees. This causes the process to upload photos/images of the receipts onto an online crowdsourcing platform and to instantiate a transcription request for each individual receipt. Since the work by workers cannot be trusted in general, for each 2 transcriptions the process creates another task for the crowd that asks workers to check the transcriptions and fix them if necessary. Checking and fixing takes less time than transcribing, so each worker can process 2 items. Another task is used to classify the receipts, e.g., into flight tickets, hotel receipts, restaurant receipts, or similar. Classifying is simple, and it is reasonable to ask a worker to classify 4 receipts. The two tasks can be performed in parallel once transcriptions are available. Upon completion, an automatic email notifies the admin about the results.

2.2 Crowdsourcing processes and streaming opportunities

The described scenario presents all the characteristics of a crowdsourcing process as defined in the introduction, which indicates a process that, next to optional human and machine tasks, also contains tasks executed by the crowd, so-called crowd tasks. A *crowd task* represents a set of micro-tasks that are jointly performed by the *crowd* via an online *crowdsourcing platform*. A *micro-task* is performed by an individual *worker*, is commonly interpreted as a task that requires limited skills and limited time (from seconds to few minutes), and is remunerated with limited rewards (from cents to few dollars). Crowd tasks can be seen like BPMN multi-instance tasks that typically require large numbers of instances to be performed in parallel (we focus on micro-tasking and do not further study the case of contest- or auction-based crowdsourcing models). For example, the above scenario asks for 40 transcriptions, 20 controls of transcriptions (2 per task), and 10 classifications of receipts (4 per task).

Figure 1 illustrates the dependencies among the crowd tasks of the scenario and the benefits that could be achieved if the process supported the streaming of micro-task instances. With the term *micro-task instance streaming* we denote the streaming of micro-task instance end events while the respective crowd task is still in execution, that is, other micro-task instances of the crowd task are still in execution. Completed instances can be streamed, that is, their end events and output data, from a crowd task A to a crowd task B, causing the instantiation of micro-tasks of B as soon as the necessary number of micro-task instances of A

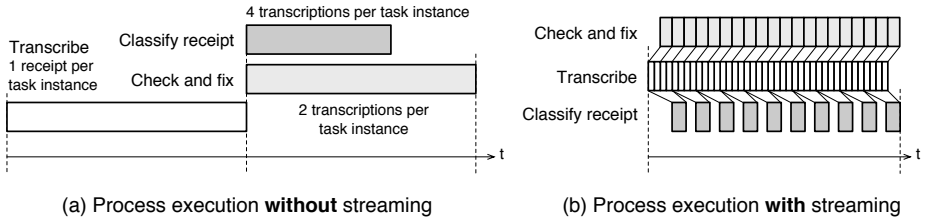

Fig. 1. The parallelization benefits of task instance streaming in crowdsourcing

have completed. For example, the transcription of 4 receipts causes the instantiation of 1 classification and 2 checks. The number of instances to be created thus depends on the data transformation logic between two crowd tasks: *grouping* outputs reduces the number of micro-task instances of the subsequent crowd task; *multiplying* outputs or *splitting* grouped outputs increases the number of micro-task instances of the subsequent crowd task. Data transformations may be needed to accommodate the mismatch between output and input data sizes of different crowd tasks, as exemplified in our reference scenario. The goal of grouping/splitting is usually that of keeping the overall effort of a micro-task constant in response to changing efforts required to process an input data item, e.g., one transcription requires roughly the same effort as four classifications of receipts. Multiplying outputs creates redundancy that can be used to increase the quality of outputs, e.g., a same receipt can be given to two different workers and their outputs can be checked for consistency.

3 Assumptions and approach

This work assumes that the crowdsourcer has working knowledge of both business process modeling with BPMN and crowdsourcing with a micro-tasking platform like CrowdFlower. Human and machine tasks are enacted by the business process engine running the BPMN process; crowd tasks are enacted by the crowdsourcing platform. The design of the UIs for the crowd tasks is done by the crowdsourcer inside the crowdsourcing platform. The platform provides programmatic access (via API) to the following abstract micro-task management functions: `uploadData` to associate micro-tasks with input data, `startInstance` to instantiate micro-tasks, `getInstanceStatus` to query the runtime status of micro-task instances, and `getInstanceOutput` to download results produced by a worker. For instance, both CrowdFlower and Amazon Mechanical Turk provide implementations of these abstract functions.

The approach to provide support for crowdsourcing processes is similar to the one already successfully adopted in prior works [6]: In order to provide insight into micro-task instance terminations, we extend the syntax and semantics of BPMN with two new modeling constructs, a *crowd task* and a *streaming connector*, that are specifically tailored to the needs of crowdsourcing. The streaming connector answers the need for a novel *data passing technique* that supports the

Fig. 2. Proposed modeling convention for micro-task instance streaming in BPMN

grouping, splitting and multiplication of streaming data as well as the passing of data between the process and the crowd tasks. We complement the language with a *visual editor* that allows the crowdsourcer to model his crowdsourcing process and equip the editor with a process *deployment tool* that transforms the process model with extended semantics into (i) a standards-compliant *BPMN process* and (ii) a set of *configurations* able to steer the crowdsourcing platform and to establish a communication channel between the platform and the engine. The extended BPMN process model contains the necessary logic for micro-task and communication management. Data streaming among crowd tasks is implemented via a simple *middleware* placed in between the BPMN engine and the crowdsourcing platform and able to monitor micro-task instance completions and to group, split or multiply respective output data. As soon as the monitor detects an expected number of micro-task instance completions, it assembles the respective data and sends to the process engine a *message* that can be intercepted by the process. Reacting to messages allows the process to create micro-task instances of dependent crowd tasks and to progress.

The goal is to provide crowdsourcing support as an extension of existing BPM practice, so as to be able to leverage on modeling conventions and software infrastructure that are already familiar to the BPMN-skilled crowdsourcer.

4 Streaming crowd tasks

Next, we introduce the BPMN modeling constructs that enable the modeling of crowsourcing processes, we discuss the options we have to transform crowdsourcing processes into standard BPMN processes, and we describe the concrete runtime infrastructure we implemented to support process execution.

4.1 Modeling micro-task instance streaming

Modeling crowdsourcing processes requires expressing tasks for the crowd and the propagation of outputs between workers. We propose to satisfy these requirements with two new constructs (Figure 2): crowd tasks and streaming connectors.

Crowd tasks are tasks that represent micro-tasks to be executed by workers inside a crowdsourcing platform. We identify crowd tasks using a crowd logo in the top left corner of the BPMN task construct. Crowd tasks cannot be expressed as simple multi-instance tasks, since these do not provide insight into

the completion of task instances and can therefore not be used to implement the expected streaming logic. The deployment and execution of crowd tasks further asks for a mediation between the process engine and the crowdsourcing platform, an aspect that goes beyond the conventional semantics of tasks in BPMN. We therefore opt for a new construct for crowd tasks that (i) provides for the execution of multiple instances of micro-tasks equipped with respective instance completion events, (ii) the deployment of the micro-tasks' input data on the crowdsourcing platform, and (iii) the start of the micro-task instances.

Streaming connectors connect two crowd tasks A and B and express that they are "followed" multiple times at runtime. How many times, depends on the *data transformation function* (Figure 3). If A has l micro-task instances and the connector *groups* m instances, B has l/m micro-task instances; if it *multiplies* instances of A by n (creating n copies by value) B has $l \times n$ micro-task instances. If it *splits* the outputs of A into its l items, B has l micro-task instances. The *flat* function hands items over as they are.

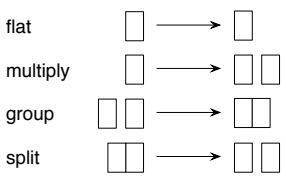

Fig. 3. Streaming data transformation functions.

The choice of a new type of connector is again justified by the need to express a logic that is not yet captured by any of the other BPMN constructs: the connector actually represents events (one for each individual data object generated by the data transformation function) that can only be handled by the internal logic of the subsequent crowd task B, which creates a micro-task for each event it receives from A. In addition, the event carries the output data produced by A, which task B uses to provide its micro-tasks with the necessary inputs. This turns the streaming connector into a data streaming connector for crowd tasks.

With the help of these two new constructs and the common constructs of BPMN, we are now able to model the crowdsourcing process described in our reference scenario as illustrated in Figure 4. The process starts with a common human task for uploading the receipts, followed by a crowd task for their transcription. The `Check and improve` and `Classify receipt` crowd tasks are executed in parallel and followed by a machine task sending the notification email with the results. The first crowd task takes as input the 40 receipts and produces respective transcriptions as output. Similarly, the outputs of the checking and classification crowd tasks are used as inputs of the final machine task. The very novel aspect of the model is the use of the streaming connector from the first to the other two crowd tasks. `Check and improve` is executed once for each couple of transcriptions (note the annotation of the connector) and `Classify receipt` once for each four transcriptions. Due to the data flow nature of the streaming connector there is no need to explicitly model data objects exchanged between crowd tasks. The data object in output of a crowd task (e.g., `Transcriptions`) is filled during task execution and is ready only when the last instance of its micro-tasks has completed, which also corresponds to the completion of the crowd task itself. This complies with the conventional semantics of BPMN.

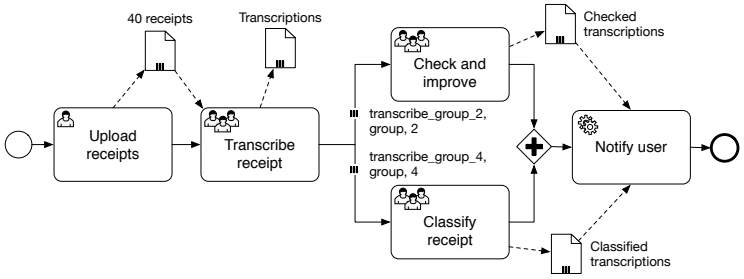

Fig. 4. Extended BPMN model of the receipt transcription crowdsourcing process

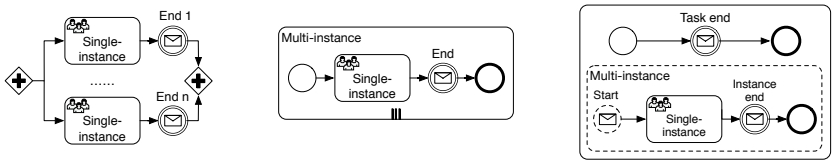

Fig. 5. Model transformation options

4.2 Model transformation

To provide BPMN modelers with an as familiar as possible modeling paradigm, we leverage on a standard BPMN engine and let it manage all and only those process execution aspects that are needed to coordinate the work of actors, machines, and the crowd. To do so we transform the process model created with abstract crowd tasks and streaming connectors into a BPMN-compliant model that can be executed and managed by an engine extended with runtime support for crowd tasks. The challenge is to overcome the mismatch between the requirement of providing insight into the execution of micro-task instances and the assumption that tasks are atomic. Next, we show how we approach the model transformation, then we focus on propagating and transforming data.

Transforming the crowdsourcing-specific constructs into executable BPMN constructs may be achieved in several ways. In particular, we identify the three approaches depicted in Figure 5 to make explicit the ***multiplicity of task instances*** at both model and execution level:

(a) *Parallel branches:* The straightforward option to execute several tasks independently in parallel is to create multiple parallel branches, each one executing a single crowd task instance in the crowdsourcing platform. The termination of instances can be captured via dedicated events by the streaming middleware. This approach in principle gives access to the results of each task instance individually. Yet, it is not convenient, since the number of branches to be created is proportional to the number of micro-tasks of the

crowd task, and the process model is only hard to read and manage. Especially the number of required instances may be large and can explode when more than a single crowd task is to be streamed. The approach also makes execution expensive, since all the branches are instantiated as soon as the process execution reaches the preceding gateway.

(b) *Multi-instance sub-processes:* This transformation option overcomes the problem of having several branches with the same logic modeled in parallel. The multi-instance sub-process behaves similar to the parallel branches: the full number of expected sub-processes is instantiated together at runtime when the first instance of the sub-process is started. However, the model is more modular, uses only one event type, and is better readable and maintainable.

(c) *Non-blocking event sub-processes:* To limit the number of parallel instances of sub-processes, it is possible to use a sub-process with a non-blocking event sub-process. The event sub-process is instantiated only upon a respective start event, and it does not block or alter the execution of the parent process. The start events can be generated dynamically at runtime as soon as the necessary input data are available; the `Instance end` event communicates task instance completions, the `Task end` event terminates the parent sub-process when all the instances have been processed.

Option (c) stands out as the most efficient transformation of the streaming constructs. However, although part of the BPMN standard, non-blocking event sub-processes are not (yet) reliably supported by state-of-the-art BPMN engines (our implementation is based on the open-source BPM platform Activiti). We therefore follow option (b), the multi-instance sub-processes, to transform models into executable format.

Given the resulting event-based nature of micro-task instance streaming, propagating data among crowd tasks requires (i) having access to the *data items* produced by each micro-task instance, (ii) enabling the *grouping/splitting/multiplication* of data items, and (iii) progressing the process based on *events*. Figure 6(a) shows a model pattern making use of the streaming connector; Figure 6(b) shows its transformed, executable model. ***Connector-crowd task pairs*** are mapped depending on their nature (streaming connectors connect crowd tasks only):

- *Standard control flow connectors followed by a crowd task* are transformed into one crowd task representing the execution of the micro-tasks inside the crowdsourcing platform and a multi-instance sub-process intercepting the respective micro-task instance terminations. The events to be intercepted are generated by the streaming middleware and contain the output data of the terminated micro-tasks. The `Store variables` script task takes the received data items and stores them in the global data object (if needed).
- *Streaming connectors followed by a crowd task* are transformed into a multi-instance sub-process that first intercepts instance terminations of the preceding micro-tasks and then runs the own micro-task instances and intercepts their terminations. Again, upon reception of each event the two script tasks store the respective data into a data object. The first script task uses a local data object, the second one fills again the global data object.

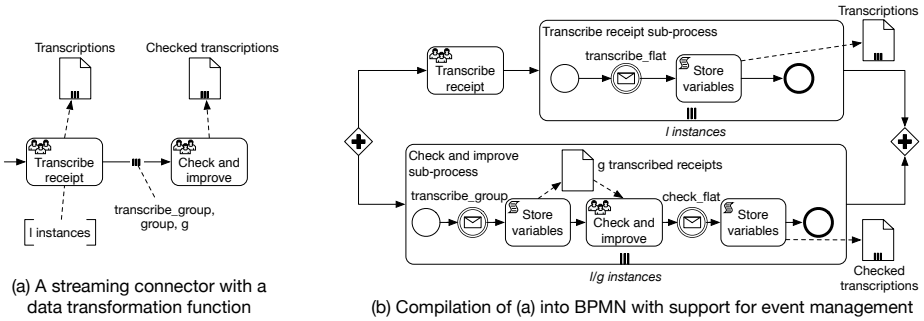

Fig. 6. Implementing micro-task instance streaming with data transformations

Note how the source sequence of crowd tasks is transformed into parallel branches of sub-processes that are synchronized via events. Incidentally, this resembles the streaming logic illustrated in Figure 1(b) also graphically.

In the executable model, crowd tasks have the following execution semantics: (i) read the data items specified as input, (ii) upload data to crowdsourcing platform, (iii) bind micro-task completions to suitable events in the middleware, (iv) start micro-tasks for each data item. The middleware and crowdsourcing platform start execution in parallel to the process engine. The engine waits for events from the middleware and processes them as specified in the model.

The model transformation logic further provides a convenient way to implement the ***data transformation functions*** illustrated in Figure 3. The key lies in the sensible use of events and the configuration of the streaming middleware:

- *Flat*: Micro-task instances are streamed as they terminate without applying any data transformation. This requires the generation and handling of one event for each termination. For instance, the `transcribe_flat` event in Figure 6(b) intercepts all instances of the `Transcribe receipt` micro-task.
- *Group*: Micro-task instance terminations are streamed only in groups. This requires the middleware to buffer instance terminations till the required number of terminations is reached and to emit an event that carries the collection of data items produced by the grouped instances. The process reacts to group events, like in the case of the `transcribe_group` event in Figure 6(b).
- *Split*: Micro-task instances are streamed as they terminate and their output data collections are split into their constituent elements, requiring the middleware to emit multiple events per termination. This function only applies to micro-tasks that produce collections of data items in output, as for instance the task `Classify receipt` in our reference scenario (four classifications).
- *Multiply*: The implementation of this data transformation function is similar to the split function, with the difference that data items are forwarded as they are, yet multiple events with data copied by value are generated and handled as separated events in the process.

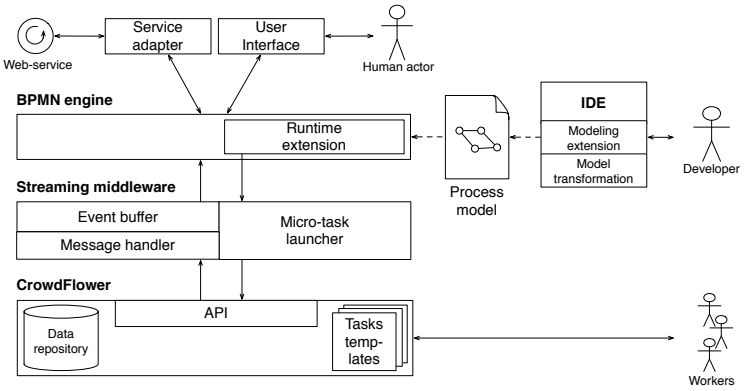

Fig. 7. Architecture of runtime environment for crowdsourcing processes with streaming support. The middleware deploys micro-tasks and manages events and data.

Thanks to this mapping logic, it is further possible to compute at transformation time the ***number of instances*** of each sub-process in the final model, starting from the number of micro-tasks of the first crowd task in the source model. We already discussed the necessary arithmetic in Section 4.1.

For a complete understanding of the proposed transformation logic, it is important to recall that the streaming connector can only be used between two crowd tasks and to note that a crowd task followed by a standard control flow connector implements the standard semantics of BPMN: the control flow connector is enacted only once all the micro-task instances of the preceding crowd task have terminated. This convention may lead to independent "islands" of streaming areas inside a process if multiple crowd tasks are separated by standard control flow constructs. For example, the crowdsourcing process modeled in Figure 4 could make use of other crowd tasks after the notification of the user about the completion of the transcription of the receipts. Each of these islands is transformed into a set of parallel branches and woven into the regular control flow structure of the source model as exemplified in Figure 6.

4.3 Runtime environment

Figure 7 illustrates the software architecture we implemented to run crowdsourcing processes. It is composed of three main blocks: (i) a *BPMN engine* where the processes are executed; (ii) a *streaming middleware* that manages the events and transforms data; and (iii) *CrowdFlower*, the crowdsourcing platform where micro-tasks are deployed and executed. To model a crowdsourcing process, the developer uses the *IDE* (an extension of the Activiti Modeler) that supports the *modeling extensions* presented previously. The process model is then *transformed* into an executable process and deployed into the BPMN engine. The engine is equipped with a *runtime extension* for the interaction with the mid-

dleware. Human actors and Web services are managed by the engine (Activiti, `http://activiti.org`) using its own user interface and service adapters.

The streaming middleware is composed of three blocks: The *micro-task launcher* deploys micro-tasks via the CrowdFlower API, given a task identifier, respective input data (if any), and a task template. The *message handler* and *event buffer* receive webhook calls from the CrowdFlower API when a task instance is completed, buffer output data, and create events for the BPMN engine.

CrowdFlower is the crowdsourcing platform where the streaming middleware deploys the micro-tasks for execution by the crowd. To enable the runtime deployment in CrowdFlower, *task templates* are designed at process modeling time and linked via suitable parameters to the crowd tasks in the BPMN model. Each template has to be designed to handle the correct number of data items in input. For example, the `Check and improve` template has two forms, one for each receipt to be processed. At runtime, the launcher feeds the templates with data from the BPMN engine, which are then available to workers as micro-tasks.

5 Case study and evaluation

5.1 Modeling and implementation

Modeling the process shown in Figure 4 in the extended Activity Modeler is a conventional BPMN modeling exercise with three exceptions: First, the crowd tasks make use of a new, dedicated modeling construct that allows the modeler to clearly identify them inside the model and to configure it's internal logic (remember Figure 2(a)). Second, streaming connectors are modeled as control flow connectors with a suitable annotation, as shown in Figure 2(b). The annotation turns the connector into a streaming connector. Third, the input and output data objects of each crowd task are set again via suitable parameters. The resulting model is almost identical to Figure 4, except for the missing notation for the streaming connector and the data objects that are referenced via parameters.

One of the key configurations of the crowd tasks is their binding with their task templates in CrowdFlower. This requires creating three task templates for the process and setting the `crowdflower_task_id` for each crowd task. The templates are created to accept as input and to provide as output the correct data expected by the process execution. The screen shots in Figure 8 show an excerpt of the three templates instantiated with concrete receipts. For all the tree templates we set the reward to 10 dollar cents, which is high for this type of micro-task, so as to attract more workers and have results in a short time.

Given the process model and the implemented task templates, it is possible to transform the process into its executable form. Specifically, Figure 9 shows the transformed model of Figure 4. In line with the transformation logic described previously, the three crowd tasks are transformed into three parallel branches containing the sub-processes managing the instances of the micro-tasks completed in CrowdFlower. The topmost branch corresponds to the original `Transcribe receipt` task and feeds the other two branches with events that

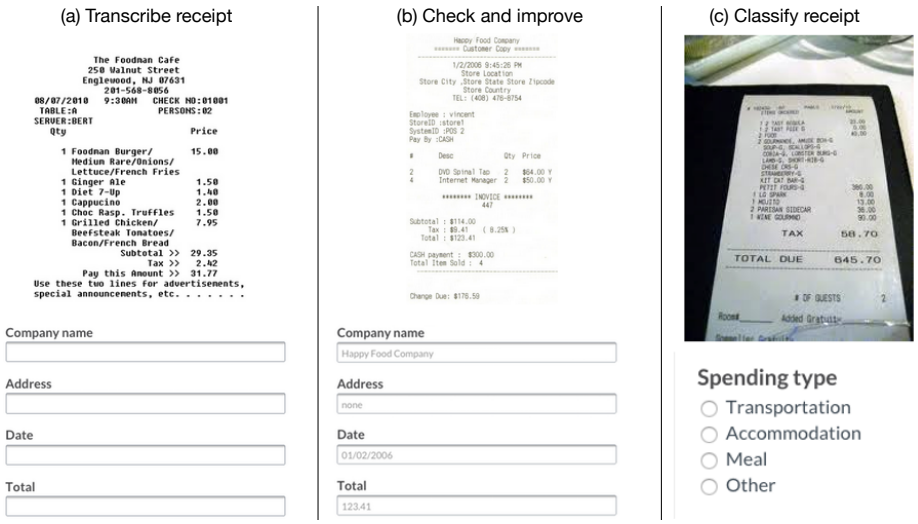

Fig. 8. Screen shots of the three micro-task pages as rendered in CrowdFlower

group two and four micro-task completions, respectively. The numbers of instances are also computed, and the process is ready for execution.

5.2 Performance evaluation

To evaluate the performances of this implementation, we performed a set of experiments in which we focused on the crowd tasks only (the process without the `Upload receipts` and `Notify user` tasks). We uploaded the 40 receipts manually, and ran the process 6 times: 3 times without streaming and 3 times with streaming (as illustrated in Figure 1). We ran the two settings on two different days (Thursday and Friday, 19/20 March, 2015) in three different batches at 12:00, 16:00, and 20:00 CET, and stopped micro-tasks after max 1.5 hours of execution. Independently of streaming or not, our experience has shown that a same micro-task can be executed within very different times (from minutes to hours). In order to prevent overlapping batches, we applied the cut-off time and manually completed outstanding instances. The cost of each execution was of approximately 8 USD, with a reward of 0.10 USD per micro-task instance.

The runtime behavior of the process executions is illustrated in Figure 10, which plots the histogram of micro-task instances performed per time unit (2:25 minutes, for best readability) for each crowd task. In all tasks, the majority of micro-task instances is completed during the productivity peak immediately after deployment, and almost all instances are completed within one hour. However, in some runs the last one or two instances took several hours to complete, and the figure applies the cutoff of 1.5 hours (for presentation purposes). We did not compare the quality of outputs between the streaming and non-streaming conditions, which is out of the scope of this work (speed).

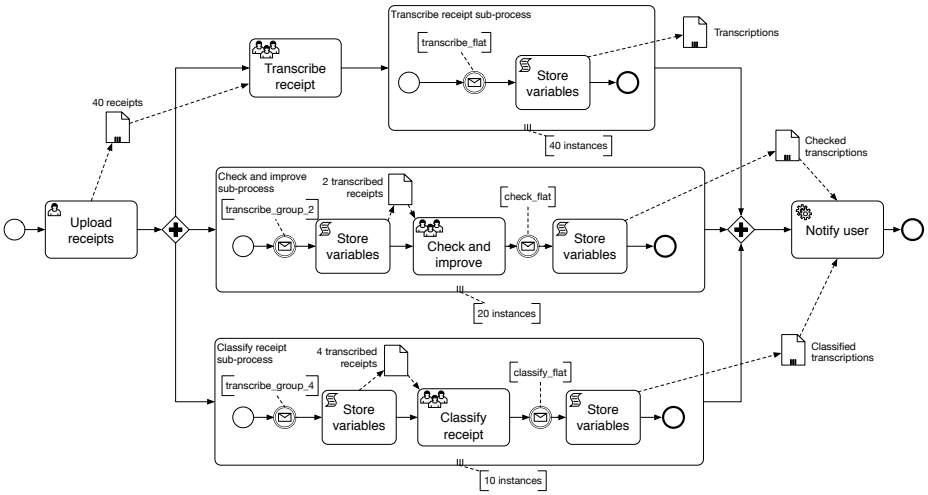

Fig. 9. Transformed BPMN process model with the micro-task instance streaming logic resolved into a set of crowd tasks for micro-task deployment and event handlers and multi-instance sub-processes for instance management and data transformation.

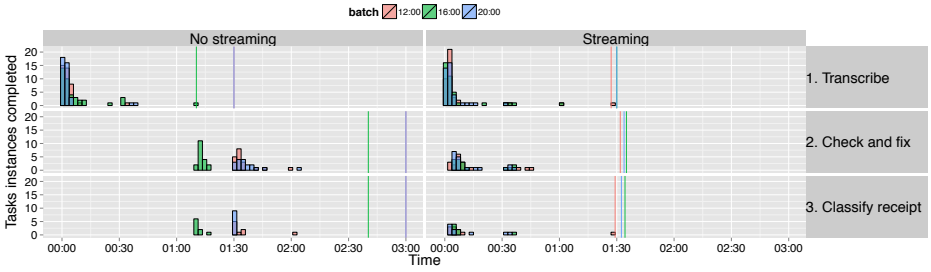

Fig. 10. Performance analysis of no streaming vs. streaming. The colored vertical lines indicate the end of the respective micro-task execution (the deployed batches).

The charts clearly show the benefit of streaming: approximately 90% of all receipts passed through all three crowd tasks within the first 10 minutes of execution, while in the no streaming condition the same amount of receipts passed only through the transcription task. With streaming, it takes only a couple of minutes to transcribe, check and fix, and classify the first receipt; without streaming it takes about one and a half hours. The two peaks in the no streaming condition happen in correspondence of the respective terminations of the prior transcription task. Note that without streaming almost all micro-task instances are processed very fast, while with streaming the last few instances are more dispersed in time. This is in line with the findings by Chilton et al. [5] that workers tend to select micro-tasks (i) that appear on the first two pages in the crowdsourcing platform and (ii) that have a high number of instances.

The last few instances of a micro-task, especially in the streaming setting where micro-tasks are deployed at the rate of individual instances, the last condition is not met. With hundreds or thousands of instances, the benefits of streaming however clearly outweigh this shortcoming.

6 Related work

The need for support for crowdsourcing processes is acknowledged by the recent emergence of a set of advanced crowdsourcing approaches: Turkit [13] and AutoMan [3] propose dedicated programming languages (a JavaScript-like language and Scala, respectively) that allow one to programmatically deploy micro-tasks on Mechanical Turk and to pass data among them. AutoMan, in particular, allows the crowdsourcer to define confidence levels for the quality of results and automatically manages the scheduling and pricing of micro-tasks as well as the acceptance and rejection of results. Jabberwocky [1] is a MapReduce-based human computation framework that consists of (i) a human and machine resource layer (Dormouse), (ii) a parallel programming framework (ManReduce), and (iii) a high-level scripting language for micro-task definition (Dog). CrowdDB [7] is an SQL-extension that allows one to embed crowd interrogations into SQL queries. Based on schemas and annotations of tables in a database, it transforms queries into workflows of crowd tasks for Mechanical Turk, generates appropriate user interfaces, and manages data integration. AskSheet [16] is a Google Spreadsheet extension with functions that allow the spreadsheet to leverage on crowdsourced work. For instance, data enrichment micro-tasks deployed on Mechanical Turk can be used to check prices or products in given grocery stores. Turkomatic [12] delegates not only work to the crowd but also work management operations. The crowdsourcer and workers alike can arbitrarily split micro-tasks into subtasks, aggregate subtasks, or perform them. The result is a self-managed workflow executed in Mechanical Turk. CrowdForge [10] is a Django-based crowdsourcing framework for composite tasks similar to Turkomatic that however follows the Partition-Map-Reduce approach. Each step in the resulting process is a crowd task performed on Mechanical Turk. CrowdSearcher [4] is a crowdsourcing system that leverages on reusable design patterns and on tasks performed by machines or people on crowdsourcing platforms or on Facebook.

In the specific context of business process management, CrowdLang [14] is a BPMN-inspired programming language with crowdsourcing-specific constructs. It helps one to design and run composite tasks using tasks performed by both machines and people sourced from various crowdsourcing platforms. Similarly, CrowdWeaver [9] allows the crowdsourcer to visually design workflows of both crowd tasks deployed on CrowdFlower and machine tasks. Finally, in our own prior work we proposed Crowd Computer [17], a BPMN-based design and runtime environment for complex crowdsourcing processes and the design of custom crowdsourcing models (e.g., from micro-tasking to auctions and contests). Composite tasks are expressed graphically as business processes and may make use of human, crowd and machine tasks as well as the full power of BPMN.

None of these approaches, however, supports the streaming of micro-task instances. To the best of our knowledge, only Appel et al. [2] focused on event stream processing in the context of BPMN. The focus of their work is on so-called event stream processing units that represent machine tasks processing real-time data streams. The focus of our work is specifically on the peculiarities of crowd work and the typical data transformations that characterize that domain.

7 Conclusion

The work described in this paper advances the state of the art in business process management with three contributions: an *extension of BPMN* for the modeling of streaming crowdsourcing processes, a BPMN engine with support for *crowd tasks*, and a *streaming middleware* able to overcome the impedance mismatch between the business process engine and the crowdsourcing platform. The analyzed case study demonstrates the convenience of the new modeling constructs and the runtime performance gains that can be achieved.

One of the limitations of the implementation so far is the lack of support for non-blocking event sub-processes, due to the lack of a respective implementation in the BPMN engine we used as starting point. Without being able to dynamically create sub-process instances at runtime, the modeler must guarantee at design time that all data transformations (splitting and grouping) can be mapped to a correct number of respective runtime events, e.g., the process in Figure 4 requires multiples of 4 data items in input. From a modeling point of view, it is currently possible to branch streaming connectors but not to join them again (joins can be implemented using the standard control flow connectors of BPMN). This limitation is due to the fact that this kind of join is no longer a simple join of the control flow but a join of data streams. Joining them asks for joining data items with different multiplicities or group sizes. This asks for logics to deal with redundancy (e.g., averaging outputs) and the correlation of data items. Another limitation that is intrinsic to the approach is that we can control only those aspects of the process execution that are handled by the BPMN engine; we do not have control over the execution semantics of the crowdsourcing platform, e.g., of how micro-tasks are instantiated, managed, canceled, assigned to workers, etc. We can thus not manage exceptions that are internal to the crowdsourcing platform, e.g., micro-tasks that are never completed.

In our future work, we intend to solve these shortcomings and to support the joining of data streams using different join logics, to provide for a model transformation that is fully integrated into the modeling environment, and to integrate support for micro-task instance streaming into our prior work on the Crowd Computer. We intend to conduct additional experiments with hundreds or thousands of micro-tasks to stress-test and fine-tune the runtime environment. In order to attack the high variance of micro-task durations, we want to understand better the reasons for slow durations, so as to dynamically re-deploy problematic micro-tasks and to speed up overall execution times.

The data and streaming middleware of this work are open-sourced on `https://github.com/Crowdcomputer/` and can be adapted to different BPM engines, crowdsourcing platforms, and application domains.

Acknowledgment. This work was partially supported by the project "Evaluation and enhancement of social, economic and emotional wellbeing of older adults" under the agreement no. 14.Z50.310029, Tomsk Polytechnic University.

References

1. S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar. The jabberwocky programming environment for structured social computing. In *UIST'11*, pages 53–64, 2011.
2. S. Appel, S. Frischbier, T. Freudenreich, and A. P. Buchmann. Event Stream Processing Units in Business Processes. In *BPM 2013*, pages 187–202, 2013.
3. D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor. Automan: a platform for integrating human-based and digital computation. *SIGPLAN Not.*, 47(10):639–654, Oct. 2012.
4. A. Bozzon, M. Brambilla, S. Ceri, A. Mauri, and R. Volonterio. Pattern-based specification of crowdsourcing applications. In *ICWE 2014*, pages 218–235, 2014.
5. L. B. Chilton, J. J. Horton, R. C. Miller, and S. Azenkot. Task Search in a Human Computation Market. In *HCOMP 2010*, pages 1–9, 2010.
6. F. Daniel, S. Soi, S. Tranquillini, F. Casati, C. Heng, and L. Yan. Distributed orchestration of user interfaces. *Inf. Syst.*, 37(6):539–556, 2012.
7. M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: Answering Queries with Crowdsourcing. In *SIGMOD 2011*, pages 61–72, 2011.
8. J. Howe. *Crowdsourcing: why the power of the crowd is driving the future of business.* Crown Publishing Group, New York, NY, USA, 1st edition, 2008.
9. A. Kittur, S. Khamkar, P. André, and R. Kraut. Crowdweaver: visually managing complex crowd work. In *CSCW '12*, pages 1033–1036, 2012.
10. A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut. Crowdforge: crowdsourcing complex work. In *UIST'11*, pages 43–52, 2011.
11. P. Kucherbaev, F. Daniel, S. Tranquillini, and M. Marchese. Composite crowdsourcing processes: challenges, approaches, and opportunities. *IEEE Internet Computing*, conditionally accepted: `http://bit.ly/1BtjMTy`, 2015.
12. A. Kulkarni, M. Can, and B. Hartmann. Collaboratively crowdsourcing workflows with Turkomatic. In *CSCW 2012*, pages 1003–1012, 2012.
13. G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. Turkit: human computation algorithms on mechanical turk. In *UIST 2010*, pages 57–66, 2010.
14. P. Minder and A. Bernstein. Crowdlang: a programming language for the systematic exploration of human computation systems. In *SocInfo*, pages 124–137. 2012.
15. O. M. G. (OMG). Business Process Model and Notation (BPMN) version 2.0. `http://www.omg.org/spec/BPMN/2.0`, 2011.
16. A. J. Quinn and B. B. Bederson. AskSheet: efficient human computation for decision making with spreadsheets. In *CSCW 2012*, pages 1456–1466, 2012.
17. S. Tranquillini, F. Daniel, P. Kucherbaev, and F. Casati. Modeling, Enacting and Integrating Custom Crowdsourcing Processes. *ACM Trans. Web*, 9(2), May 2015.
18. W. M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.