Mining and Quality Assessment of Mashup Model Patterns with the Crowd: A Feasibility Study

CARLOS RODRÍGUEZ, University of Trento
FLORIAN DANIEL, Politecnico di Milano
FABIO CASATI, University of Trento

Pattern mining, that is, the automated discovery of patterns from data, is a mathematically complex and computationally demanding problem that is generally not manageable by humans. In this article, we focus on small datasets and study whether it is possible to mine patterns with the help of the crowd by means of a set of controlled experiments on a common crowdsourcing platform. We specifically concentrate on mining model patterns from a dataset of real mashup models taken from Yahoo! Pipes and cover the entire pattern mining process, including pattern identification and quality assessment. The results of our experiments show that a sensible design of crowdsourcing tasks indeed may enable the crowd to identify patterns from small datasets (40 models). The results, however, also show that the design of tasks for the assessment of the quality of patterns to decide which patterns to retain for further processing and use is much harder (our experiments fail to elicit assessments from the crowd that are similar to those by an expert). The problem is relevant in general to model-driven development (e.g., UML, business processes, scientific workflows), in that reusable model patterns encode valuable modeling and domain knowledge, such as best practices, organizational conventions, or technical choices, that modelers can benefit from when designing their own models.

Categories and Subject Descriptors: D.2.13 [**Software Engineering**]: Reusable Software; H.1.2 [**Information Systems**]: User/Machine Systems; H.2.8 [**Database Management**]: Database Applications

General Terms: Algorithms, Experimentation, Human Factors

Additional Key Words and Phrases: Model patterns, pattern mining, crowdsourcing, mashups

ACM Reference Format:
Carlos Rodríguez, Florian Daniel, and Fabio Casati. 2016. Mining and quality assessment of mashup model patterns with the crowd: A feasibility study. ACM Trans. Internet Technol. 16, 3, Article 17 (June 2016), 27 pages.
DOI: http://dx.doi.org/10.1145/2903138

1. INTRODUCTION

Mashups are composite web applications that are developed by integrating data, application logic, and pieces of user interfaces [Daniel and Matera 2014]. *Mashup tools* are integrated development environments (IDEs) that aim to ease the implementation of mashups. Typically, these tools adopt a *model-driven* development paradigm, where developers express application logic via graphical models that can be interpreted at runtime or transformed into executable code. Yet, developing good mashups is still a

Authors' addresses: C. Rodríguez and F. Casati, Department of Information Engineering and Computer Science, University of Trento, Via Sommarive 9, 38123 Povo (TN), Italy; emails: {crodriguez, casati}@disi.unitn.it; F. Daniel, Politecnico di Milano - DEIB, Via Ponzio 34/5, 20133 Milano, Italy; email: florian.daniel@polimi.it.

nontrivial task. It requires not only fluency in the modeling language but also intimate knowledge of the target domain (e.g., the practices, conventions, and procedures that characterize the domain) and of the respective technologies. These requirements do not apply to mashups only. We find them over and over again in all those contexts that leverage on model-driven formalisms, such as service composition [Alonso et al. 2003], business processes [Weske 2007], scientific workflows [Deelman et al. 2009], UML [OMG 2014], or web engineering with IFML [Object Management Group 2014].

One approach to mitigate this complexity is assisting developers in their task by providing them with automated modeling *recommendations*. In the context of mashups, this approach has produced a variety of techniques: Carlson et al. [2008], for instance, recommend possible next components to be used in response to the modeler using a given component; the approach is based on semantic annotations of component descriptors. Greenshpan et al. [2009] propose an approach that recommends components and connectors (so-called glue patterns) in response to the modeler providing a set of desired components; the approach computes top-k recommendations out of a graph-structured knowledge base containing components and glue patterns (nodes) and their relationships (arcs). Chen et al. [2009] allow the modeler to mash up components by navigating a graph of components and connectors; the graph is generated in response to a query providing descriptive keywords. Riabov et al. [2008] also follow a keyword-based approach to express goals to feed a planner that derives candidate mashups. Elmeleegy et al. [2008] recommend components based on co-occurrence probabilities and semantic matching; upon the selection of a component, an automatic planner derives how to connect the selected component with the partial mashup.

In our own prior work, we contributed to the state of the art with an extension of Yahoo! Pipes (http://pipes.yahoo.com) that interactively recommends and weaves complex *mashup model patterns* while modeling a "pipe" (a data mashup). Recommended patterns were mined from a repository of freely accessible pipes models [Rodríguez et al. 2014b]; the specific dataset used consisted of 997 pipes taken from the "most popular pipes" category, assuming that popular pipes are more likely to be functioning and useful. Before their use, patterns were checked by an expert to ensure their meaningfulness and reusability. The extension is called Baya, and our user studies demonstrate that recommending model patterns has the potential to significantly lower development times in model-driven mashup environments [Roy Chowdhury et al. 2014].

The approach, however, suffers from problems that are common to pattern mining algorithms in general: identifying good support threshold values, managing large numbers of produced patterns, coping with noise (useless patterns), giving meaning to patterns, and dealing with the cold start problem (mining patterns from empty or very small datasets that still need to grow) is hard. Inspired by the recent research on *crowdsourcing* [Howe 2008], the intuition emerged that it might be possible to attack these problems with the help of the crowd, that is, by involving humans in the mining process. The intuition is backed by the observation that pure statistical support does not always imply interestingness [Geng and Hamilton 2006], and that humans are anyway the ultimate ones responsible for deciding about the suitability of discovered patterns.

In Rodríguez et al. [2014a], we studied one approach to mine mashup model patterns for Yahoo! Pipes with the help of the crowd (the *Naïve* approach presented in this article), compared it with our automated mining algorithm described in Rodríguez et al. [2014b], and discussed its applicability to business process models. The study in this article builds on these prior works and advances them along three directions:

—The design and implementation of two new crowd-based *pattern mining* approaches that aim to understand if the crowd is able to spot repeated patterns in a dataset (support) and their comparison with the *Naïve* and automated approaches

—The design, implementation, and comparison of two crowd-based pattern *quality assessment* approaches and one expert-based quality assessment approach
—An analysis of the *work performance* of the crowd and of its ability to self-curate a pattern knowledge base as required by recommendation tools.

Next, we introduce the background of this study and state our research questions. Then, we describe the overall design of the study and the adopted methodology. In Sections 4 and 5, we describe the mining and quality assessment approaches and refine the research questions into more concrete subquestions, study them, and discuss our findings. In Section 6, we discuss the limitations and threats to the validity of the work and then close the article with related works and our final considerations.

2. BACKGROUND AND RESEARCH QUESTIONS

2.1. Reference Mashup Models: Data Mashups

Data mashups are a special type of mashups that specifically focus on the integration and processing of data sources available on the web. Typical data sources are RSS and Atom feeds, plain XML and JSON resources, or more complex SOAP and RESTful web services. *Data mashup tools* are IDEs for data mashup development. They provide a set of data processing operators (e.g., filters or split-and-join operators) and the possibility to configure data sources and operators (we collectively call them components).

In this article, we specifically focus on the data mashup tool Yahoo! Pipes and our pattern recommender Baya [Roy Chowdhury et al. 2014]. The components and mashups supported by these tools can be modeled as follows: let CL be a library of *components* of the form $c = \langle name, IP, IF, OP, emb\rangle$, where *name* identifies the component (e.g., RSS feed or Filter), IP is the set of input ports for data-flow connectors, IF is the set of input fields for the configuration of the component, OP is the set of output ports, and $emb \in \{yes, no\}$ tells whether the component allows for the embedding of other components or not (e.g., to model loops). We distinguish three classes of components: *Source* components fetch data from the web or collect user inputs at runtime. They don't have input ports: $IP = \emptyset$. *Data processing* components consume data in input and produce processed data in output: $IP, OP \neq \emptyset$. A *sink* component (the Pipe Output component) indicates the end of the data processing logic and publishes the output of the mashup (e.g., using JSON). The sink has neither input fields nor output ports: $IF, OP = \emptyset$.

A *data mashup* (a pipe) can thus be modeled as $m = \langle name, C, E, DF, VA\rangle$, where *name* uniquely identifies the mashup, C is the set of integrated components, $E \subseteq C \times C$ represents component embeddings, $DF \subseteq (\cup_i OP_i) \times (\cup_j IP_j)$ is the set of data-flow connectors propagating data from output to input ports, and $VA \subseteq (\cup_k IF_k) \times STRING$ assigns character string values to input fields. Generic strings are interpreted as constants, and strings starting with "item." map input data attributes to input fields. A pipe is considered *correct* if it (1) contains at least one source component, (2) contains a set of data processing components (the set may be empty), (3) contains exactly one sink component, (4) is connected (in the sense of graph connectivity), and (5) has value assignments for each mandatory input field.

A *mashup model pattern* (see Figure 4 for an example) can thus be seen as a tuple $mp = \langle name, desc, tag, C, E, DF, VA\rangle$, with *name*, *desc*, and *tag* naming, describing, and tagging the pattern, respectively, and C, E, DF, VA being as defined earlier, however with relaxed correctness criteria: a pattern is *correct* if it (1) contains at least two components, (2) is connected, and (3) has value assignments for each mandatory input field.

For a better understanding of the type of patterns we are looking for, Figure 1 shows the screen shot of a correct pattern and links its elements to the conceptual model. In

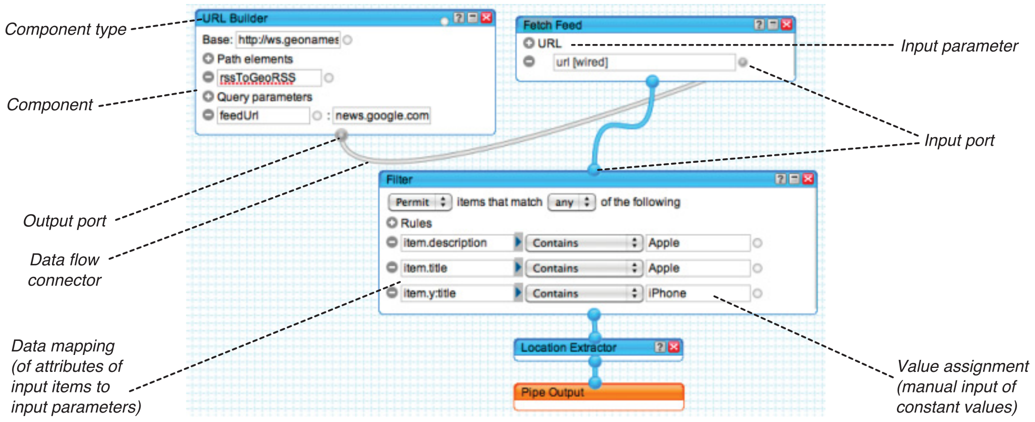

Component type
Component
Output port
Data flow connector
Data mapping (of attributes of input items to input parameters)
Input parameter
Input port
Value assignment (manual input of constant values)
URL Builder
Base: http://ws.geonames
Path elements
rssToGeoRSS
Query parameters
feedUrl : news.google.com
Fetch Feed
URL
url [wired]
Filter
Permit items that match any of the following
Rules
item.description Contains Apple
item.title Contains Apple
item.y:title Contains iPhone
Location Extractor
Pipe Output

Fig. 1. A simple Yahoo! Pipe that plots news on a map [Roy Chowdhury et al. 2014].

essence, the pattern shows how to enrich an RSS feed with geo-coordinates and plot the enriched feed on a Yahoo! map. If we analyze the figure, we easily acknowledge that implementing the respective logic requires good knowledge of Yahoo! Pipes: the URL Builder component requires setting configuration parameters. Components need to be connected in order to allow data to flow; that is, the outputs of the components must be mapped to inputs of other components. More importantly, plotting news onto a map implies knowing that this requires enriching an RSS feed with geo-coordinates, fetching the actual feed, and only then plotting the items on a map. This logic is neither trivial nor intuitive without profound prior knowledge—and the example is still simple.

2.2. Crowdsourcing

Crowdsourcing (CS) is the outsourcing of a unit of work to a crowd of people via an open call for contributions [Howe 2008]. A *worker* is a member of the crowd (a human) that performs work, and a *crowdsourcer* is the organization, company, or individual that crowdsources work. The crowdsourced work comes in the form of a *crowd task* (i.e., a unit of work that requires human intelligence and that a machine cannot solve in useful time or not solve at all). Examples of crowd tasks are annotating images with tags, translating text from one language into another, or designing a logo.

A *crowdsourcing platform* is an online software infrastructure that provides access to a crowd of workers and can be used by crowdsourcers to crowdsource work. Multiple CS platforms exist, which all implement a specific *CS model*: the *marketplace* model caters to crowd tasks with fixed rewards for workers and clear acceptance criteria by the crowdsourcer. The model particularly suits micro-tasks like annotating images and is, for example, adopted by Amazon Mechanical Turk (https://www.mturk.com) and CrowdFlower (http://crowdflower.com). The *contest* model caters to tasks with fixed rewards but unclear acceptance criteria; workers compete with their solutions for the reward, and the crowdsourcer decides who wins. The model suits creative tasks like designing a logo and is, for example, adopted by 99designs (http://99designs.com). The *auction* model caters to tasks with rewards to be negotiated but clear acceptance criteria. The model suits creative tasks like programming software and is, for example, adopted by Freelancer (http://www.freelancer.com).

For the purpose of this article, we specifically leverage on micro-tasks in marketplace CS platforms. Crowdsourcing a task in this context involves the following steps: The crowdsourcer publishes a description of the task to be performed, which the crowd can inspect and possibly express interest for. In this step, the crowdsourcer also defines the reward workers will get for performing the task and how many answers (task instances) should be collected (instantiated) per task. 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 called preselection, 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 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 quality. The crowdsourcer rewards only work that passes the check.

2.3. Research Questions

The final *goal* of this work is to understand whether it is possible to delegate to the crowd the identification of patterns and the curation of a pattern knowledge base in situations with “small” datasets (dozens rather than thousands or millions). These are typical cold-start situations in which automatic algorithms perform poorly.

The *assumptions* underlying this research are that (1) the patterns obtained with the help of the crowd can be as interesting as the ones obtained by automated algorithms, (2) we have access to a repository of mashup models of limited size (in our experiments we specifically study the case of 40 models), (3) the identification of patterns can be crowdsourced as micro-tasks via maketplace-based CS platforms (we study three task designs with different levels of visibility into the dataset: one, three, and 10 models), and (4) the assessment of patterns can be crowdsourced as micro-tasks (we study two task designs that allow workers to rate individual patterns or to pairwise compare them).

Accordingly, the work aims to answer the following *research questions*:

RESEARCH QUESTION 1 (PATTERN MINING). *Is the crowd able to discover meaningful, reusable mashup model patterns from a dataset of 40 models using micro-task designs that provide different levels of visibility of the dataset (one / three / 10 models)?*

RESEARCH QUESTION 2 (QUALITY ASSESSMENT). *Is it possible to crowdsource the quality assessment of identified patterns using task designs that allow the crowd to rate individual patterns or to pairwise compare them?*

It is important to note that we use the term “mining” with its generic meaning of “discovering knowledge,” which does not necessarily imply machine learning. In particular, the intuition of this work is that individual workers are able to identify reusable fragments (patterns) inside mashup models, where the identified patterns are not based on statistical recurrence but rather on human inspection and reasoning. Also, note that the second research question subsumes the availability of suitable mashup model patterns, for instance, identified by the crowd, mashup developers, or experts.

3. STUDY DESIGN

3.1. Methodology

The study is designed to address the two research questions and is inspired by the work presented in Stolee and Elbaum [2010]. It is divided into two parts. In the first part, we answer Research Question 1 by crowdsourcing the identification of patterns. The key design decisions of this part regard the datasets used, the task design, the selection of workers, and the acceptance criteria for patterns. The second part of the study is devoted to the assessment of patterns submitted by the crowd. The key design decisions are related, again, to the design of the tasks, the set of criteria used to assess the patterns, and the assessment of workers. The detailed design of the experiment is outlined in Figure 2 that illustrates the conceptual process model of the research methodology using BPMN4Crowd, a BPMN [OMG 2011] extension for crowdsourcing we are developing in another project [Tranquillini et al. 2014].

The study starts with downloading a set of mashup models from Yahoo! Pipes. The collected dataset consists of 997 pipes randomly selected from the "most popular" pipes category of Yahoo! Pipes' online repository. We opted for this category because it is very common to find pipes in Yahoo! Pipes that are just half elaborated or simply do not work. By focusing our attention on this category, pipes are most likely to be functional and useful. Pipes models are represented in JSON. The "small dataset" was constructed by randomly selecting 40 pipes out of the 997 (making sure we kept only pipes that were runnable and meaningful). To feed the crowd-based mining experiments, we also collected the screen shots of each of the 40 pipes through the Yahoo! Pipes editor.

Once the datasets are constructed, we run two pattern identification experiments based on (1) a crowd-based approach with three different designs for the pattern identification task and (2) an automated mining approach run over two datasets of different sizes (with 40 and 997 pipes) (Section 4). The crowd-based approach aims to study the performance of the crowd in the function of different levels of dataset visibility (to understand if the crowd also looks for support); the automated approach studies the performance of the machine in the function of different dataset sizes and minimum support levels. The SplitWith(n) operator splits the input dataset into collections of one, three, and 10 pipes; each collection is processed by a corresponding task design of the crowd-based approach (as explained in Section 4.1). The three crowd tasks are followed by a Filter operator that drops patterns that have fewer than two components, are not connected, and do not have all input fields filled with meaningful information. This last part of the Filter operator is done manually by an expert, for example, to filter out meaningless text like random key inputs such as "asdf" and "qwerty." Then, all patterns are used to compute a set of metrics (number of patterns, average pattern size, cost per pattern) that, in turn, are used to answer Research Question 1.

Starting from the set of patterns produced by the best crowd approach, the process proceeds with the study of whether the crowd can be used to validate patterns (Section 5). We compare three approaches: expert assessment (us) versus crowd assessment of individual patterns versus crowd assessment of pairs of patterns. Patterns are split into two equal-sized subsets for the experts, subsets of size 1 for the first crowd assessment, and pairs of patterns for the second crowd assessment. The process models also the experts' task as a crowd task, in that they too can be seen as a crowd (a group of two). The study ends with a comparison of the results and the answer to Research Question 2.

3.2. Crowdsourcing Approach

Figure 3 illustrates the approach we followed to crowdsource the crowd mining and assessment tasks of the study using the crowdsourcing platform CrowdFlower (http://www.crowdflower.com). The deployment of tasks on CrowdFlower requires the configuration of suitable forms to collect data from the crowd, the uploading of the dataset that contains the units of work (i.e., the mashup models and patterns), and the preparation of the qualification tests for workers, among other tasks that are specific to CrowdFlower. Once the tasks are deployed, CrowdFlower posts them to third-party platforms, such as Amazon Mechanical Turk or MinuteWorkers, where the crowd can actually perform the requested work.

Each *pattern identification task* points to an external pattern selector page where the crowd can select patterns from the mashups in the dataset. Each mashup model is

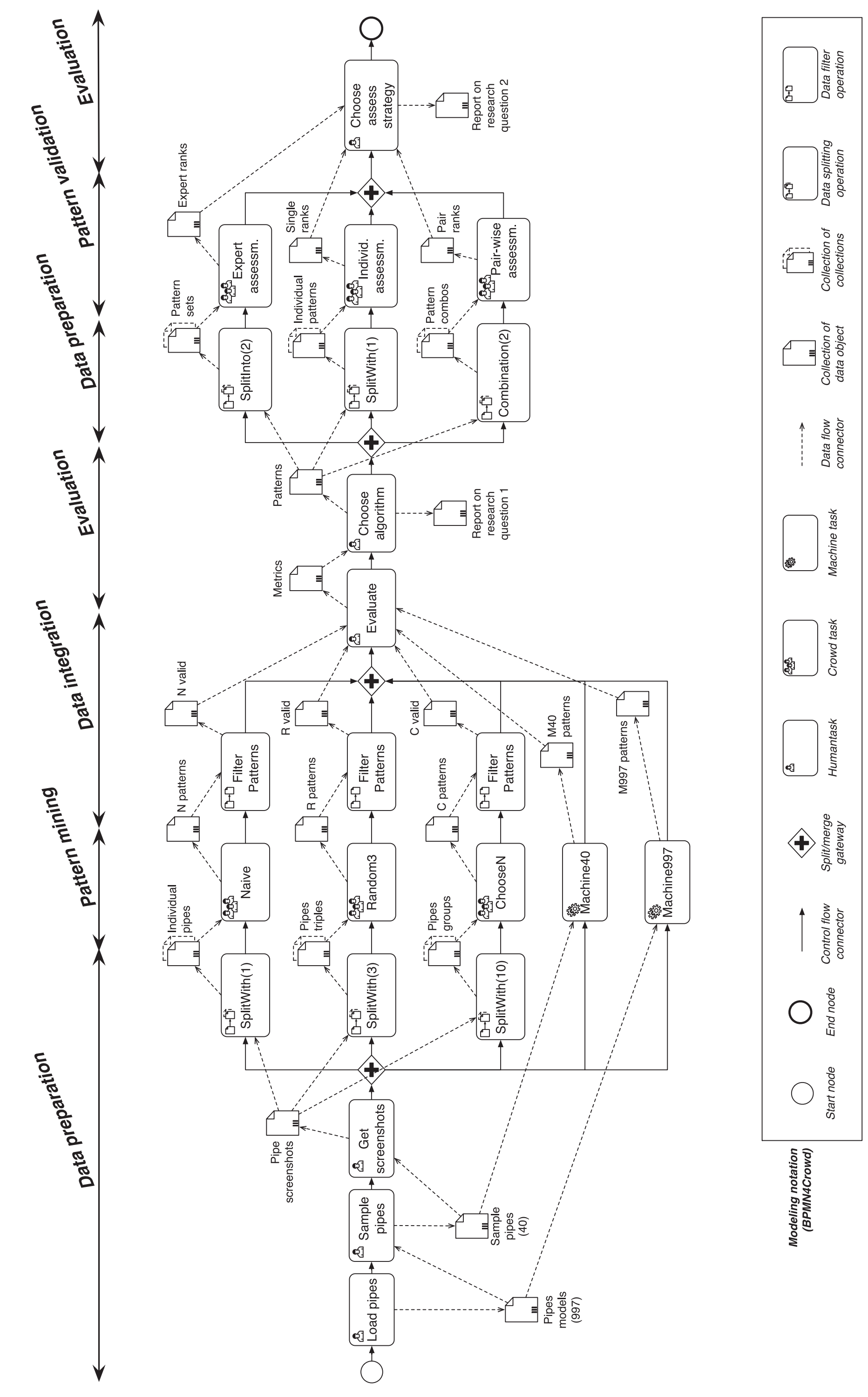


Fig. 2. Process model of the methodology underlying the study described in this article expressed in a BPMN extension for crowdsourcing processes (BPMN4Crowd). The process comprises crowd, human, and machine tasks for workers, researchers, and web services, respectively, and describes how the different experiments are conducted and integrated. On top of the process, we identify the high-level phases of the study.

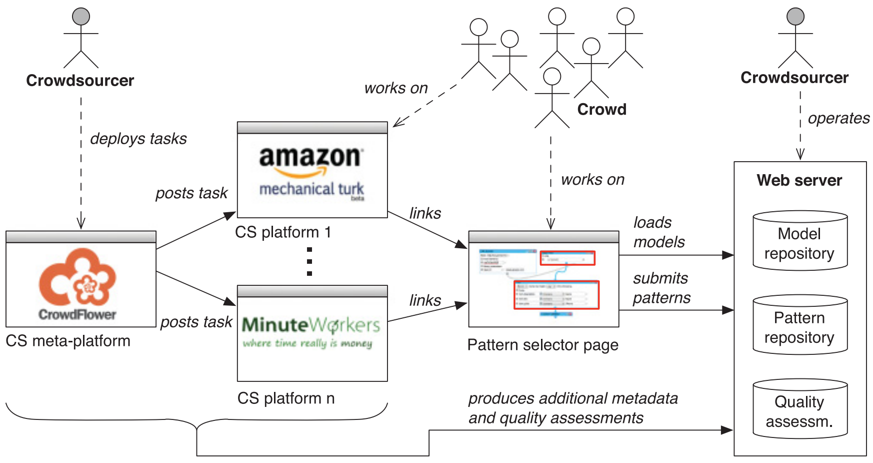


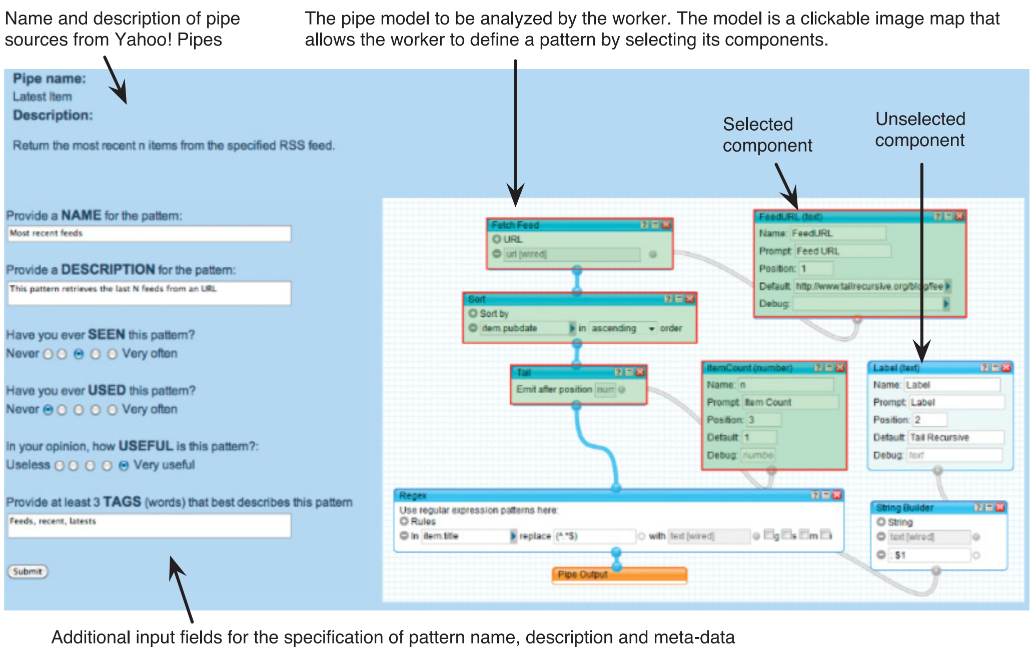
Fig. 3. Approach to crowd-based pattern mining with CrowdFower: patterns are collected via a dedicated Pattern Selector application and assessed using standard forms as provided by CrowdFlower.

configured to be shown to at least three workers, in order to guarantee that each model gets properly inspected, and a monetary reward is set for each pattern provided by the crowd. For each of the three crowd approaches, the pattern selector provides a different implementation (discussed in the next section). The pattern selector page consists of a standard web application implemented in Java, HTML, CSS, and JavaScript that renders the screen shot(s) of the pipe(s) to be analyzed. The worker can also provide a name, a description and a list of tags to equip the pattern with additional metadata. Inputs provided by the workers are validated, for example, to check that a worker indeed selects a pattern in the mashup model. The web application is hosted on a web server operated by us. The web server hosts a model repository where the mashup models are stored and from where the pattern selector page gets the models. It also hosts a pattern repository where the patterns selected by the crowd are stored for further analysis.

For the *quality assessment* tasks, instead, CrowdFlower provides enough support to implement suitable evaluation questionnaires and to crowdsource the tasks without the need for any external application. Collected quality assessments are stored for further analysis in a dedicated quality assessments repository on our web server.

4. MINING MODEL PATTERNS

To answer Research Question 1 (Section 2.3), we implemented three different crowd task designs and one automated mining algorithm. The three designs are an attempt to compare the performance of the crowd by varying the number of pipes per task, the key property that distinguishes the crowd approaches from the automated one; they do not yet represent an in-depth study of how to identify the best design. The idea of presenting workers with different levels of insight into the available dataset stems from the background of this work (i.e., pattern mining), which is commonly based on the concept of statistical support (repetition). The question the three designs aim to answer, hence, is not only whether the crowd is able to identify patterns but also if it is able to spot repetitions and how much insight into the dataset is beneficial, if at all. The automated algorithm is run with different support levels and dataset sizes and the results are compared to the ones obtained by the crowd-based approach.



Pipe name:
Latest Item
Description:

Return the most recent n items from the specified RSS feed.

Provide a **NAME** for the pattern:
Most recent feeds

Provide a **DESCRIPTION** for the pattern:
This pattern retrieves the last N feeds from an URL

Have you ever **SEEN** this pattern?
Never ○ ○ [✓ ●] ○ ○ Very often

Have you ever **USED** this pattern?
Never [✓ ●] ○ ○ ○ ○ Very often

In your opinion, how **USEFUL** is this pattern?:
Useless ○ ○ ○ ○ [✓ ●] Very useful

Provide at least 3 **TAGS** (words) that best describes this pattern
Feeds, recent, latests

Fig. 4. Task design for the identification of mashup model patterns (*Naïve* setting).

4.1. Mining Tasks and Algorithms

A core decision when crowdsourcing a task is how to design the UI used to interact with workers. In general, all crowdsourcing platforms available today allow the crowdsourcer to design form-based user interfaces directly inside the crowdsourcing platform. For the crowdsourcing of simple tasks, such as the annotation of images or the translation of a piece of text, this is sufficient to collect useful feedback. In more complex crowdsourcing tasks, such as our problem of identifying patterns inside mashup models, textual, form-based UIs are not enough and a dedicated, purposefully designed graphical UI (the pattern selector page) is needed.

In order to make workers feel comfortable with the selection of patterns inside pipes models, we wanted the representation of the pipes to be as close as possible to what real pipes look like. In other words, we did not want to create an abstract or simplified representation of pipes models (e.g., a graph or textual description) and, instead, wanted to keep the full expressive power of the original representation. We therefore decided to work with screen shots of real pipes models, on top of which we allow workers to select components of the model and to construct patterns by simply clicking on the respective components. Figure 4 shows a screen shot of the UI we implemented for selecting patterns inside a pipes model; the UI shows a pipe with a total of nine components, of which five have been selected by the worker to form a pattern (the green-shaded components). In this case, the selected pattern performs a useful task (in the context of Feed processing) that consists of fetching (unsorted) feeds from the web and taking only the N (three in this case) most recent feeds. The worker only selects components; the derived pattern contains the components along with the respective data-flow connectors, value assignments, and data mappings (this to keep the pattern identification simple). Next to selecting a pattern, the worker is also asked to provide additional information about the pattern, such as a name, a description, and at least three tags. The worker may further tell how often he or she has already seen or used the pattern before, and how useful he or she thinks the pattern is.

In order to make sure that workers know Yahoo! Pipes, we include a set of five multiple-choice preselection questions in each of the task designs, such as "Which of the following components can be embedded into a loop?" or "What is the maximum number of Pipe Output modules permitted in each pipe?" (See Appendix A for a screen shot of the questionnaire used.) In order for a worker to be paid, he or she must correctly answer these questions, for which we already know the answers (so-called *gold data*).

4.1.1. Naïve Task Design. This task design presents the worker with one pipe and asks him or her to identify a fragment of the pipe that consists of components that jointly perform a useful task and that can be reused to solve a recurrent problem. The task design is exactly that in Figure 4, which provides the maximum freedom in selecting the components that make up a pattern.

4.1.2. Random3 Task Design. This task design randomly selects three pipes from the available dataset of 40 pipes and shows them to the worker; the first pipe allows the worker to identify patterns as described in Figure 4. Figure 13 in Appendix A illustrates the complete task UI for this setting. This design provides the worker with access to more data, which may help him or her identify repetitive patterns. Again, the worker is asked to provide additional metadata via the form.

4.1.3. ChooseN Task Design. This task design randomly selects 10 pipes out of the dataset of 40 pipes and allows the worker to select again $n \in \{1..10\}$ pipes for the identification of patterns. First, the worker selects the pipes of interest, and then he or she is presented with a page for pattern identification similar to the one adopted for the *Random3* setting. Figure 14 in Appendix A illustrates the UI that allows the worker to choose the pipes of interest. This task design provides the worker with most information for the identification of patterns.

4.1.4. Machine Algorithm. Automatically mining mashup patterns of the form $mp = \langle C, E, DF, VA \rangle$ requires identifying repetitive fragments that include the components C, possible component embeddings E, the data flows DF among the components, and the values VA for their input parameters. Doing so is nontrivial, because we need to mine data at different levels of granularity and make sure that the resulting patterns are indeed meaningful. We do this by means of Algorithm 1 (*Machine*), which mines what in Rodríguez et al. [2014b] we called "multicomponent patterns." The algorithm first identifies recurrent components and data flows (lines 1–5), then it identifies the most recurrent value assignments and data mappings for the identified components (lines 6–18), and finally assembles everything into connected patterns (lines 19–22). Data mappings of input data attributes to input fields are dealt with by the parameter values that have a prefix "item." for the input fields that accept dynamic input data. The core mining tasks are based on a standard frequent subgraph mining algorithm (line 5) and a standard frequent itemset mining algorithm (line 18) [Tan et al. 2005].

4.2. Experiment Design

In order to be able to provide clear insights into the ability of the crowd to mine mashup model patterns in the presence of small datasets, we articulate Research Question 1 into the following subquestions:

—*Feasibility*: does crowdsourcing the tasks $Naive$, *Random*3, or $ChooseN$ allow one to mine reusable mashup model patterns from mashup models?

—*Value*: do model patterns identified through $Naive$, *Random*3, or $ChooseN$ contain more domain knowledge than automatically mined patterns?

ALGORITHM 1: Machine

```
Data: repository of mashup models $M$, minimum support ($minsupp_{cdf}$) for components
      and data flows, minimum support ($minsupp_{par}$) for parameter values
Result: set of patterns $\langle C, E, DF, VA \rangle$.
1  $DB_g$ = array();          // create database of graph representations of mashups
2  foreach $m \in M$ do
3      $g$ = getGraphRepresentation($m$);
4      append($DB_g, g$);
5  $FG$ = mineFrequentSubraphs($DB_g, minsupp_{cdf}$);          // mine frequent subgraphs
6  $DB_{mc}$ = array();     // retrieve all instances of each subgraph from dataset M
7  foreach $m \in M$ do
8      foreach $fg \in FG$ do
9          if getGraphRepresentation(m) contains $fg$ then
10             $mf$ = getSubgraphInstance($m, fg$);
11             append($DB_{mc}[fg], mf$);
12 $Patterns$ = set();                            // create database of mashup patterns
13 foreach $MC \in DB_{mc}$ do
14     $DBVA$ = array();  // mine frequent parameter values (including data mappings)
15     foreach $mc \in MC$ do
16         foreach $c \in mc.C$ do
17             append($DBVA, c.VA$);
18     $FI_{va}$ = mineFrequentItemsets($DBVA, minsupp_{par}$);
19     foreach $VA \in FI_{va}$ do
20         foreach $mc \in MC$ do
21             if $VA \in mc$ then
22                 $Patterns = Patterns \cup \{\langle mc.C, mc.E, mc.DF, VA \rangle\}$;   // assemble patterns
23 return $Patterns$;                             // return database of mashup patterns
```

—*Cost effectiveness*: is pattern identification with *Naive*, *Random*3, or *ChooseN* more cost-effective than with a dedicated domain expert?

The subquestions interpret crowd-based pattern mining as a solution that brings together characteristics of both automated algorithms and domain experts and thus compares them with the quality of automated algorithms, on the one hand, and the cost of involving a domain expert, on the other hand.

4.2.1. Experiment Design and Dataset. The three crowd approaches are implemented as outlined earlier using the CS platform CrowdFlower. Running them is a joint manual and automated effort: the pattern selector application takes care of initializing the dataset (the pipes models), partitioning it, and mapping partitions to tasks at runtime. The actual tasks are deployed manually on CrowdFlower and executed by the crowd. Filtering out valid patterns is again done manually. For each pipe, we request at least three judgments, estimated a maximum of 300 sec per task, and rewarded USD 0.10 per task. The *Machine* algorithm is implemented in Java. The core parameter used to fine-tune the algorithm is the minimum support that the mined subgraphs must satisfy ($minsupp_{cdf}$); we therefore use this variable to test and report on different test settings.

The *dataset* available consists of 997 pipes (with 11.1 components and 11.0 connectors on average) randomly selected from the "most popular" pipes category of Yahoo! Pipes' online repository. The JSON representation of the pipes is used by the automatic mining algorithm and to validate inputs in the task UIs; the screen shots are used to collect patterns from the crowd as explained earlier. We run the *Machine* algorithm with datasets of 997 (big dataset) and 40 pipes (small dataset). We use $Machine^{997}$ and $Machine^{40}$ to refer to the former and the latter setting, respectively. We run $Naive$, *Random*3, and $ChooseN$ only with 40 pipes (small dataset).

4.2.2. Evaluation Metrics. While for automated mining it is clear by design what the output of an algorithm looks like, this is not as clear if the identification of patterns is delegated to the crowd. As described earlier, workers may not clearly understand the goals of a task or cheat and, hence, not provide meaningful data. To filter out those patterns that we can instead reasonably trust, we define a set of minimum criteria for crowd-mined patterns: a *valid* mashup pattern is a correct pattern that consists of at least two modules and where the modules are connected, the name and description of the pattern are not empty, and the description and the pattern structure match semantically. The first three criteria we enforce automatically in the pattern identification UIs of the three crowd tasks. Whether the description and pattern structure match semantically (i.e., whether the description really tells what the pattern does) is assessed manually by experts (us). The result of this analysis is a Boolean: either a pattern is considered valid (and it passes the filter) or it is considered bad (and it fails the filter). Note that with “valid” we do not yet assert anything about the actual value of a pattern; this can only be assessed with modelers using the pattern in practice. The same expert-based filter is usually also applied to the outputs of automated mining algorithms and does not introduce any additional subjective bias compared to automated mining scenarios. These prefiltering criteria are intimately related to the context we are dealing with (i.e., Yahoo! Pipes); other contexts may require different criteria.

In order to compare the performance of the five test settings, we use three metrics to compare the pattern sets they produce in output: the *number of patterns found* gives an indication of the effectiveness of the algorithms in finding patterns; the *average pattern size*, computed as the average number of components of the patterns in the respective output sets, serves as an indicator of how complex and informative identified patterns are; and the *distribution of pattern sizes* shows how diverse the identified patterns are in terms of complexity and information load. The *cost per pattern* of the different approaches allows us then to reason on the cost efficiency.

We use the size of patterns/pipes (number of components) as a proxy to measure complexity. This is an approximation of the true complexity of model patterns. In general, complexity is multifaceted and may consist of different aspects, such as McCabe’s cyclomatic complexity [McCabe 1976] for generic code that counts the number of possible independent paths through the code (indeed, model patterns can be seen as fragments of code). Given the context of this work (i.e., recommending model patterns inside modeling environments), the size of patterns is, however, a good approximation of how pattern complexity is perceived by users inside the modeling environment.

4.3. Results

Figure 5 summarizes the task instances created and the patterns collected by running the three crowd tasks. The crowd started a total of 326 task instances of $Naive$, while it submitted only 174 patterns through our pattern selector application. This means that a total of 152 task instances were abandoned without completion. Out of the 174 patterns submitted, only 42 patterns satisfied our criteria for valid mashup patterns; the 42 valid patterns were identified by eight different workers. Running *Random*3 and $ChooseN$ produced a similar number of task instances each (320 and 334), while the number of submitted patterns significantly dropped (17 and 14), as did the number

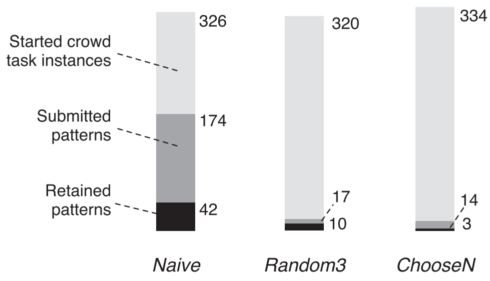
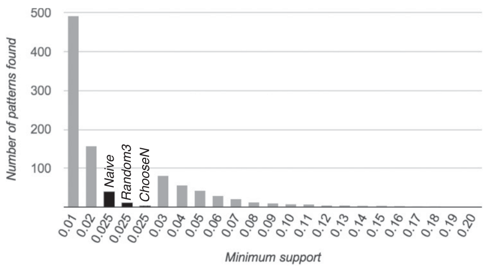
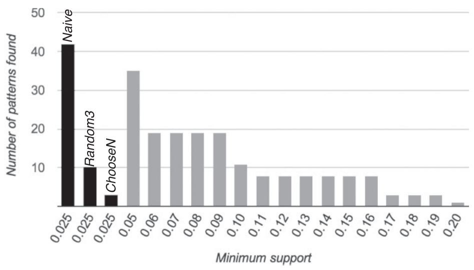


Fig. 5. Crowd task instances started and patterns.

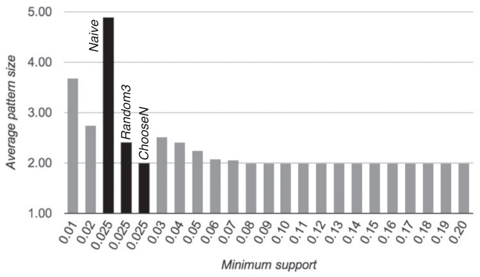


(a) *Machine*^{997}(gray) compared to the crowd (black)

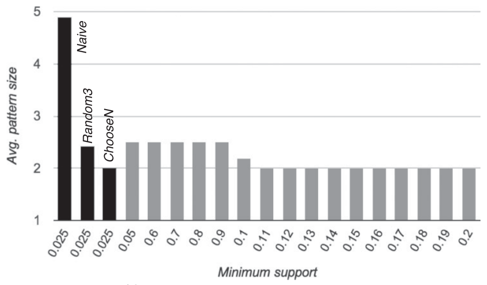


(b) *Machine*^{40} (gray) compared to the crowd (black)

Fig. 6. Numbers of patterns produced by automated mining under varying minimum support levels. The charts also report the number of patterns produced by the three crowd-based mining algorithms (in black).



(a) *Machine*^{997}(gray) compared to the crowd (black)



(b) *Machine*^{40} (gray) compared to the crowd (black)

Fig. 7. Average size of the patterns produced by automated mining under varying minimum support levels. The average sizes of the patterns produced by the crowd-based mining algorithms are reported in black.

of valid patterns retained (10 and 3). The difference between submitted and retained patterns confirms the viability of the valid pattern criteria.

For *Naive* (which shows the best results), we checked whether there is a correlation between the complexity of a pipe and the number of patterns submitted. The Pearson's correlation coefficient computed for all *submitted* patterns is $r_S = -0.1422$, while for all *retained* patterns it is $r_R = 0.0750$. These values are quite low and we cannot conclude that there is a significant association between the complexity of pipes and the number of patterns submitted or retained, nor could we find any threshold for the complexity of pipes above/below which the performance of the crowd drops.

The charts in Figures 6 through 8 report on the numbers of patterns, average pattern sizes, and distribution of pattern sizes obtained by running *Machine*^{997} and *Machine*^{40} with different minimum relative support levels sup_{min}. The bars in gray are the results of the *Machine* algorithm; the black bars represent the results of the crowd approaches.

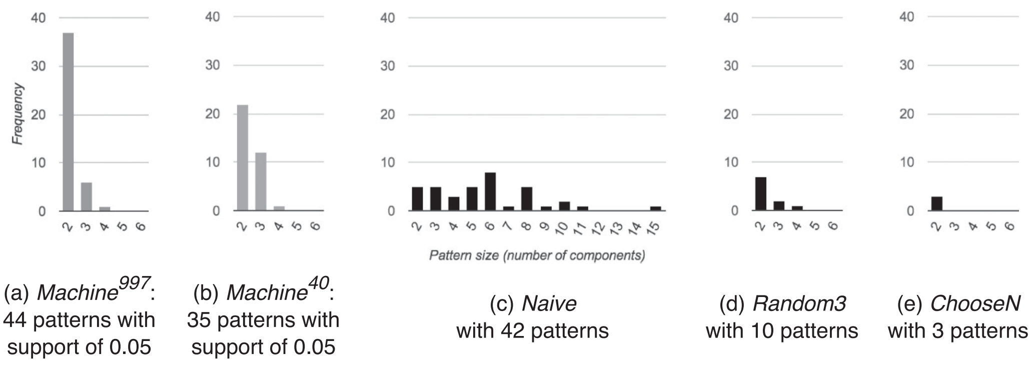


Fig. 8. Pattern size distribution by the five algorithms. The histograms of $Machine^{997}$ and $Machine^{40}$ refer to the run with the minimum support level that produced a number of patterns similar to $Naive$.

For comparison, we placed these latter at a support level of $sup_{min} = 0.025$, which corresponds to $1/40 = 0.025$, in that we ask workers to identify patterns from a single pipe without the need for any additional support (even if more than one pipe is shown).

4.3.1. Feasibility. Figure 6(a) illustrates the number of patterns found by $Machine^{997}$. The number quickly increases for $Machine^{997}$ as we go from high support values to low values, reaching almost 500 patterns with $sup_{min} = 0.01$. Figure 6(b) shows the results obtained with $Machine^{40}$. The lowest support value for $Machine^{40}$ is $sup_{min} = 0.05$, which corresponds to an absolute support of two in the dataset. It is important to note that only very low support values produce a useful number of patterns. In both figures, the black bar represents the 42, 10, and three patterns respectively identified by $Naive$, *Random*3, and $ChooseN$.

The two figures show the typical problem of automated pattern mining algorithms: only a few patterns for high support levels (which are needed, as support is the only criterion expressing significance), too low support levels required to produce useful output sizes with small datasets (our goal), and an explosion of the output size with large datasets. Figure 5 shows that $Naive$ is instead able to produce a number of patterns in output that is similar to the size of the dataset in input; *Random*3 and $ChooseN$ do not perform as well. Notice also that, while Figure 6 reports on all the patterns found by $Machine$, the data for the crowd algorithms include only valid patterns. This means that not only is $Naive$ able to find a good number of patterns but also it is able to find practically meaningful patterns. Understanding the actual value of the patterns would require its own study in which the users of the patterns are involved. We discuss next the value of the resulting patterns from our perspective and that of our experimental setting.

4.3.2. Value. Figure 7 shows the average pattern sizes of $Machine^{997}$ and $Machine^{40}$ compared to that of the crowd approaches. In both settings, the average pattern size obtained with $Naive$ clearly exceeds the one that can be achieved with $Machine$, even for very low support values (0.01); *Random*3 and $ChooseN$ perform similarly to $Machine$. With Figure 8, we look more specifically into how these patterns look like by comparing those runs of $Machine^{997}$ and $Machine^{40}$ with the crowd approaches that produce a similar number of patterns in output as $Naive$. In both settings, this happens for $sup_{min} = 0.05$ and produced 44 and 35 patterns, respectively. Figures 8(a) and 8(b) show that automatically mined patterns are generally small (sizes range from two to four), with a strong prevalence of the most simple and naïve patterns (size two). Figure 8(c), instead, shows that the results obtained with $Naive$ present a much higher diversity in the pattern sizes, with a more homogeneous distribution and even very complex patterns of sizes that go up to 11 and 15 components. *Random3* and *ChooseN* (Figures 8(d) and 8(e)) again do not perform better than *Machine*. *Naive* is thus able to collect patterns that contain more complex logics and that are more informative; that is, they provide richer examples of how to use components and how to combine them together. This can be attributed to the higher freedom in selecting components when working with *Naive* and to the fact that the crowd tends to work on a least-effort basis (it is harder to come up with elaborated patterns when working with *Random3* and *ChooseN*). These patterns also come with a characterizing name, description, and list of tags. These annotations not only enrich the value of a pattern with semantics but also augment the domain knowledge encoded in the pattern and its reusability. Patterns identified with *Naive* thus contain more domain knowledge than the patterns mined automatically and the ones mined with *Random3* and *ChooseN*; these latter instead produce patterns of similar size to the automatically mined patterns, with *ChooseN* performing the worst among all studied approaches.

4.3.3. Cost Effectiveness. The aforementioned results for *Naive* show that crowd-based pattern mining can outperform machine-based mining for small datasets in terms of productivity (more specifically, the ratios of number of patterns found per number of pipes in input are $35/40 = 0.86$ and $42/40 = 1.05$ for $Machine^{40}$ and *Naive*, respectively). The alternative to automated mining would be asking an expert to identify patterns, which is expensive. Here, crowd-based mining also outperforms the expert. As outlined in Figure 9, with a cost per pattern of USD \$ 0.42 and a running time of approximately 6 hours, *Naive* proves to be a very competitive alternative to hiring a domain expert: we paid only USD \$ 2.83 per hour, a price that is hard to beat compared to hiring a domain expert. Given the low number of patterns identified by *Random3* and *ChooseN*, their cost per pattern is significantly higher (USD \$1.76 and USD \$5.58), which makes them less suitable also from an economical point of view.

4.4. Discussion

The aforementioned results manifest a high *sensitivity* of the crowd mining algorithms to the design of the crowd tasks. In this respect, we distinguish between intuitiveness (including ease of use) and complexity of tasks. Regarding the *intuitiveness*, we considered collecting patterns via textual input (e.g., the list of component names in a pattern) or via abstract data- flow graphs (automatically constructed from the JSON representation of pipes). After a set of informal, pre-experiment tests of the crowd task designs to adopt, we opted for the screen shots. This has proven to be the representation workers seem to be most familiar with (screen shots do not introduce any additional abstraction), and this is the approach we implemented in the three crowd tasks. The identification of the design to adopt was a best-effort task not aimed at identifying the best possible design, which we consider future work. As for the *complexity* of the tasks, the *Naive*, *Random3*, and *ChooseN* algorithms provide the worker with access to one, three, and 10 pipes, respectively, that is, with different information loads. The three algorithms produced a comparable number of task instances, while they strongly differ in the number of patterns submitted and retained. The three alternative designs allowed us to understand whether more visibility into the available dataset would allow the crowd to spot repeated patterns, or whether pattern identification by the crowd is mostly based on semantic/functional considerations. The results we obtained from our experiments confirm that the side effect of such expanded visibility inevitably leads to more complexity, which in turn leads to high abandon rates (see Figure 5). We interpret this as evidence that high information loads only scare people away (instead of helping them) in the context of pattern identification. The lesson learned is thus to keep tasks

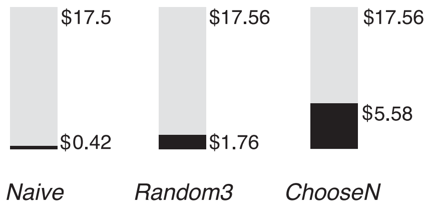


Fig. 9. Total cost of crowdsourcing experiments (gray) and cost per pattern (black).

as simple as possible, that is, to apply the KISS (Keep It Simple, Stupid) principle. The result, although in line with similar findings in the area of crowdsourcing [Mason and Watts 2010], is particularly important in the area of pattern mining that instead typically requires the analysis of large datasets to produce viable outputs.

In order to ensure that workers had the necessary mashup knowledge, we performed a selection using gold data. Yet, our questions were too tough in our first tests, and we had to lower our expectations. What happened with the tough questions was that it was hard to process the whole dataset and at the same time meet our *valid pattern* criteria. Lowering the toughness of the questions allowed us to process the whole dataset and to obtain more patterns, not all of them of good quality, however, as reported in previous sections. We also noticed a *natural selection* phenomenon: the majority of patterns was submitted by only a few workers. We assume these were workers with good knowledge in Pipes that simply liked this kind of modeling task and, therefore, submitted multiple patterns not only for the sake of the reward but also for personal satisfaction.

Despite the selection of workers, it is strongly advised to check all inputs for *formal validity* (e.g., no empty strings); otherwise, workers may just skip inputs or input fake content (e.g., a white space). Our task designs ensure that only correct patterns can be submitted, yet the difference between submitted and retained patterns (the semantic check) illustrated in Figure 5 highlights the importance of advanced input validation.

Regarding the *robustness* of these results against varying reward amounts, we did not notice any reduction in the number of tasks instantiated by the crowd or the number of patterns collected if we lowered the reward from USD $0.10 down to USD $0.05 (in an independent, pilot experiment). Our feeling is that we could have gotten the same results also for less money without any loss of quality, however, demonstrating this would require additional, purposefully designed experiments.

We also inspected manually the *diversity* of collected patterns and noticed a preference for complex combinations of components over more naive ones, which results in a high diversity of patterns. At a more fine-grained detail, however, it is possible to identify overlapping parts, especially regarding the use of popular pairs of components (e.g., it is common to see the association of the Fetch Feed and the URL Builder components). The inspection of the structure and content of the patterns reveals a preference for self-contained configurations, able to provide useful functions on their own.

The *fairness* of the comparison with an algorithm based on support as a measure of interestingness may be debatable, as such is expected to work poorly on small datasets. However, this poor performance establishes the triggering condition to look for an alternative solution, such as crowd-based mining and, thus, represents a baseline. In addition, the studied scenario is typical in model-driven development paradigms: it is very rare to find repositories with millions of models where data-mining-based approaches can take full advantage (e.g., SAP's reference process model repository, one of the biggest publicly available, contains only a total of 604 models [Dijkman et al. 2011]). Perhaps dedicated rule-based systems, instance-based learning, case-based reasoning, or other automated approaches could allow a fairer comparison, yet we were not able to identify any readily available, suitable algorithms and therefore relied on our previous automated algorithm to understand whether the crowd-based approach may represent a valid alternative. In any case, automated approaches in the current state of the literature would still hardly be able to beat humans in providing rich descriptions for patterns and, what is even harder, useful prescriptions on how to use them.

5. ASSESSING MODEL PATTERNS

To answer Research Question 2, we focus on the patterns obtained by *Naive*. We implemented one expert quality assessment experiment (that serves as the ground truth) and two crowd quality assessment experiments to validate the patterns.

5.1. Assessment Tasks

The crowd tasks for this experiment consist of questionnaires created within CrowdFlower that allow us to visualize the patterns and to collect ratings using multiple-choice questions. We discuss next the two approaches used for pattern assessment.

5.1.1. Individual Expert/Crowd Assessment. The first task design is shared by both the *Expert* assessment and the *Individual* crowd assessment and aims to test if the crowd interprets individual patterns similarly to an expert. Each task shows one pattern (screen shot, name, description, list of tags) and asks for the assessment of its understandability, reusability, usefulness, and novelty (we discuss these later in this section) using a Likert scale (from 1-negative to 5-positive). We also include the preselection questionnaire already used for the mining tasks, in order to make sure that workers are knowledgeable in Yahoo! Pipes. Figure 15 in Appendix B shows the task design.

5.1.2. Pairwise Crowd Assessment. This task design implements the *PairWise* crowd experiment that aims to study whether it is possible to obtain an overall pattern ranking similar to the one by the experts. It shows a *pair* of patterns and asks workers to identify the one with the highest understandability, reusability, usefulness, and novelty. Patterns are thus not assessed individually, but in relation to other patterns, which has proven to help workers make better decisions [Jung and Lease 2011]. To compute a (partial) ranking, each pattern is given six chances to be voted. The task again includes the preselection questionnaire used in the previous tasks. See the task UI design in Figure 16 in Appendix B.

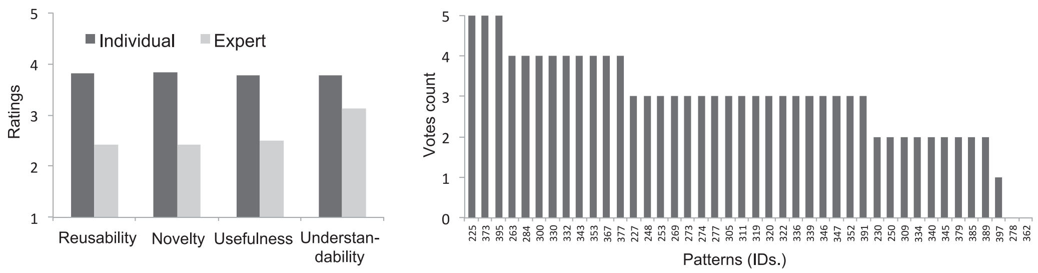
5.2. Experiment Design

Answering the second research question requires again identifying a set of subquestions that can be studied in detail. The two questions we want to study in the following to understand whether the crowd is able to assess the quality of mashup patterns are as follows:

—*Replaceability*: does crowd-based quality assessment with *Individual* and *PairWise* produce more similar results than an expert-based assessment?
—*Reliability*: in crowd assessments collected with *Individual* and *PairWise*, is the bias introduced by misunderstandings, cheaters, or malevolent workers negligible?

In order to answer these questions, we compare the *Expert*, *Individual*, and *PairWise* assessment approaches using the following data and metrics.

5.2.1. Experiment Design and Dataset. The *Expert* assessment is done locally on our own machine; the *Individual* and *PairWise* crowd assessments are again implemented on CrowdFlower. For both crowd tasks we estimated a maximum duration of 300 sec per task and payed a reward of USD $0.02 per task. The *Individual* setting asks for exactly



(a) *Individual* vs. *Expert* rating (avg of Likert ratings) (b) *Understandability* pattern ranking in decreasing order of aggregated votes

Fig. 10. Quality assessment results: (a) *Individual*/*Expert* ratings, (b) *PairWise* understandability ranking.

three judgments per pattern, a requirement that is configured in CrowdFlower. The *PairWise* setting makes sure that each pattern appears exactly 6 times in different pair combinations. All tasks use as an input dataset the 42 patterns obtained by *Naive*.

5.2.2. Metrics and Statistical Tests. We use four criteria to assess the quality of patterns. The *understandability* of a pattern refers to how comprehensible the pattern is; a pattern is understandable if one can easily figure out what the pattern does and how to use it. The *usefulness* of a pattern refers to the utility of the pattern, that is, to how handy and convenient it is to use the pattern for solving a given problem. The *reusability* of a pattern is the extent to which the pattern can be used in different contexts. Finally, the *novelty* of a pattern refers to how new and innovative the pattern is. Appendix C discusses examples of good and bad patterns for each of these criteria.

Replaceability: To compare the *Expert* and *Individual* assessments, we use Mann-Whitney's U test (nonpaired) to test whether they produce comparable ratings. Both experiments produce ordinal data (Likert scale) that generally do not meet the normality condition. We further compute Spearman's correlation coefficient (for paired ordinal data) for *Individual* and *PairWise* against *Expert*. Finally, we order all patterns in decreasing order for each of the three approaches and quality criteria individually and check the *precision* ($P = \frac{TruePos}{TruePos+FalsePos}$) and *recall* ($R = \frac{TruePos}{TruePos+FalseNeg}$) of the top-ranked patterns by *Individual* and *PairWise* compared to *Expert* (the ground truth). We specifically compute P and R for the 25th, 50th, and 75th percentiles to test different quality assessment policies (hard vs. soft).

Reliability: We use Fiedman's analysis of variance to test whether the *Individual* ratings provided for each criterion are comparable among each other. The use of this test is again motivated by the use of the Likert scale for ratings and the nonnormality of the distribution of the dataset; in addition, since we test a set of criteria that refer to the same set of patterns and patterns are assessed by the same set of workers, we cannot assume independence in this test. Spearman's correlation coefficient provides further insight into the strength and direction of the associations for each criterion. For the *PairWise* assessment, we compare whether there is a bias in the preferences expressed by the crowd toward either the first or the second pattern shown in the task. The samples (rankings of first patterns vs. rankings of second patterns) are dependent, are expressed with ordinal data, and follow a nonsymmetrical distribution. We thus use the Wilcoxon signed-rank test for this decision. In all statistical tests we use a significance level of $\alpha = 0.05$.

5.3. Results

Figures 10 and 11 present the aggregate assessments by *Expert* and *Individual*, one of the pattern rankings obtained by *PairWise*, and the respective precision/recall charts.

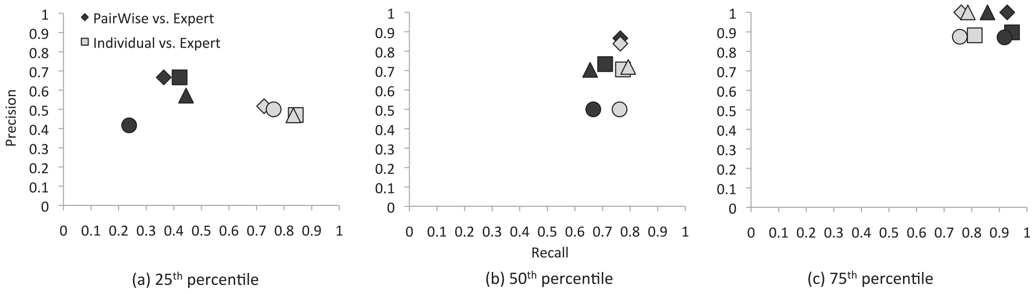


Fig. 11. Precision and recall of the *Individual* and *PairWise* assessment experiments with varying selectivity (top 25th, 50th, and 75th percentiles) for understandability (□), usefulness (◇), reusability(△), and novelty (○).

Table I. Comparison of Ratings Between *Expert* and *Individual* ($H_0 : \eta_{ind} = \eta_{exp}$), and Spearman's Correlation Coefficient ρ for *Expert* Versus *Individual* and *Expert* Versus *PairWise*

		Spearman's Correlation Coefficients ρ	
Criteria	**Mann-Whitney's test**	***Expert* vs. *Individual***	***Expert* vs. *PairWise***
Reusability	$p = 5.787 \times 10^{-9}; U = 8029$	−0.0783	0.1581
Novelty	$p = 3.287 \times 10^{-13}; U = 8741$	0.1212	−0.1017
Usefulness	$p = 6.392 \times 10^{-10}; U = 8197$	0.0257	0.1755
Understandability	$p = 5.744 \times 10^{-4}; U = 6870$	0.0732	0.1403

Table II. In-Group, Cross-Criteria Spearman's Correlations for the *Expert*, *Individual*, and *PairWise* Quality Assessments

	Spearman's Correl. Coefficients ρ		
Criteria	***Expert***	***Individual***	***PairWise***
Reusability vs. Novelty	−0.5452	0.8249	0.8862
Reusability vs. Usefulness	0.5682	0.8520	0.9053
Reusability vs. Understandability	0.6389	0.8319	0.9026
Novelty vs. Usefulness	−0.2545	0.7971	0.9503
Novelty vs. Understandability	−0.3356	0.8075	0.9485
Usefulness vs. Understandability	0.7978	0.8832	0.9969

We use these and Table I to study replaceability. Tables II and III help us study reliability.

5.3.1. Replaceability. Two observations can immediately be made from Figure 10(a): (1) there seems to be an important difference between the average ratings produced by *Expert* and *Individual*, and (2) *Individual* seems to provide similar ratings for all criteria. Here, we consider only (1); we leave (2) for the discussion of the reliability subquestion. We use Mann-Whitney's U test (nonpaired) with the null hypothesis $H_0 : \eta_{ind} = \eta_{exp}$ (the medians of the two experiments are equal) to test whether experts and the crowd produce comparable results. The alternative hypothesis is that the medians are not equal, that is, $H_A : \eta_{ind} \neq \eta_{exp}$. The results of the test reported in Table I make us reject H_0 for all four criteria and conclude that the ratings produced by *Individual* and *Expert* are indeed significantly different. A further inspection of the ratings using Spearman's correlation (see the third column in Table I) shows that also the pairwise correlation (per same pattern) of the ratings by *Individual* and *Expert* is very low.

The data produced by the *PairWise* experiment are not directly comparable to the one by the experts (votes vs. Likert scale rating). We can, however, rank patterns using the sum of the votes they obtained. Figure 10(b), for instance, plots the pattern ranking for the understandability criterion (we compute similar rankings for all four criteria).

Table III. Test of Preference Bias Between Patterns Shown *First* Versus Patterns Shown *Second* in the *PairWise* Assessment Task

Criteria	Wilcoxon Signed Rank Test
Reusability	$p = 4.922 \times 10^{-7}; W = 528$
Novelty	$p = 8.844 \times 10^{-8}; W = 666$
Usefulness	$p = 9.910 \times 10^{-8}; W = 666$
Understandability	$p = 1.488 \times 10^{-7}; W = 630$

Now we can compute the pairwise Spearman correlation between *PairWise* and *Expert* (see the last column in Table I). The correlation is very low, and we cannot conclude that the two experiments produce similar assessments for individual patterns.

If we relax our similarity criteria and only look at the selectivity of the three experiments ("good" vs. "bad" inside the ranked lists of patterns), we can compute the precision and recall of the two crowd experiments. Figure 11 shows the results obtained when comparing the top 25th, 50th, and 75th percentiles of the rankings. The results obtained for the 25th percentile report a mediocre precision; *Individual* looks more promising in terms of recall. As we relax the selectivity, the performance increases, with *PairWise* outperforming *Individual* in the 75th percentile. Although these results appear positive, we do not consider them reliable enough (too few data); for example, Figure 10(b) shows that the 50th percentile includes 30 patterns out of 42 and the 75th even 39.

5.3.2. Reliability. Even if the *Individual* experiment produces assessments that are different from those by the experts, we would expect ratings to vary across criteria. Yet, we already pointed out that *Individual* produces roughly the same ratings for all criteria. Using Friedman's analysis of variance and the null hypothesis that the medians of the ratings are instead the same for all criteria, that is, $H_0 : \eta_{und} = \eta_{use} = \eta_{reu} = \eta_{nov}$, we obtain a p-value of $p = 0.4995$ ($df = 3, n = 42, \chi^2 = 2.368$). The high p-value forces us to accept H_0 and that the ratings are the same for the different criteria. This conclusion is reinforced by a further inspection using Spearman's correlation (third column of Table II) that shows high correlations (ρ) across criteria.

The design of the *PairWise* experiment makes sure that each pattern appears exactly 3 times as the first pattern in the task (see Figure 16 in the appendix) and 3 times as the second. We would thus expect that the votes by the crowd are uniformly distributed over the pattern appearing first and second.

We use the Wilcoxon signed rank test to test this assumption, using the null hypothesis that for each criterion the median of aggregated votes is the same for the patterns appearing first and those appearing second, that is, $H_0 : \eta_{1st} = \eta_{2nd}$. The alternative hypothesis is that they are different ($H_A : \eta_{1st} \neq \eta_{2nd}$). The results are reported in Table III. We can see that H_0 must be rejected for all criteria; that is, we are in the presence of a bias. Specifically, the crowd consistently preferred the first pattern over the second. We also compute Spearman's correlation for the *PairWise* experiment and report them in the last column of Table II. The results again show high, positive cross-criteria correlation coefficients, which contrasts the values obtained for *Expert*, not only in strength but also partly in direction.

5.4. Discussion

At first sight, the conclusions of the mining study and those of the quality assessment study seem to contradict each other. But this is not true: the mining tasks impose strict requirements for valid patterns, which eliminate most of the noise (i.e., nonvalid patterns) from feedbacks. This is not possible in the quality assessment study that asks for opinions (not artifacts) for which there are no clear acceptance criteria. In this case, noise (i.e., the assessments produced by nonserious workers) stays in the data and, as our study shows, prevails over the feedback by the more serious workers.

We note that the quality of the elicited patterns as perceived by the experts is not extraordinary. Only the understandability is high compared to the other criteria. The availability of pattern names, descriptions, and tags (which automated algorithms are not able to produce) surely contributed to this result. We ascribe this mostly to the small size of the dataset: we cannot expect lots of revolutionary patterns out of 40 pipes (Appendix C discusses examples of good and bad patterns). Using feedback from only two experts is, however, a limitation of the study, and we would like to involve additional independent experts or users of Baya in our follow-up studies. It is, however, also important to note that the perceived average quality of patterns does not yet mean that patterns are not "good" in practice; eventually, this can only be ascertained by using them for development. Also, we did not assess the quality of the automatically mined patterns and are thus not able to say which approach produces better patterns according to the adopted criteria.

The data collected through our experiments seem to indicate that the crowd works on a least-effort basis. Their performance strongly depends on the task design. For instance, in the *Individual* experiment, workers consistently provide similar ratings for different criteria. Our intuition is that, given the task design in Figure 15, it is simply easier for workers to choose one column and then just click vertically through all criteria. Similarly, in the *PairWise* experiment, we believe that workers actually inspected only the first pattern and essentially neglected the second one, very likely because inspecting two patterns was considered too complex. Also, the use of gold data to assess workers as proposed by CrowdFlower does not solve the problem, and even if there was some notion of acceptable or less acceptable opinions (e.g., a range), checking ranges in Likert scales via gold data questions is not yet supported.

Regarding the alternatives to the crowd-based assessment of mashup patterns, we identify two options: involving an expert (expensive) or asking the users of the patterns to rate them (for free). This latter approach we started exploring in Baya [Roy Chowdhury et al. 2014], where we let users vote on patterns, eliminating the need for an expert (we also plan to allow them to save and share their own patterns). If an expert is involved, the general question may be whether it is better to directly ask him or her to identify high-quality patterns or whether to ask him or her to assess lower-quality patterns by the crowd. An argument in favor of crowd pattern identification is that it likely produces a better variety of patterns than a single expert and that assessing many crowd patterns will not cost more than producing few expert patterns. There is not a single answer to this question and the answer depends on the actual use of the patterns.

6. THREATS TO VALIDITY

This section summarizes the threats that may impose limitations on our findings and their generalization to other scenarios where patterns are mined with the crowd.

6.1. Task Design

The crowdsourcing platform we use for our experiments (CrowdFlower) allows us to perform the *selection of workers* through the use of gold data (to implement questions whose answers are known and formalized a priori so that they can be checked automatically). This means that worker selection and actual task are seamlessly integrated, and workers are judged by the platform while they perform the task. The nature of the task we crowdsource, however, led us to the integration of a preselection questionnaire with the actual task, since it would be very hard to choose a priori patterns from a pipe to use as gold data. This is of course a threat to the worker selection phase because once the worker figures out the right answers for the preselection questionnaire, he or she can use it over and over without the pressure of having to provide good answers to the actual task as when gold data was used.

Our experience also shows that using hard questions can be too selective, and repeatedly trying possible answers will cause the platform to prematurely exclude workers from the task. The identification of mashup model patterns in Pipes is clearly a task that requires specific qualifications from workers that may not be easy to find in crowdsourcing platforms like CrowdFlower. Indeed, the reports provided by the crowdsourcing platform show that only eight workers out of 93 actually managed to pass the control questions for the *Naive* algorithm, while for *Random*3 and *ChooseN* the numbers are 21 of 52 and six of 45, respectively; for the *Individual* assessment experiment, only four workers out of 152 passed our control questions, while for *PairWise* it was four out of 41. One potential threat of this is that the results we obtained may not be reproducible if we do not have the same workers performing our tasks together in the same period of time. The consequence of this is that we may not get the same performance or not get any patterns at all. This issue needs to be explored further in future work, for example, to understand how much the performance varies across different days of the week or times of the day, as well as with different rewards. The challenge thus lies in finding the tradeoff between the difficulty of the questionnaire and the level of expertise of workers, as well as in the postprocessing of the patterns to verify their quality.

The actual *design of our tasks* is based on a best effort. We have internally thought of and analyzed alternative designs based on text-based representation of the models, abstractions based on graphs, and form-based representations, but the alternative that resulted most convincingly is the use of screen shots of real pipes that allow workers to click on individual components to build patterns. Using this representation, we propose our three task designs with different levels of complexity and information load to understand what is acceptable for the crowd. The main issue here (which at the same time was necessary for the comparison) is that in our experiments, we pay the same reward for task designs of different complexities. The poor performance of Random3 and ChooseN may thus be explained partially by the low reward offered to workers for that level of task complexity. Raising the rewards might have produced results comparable to Naive, but it would probably have made these algorithms even less cost-effective than they are now. Our findings therefore assume a same reward policy.

The task design for the *assessment of patterns* asks for opinions, which are collected through the mechanisms supported by the crowdsourcing platform: web forms. The main issue in this case is that opinions using web forms are prone to cheating. The challenges we face here are similar to those of the task design for collecting patterns: it is not possible to create gold data for opinions, because a pattern that is good for one worker may not be as good for another worker. Our findings show that such conditions and the lack of support by the chosen crowdsourcing platform (and similar ones) for more sophisticated mechanisms (e.g., an approach based on the games with a purpose paradigm [Von Ahn 2006]) make the assessment of patterns as proposed in this article produce results that are significantly different from what a dedicated domain expert would produce.

6.2. Experimental Setting

Crowdflower and other state-of-the-art crowdsourcing platforms depend on many configuration parameters including reward, number of answers per task, number of tasks per job, gold data, success rate (on gold data) for the cutoff threshold, target countries, target languages, and the day/time when the tasks are launched, among other more detailed parameters. In addition, Crowdflower also offers the possibility to post tasks to third-party crowdsourcing platforms that, in turn, have their own configuration parameters that are set up internally by Crowdflower and that are obscure to the work provider. This makes the experimental environment and setting complex and represents an important threat to the reproducibility of the experiment (as with all crowdsourcing experiments). In our studies, we make decisions on how to configure these parameters based on a best effort, and we try to control them whenever this is possible. We acknowledge that the configuration used for each single parameter is debatable and each one of these represents a research question that may deserve an investigation on its own.

6.3. Results and Findings

Our studies confirm that crowdsourcing tasks for marketplace platforms should be kept simple and straightforward, in line with what has already been reported in contexts different from ours [Kittur et al. 2008; Mason and Watts 2010]. While at first sight, some findings may thus not be seen as novel contributions, it is important to interpret them properly in their specific context: pattern mining. The purpose of the three different levels of complexity in our studies was, on the one hand, to answer the typical question the data mining expert would come up with (e.g., "what if you showed more models to the worker?") and, on the other hand, to understand which level of complexity is acceptable. So, the question this article aims to answer is not just *if* complexity matters, but *what level* of complexity. And the answer to this question is anything but trivial or obvious. Indeed, the results of our experiments show that there is a noticeable difference in the performance of the crowd. Crowdsourcing is a complex domain, and each task has its own peculiarities and pitfalls; it is in general not possible to apply results from other studies straightaway to tasks and task designs in different contexts, which leads to the need for studies that explore the use of crowdsourcing in different contexts and scenarios.

To the best of our knowledge, this is the first work investigating the specific problem of pattern mining from models with the crowd. As such, it is an exploratory study that unveils where the problems are. And its contribution is exactly this: starting the thread and showing some techniques that work as well as others that do not. Building on this, others (or we ourselves) can work on more targeted, task-specific improvements, just as it happened with many similar task-specific studies in the domain of crowdsourcing (e.g., data labeling and multimedia quality assessment, among other studies).

7. RELATED WORK

In the area of *databases*, the current trend is to exploit crowdsourcing to help answer queries that are hard for computers. For example, Parameswaran and Polyzotis [2011] propose hQuery, a declarative language that allows for the description of query answering tasks that involve human processors, databases, and external algorithms. Franklin et al. [2011] propose CrowdDB, an approach to answer database queries through crowdsourcing. As opposed to the previous work, CrowdDB proposes a modified version of the standard SQL with annotations at the level of both data definition language (DDL) and data manipulation language (DML) enabling the crowdsourcing of database query answering. Marcus et al. [2011] propose Qurk, an on-paper system managing the workflow of querying relational databases, including tasks like filtering, aggregation, sorting, and joining of data sources. The idea is rather simple: it proposes to use standard SQL where the crowd tasks are wrapped into user-defined functions (UDFs) that contain the concrete task description to be sent to the crowd.

Few works focus on the intersection of crowdsourcing with *data mining* and *machine learning*. The current trend is to use the crowd mostly for pattern recognition, classification, and labeling of data items that can be later on used as training examples for machine-learning algorithms. For example, Li et al. [2012] propose CrowdMine, an approach that leverages human computation for solving problems in verification and debugging. This work targets the involvement of nonexperts in solving such problems, and to do so, the authors propose to abstract the problem through a pattern identification game that can be played by any individual of the crowd. In the context of data preprocessing for machine learning, Sheng et al. [2008] propose a crowd-based approach for collecting training labels for data items also from nonexpert human labelers for the use in supervised induction. In particular, the authors consider the repeated labeling for some or all of the examples in the dataset and their impact on the quality level of the training examples. In the same line, McCreadie et al. [2010] propose to leverage crowdsourcing to collect labels for a news dataset. The goal of the work is to produce a reference dataset that can be used for the evaluation and comparison of approaches proposed for news query classification. Von Ahn and Dabbish [2004] propose the ESP game, an interactive system in the form of a game in which players must agree with other players on the labels that can be associated to images in order to proceed in the game. Labels can be used later on for improving image search or as training examples.

A common characteristic of the works presented in this section is that they all target *nonexperts* that are capable of performing relatively simple tasks. They leverage on human-innate capabilities such as image identification, text interpretation, and pattern recognition. The main challenge of these works therefore is the translation of the original task (e.g., image classification or database query answering) to task descriptions and visual metaphors that are in the reach of these types of workers. Our approach is slightly different and more similar to the one proposed by Stolee and Elbaum [2010], in that we target workers that have a minimum knowledge of Yahoo! Pipes. Eliciting from the crowd artifacts as complex as model patterns has not yet been studied before.

The *quality* of mashups and mashup patterns has been studied in the past few years. Cappiello et al. [2012] study the composition of mashups by taking into account different quality dimensions such as the syntactic and semantic compatibility of components, aggregated quality of the mashup, and added value in terms of data, functions, and visualization. These quality dimensions are considered in the context of delivering recommendations for building mashups. Janner et al. [2009] present five different mashup patterns for enterprise mashups that enable cross-organizational collaboration. The authors highlight the requirements imposed by enterprise mashups in terms of security, availability, governance, and quality, which need to be taken into account by mashup patterns in the context of enterprises. Cappiello et al. [2010] address the issue of information quality in mashups. Here, the authors argue that the quality of mashups and mashup patterns strongly depends on the quality of the information that different components can provide, and that other quality aspects such as maintainability, reliability, and scalability play a minor role because mashups are used only for a short time. In our work, we are rather concerned with quality aspects such as the functionality, reliability, and (re)usability of the mashup patterns as investigated by Cappiello et al. [2011] or the composability, openness, and encapsulation as defined by Kohls [2011]. Kohls also discusses the intrinsic subjectivity and the difficulty of agreeing on patterns, two factors all pattern identification approaches have in common.

Whether the crowd can be used to assess the quality of model patterns has not been investigated before, to the best of our knowledge. Several studies on how to use the crowd to assess the quality of other kinds of artifacts, though, exist, with diverging conclusions. Keimel et al. [2012], for instance, study the problem of assessing the quality of videos with the crowd and conclude that the crowd produces similar results to traditional lab tests. Khattak and Salleb-Aouissi [2011] study the performance of the crowd in labeling tasks; they conclude that "injecting a little expertise [expert-provided labels] in the labeling process will significantly improve the accuracy of the labeling task." Instead, Gillick and Liu [2010] demonstrate that "non-expert judges are not able to recover system rankings derived from experts" for more complex text summarization assessments. These results are in line with our finding: the higher the complexity of the task is, the lower the reliability of the crowd assessment. In fact, the first work asks workers only to provide a rank (mouse input), the second to write labels (text input), and the third to read a longer text and two summaries and to rate them (conceptual effort plus mouse input).

8. CONCLUSION

This article studies two research questions: The first question is whether the crowd is able to discover reusable mashup model patterns from a dataset of 40 models using micro-task designs that provide different levels of visibility into the available dataset (one/three/10 models). The different levels of visibility aim to enable the crowd to spot repetitions of patterns in the dataset, similar to how automated algorithms proceed. The finding is that the crowd is indeed able to identify patterns that are meaningful and rich in domain knowledge (*Naive*), but also that more visibility into the dataset is actually counterproductive (*Random*3 and *ChooseN*) and that support is not needed if humans are asked to identify patterns. The second question is whether it is possible to crowdsource the quality assessment of identified patterns. The finding is that with the two task designs we implemented, *Individual* and *PairWise*, we are not able to reproduce assessments that are close to those given by experts. We cannot exclude that other task designs produce better results; our results, however, show that further experiments are needed to study how to simplify the task to make it more accessible, how to incentivize workers, and how to check the skills of workers and the reliability of opinions.

In our future work, we would like to study how to further simplify our tasks, if crowdsourcing can be leveraged also for big datasets, for example, by introducing pattern similarity metrics, or whether the quality of patterns changes if no reward is given at all. A complementary technique to collect and rate patterns could be asking developers themselves to identify and rate patterns, for example, directly inside their modeling environment. We already implemented this idea in our pattern recommender for Yahoo! Pipes, Baya, which allows one to save patterns and to up-/down-vote patterns (https://www.youtube.com/watch?v=AL0i4ONCUmQ). Collecting usage data will allow us to understand if and where real users perform better than the crowd.

ACKNOWLEDGMENT

This research was supported by the project "Evaluation and Enhancement of Social, Economic, and Emotional Wellbeing of Older Adults" under agreement no. 14.Z50.310029, Tomsk Polytechnic University (TPU).

REFERENCES

Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. 2003. *Web Services: Concepts, Architectures, and Applications*. Springer.

Cinzia Cappiello, Florian Daniel, Agnes Koschmider, Maristella Matera, and Matteo Picozzi. 2011. A quality model for mashups. In *The International Conference of Web Engineering (ICWE'11)*. Springer, 137–151.

Cinzia Cappiello, Florian Daniel, Maristella Matera, and Cesare Pautasso. 2010. Information quality in mashups. *IEEE Internet Computing* 14, 4 (2010), 14–22.

Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Florian Daniel, and Adrian Fernandez. 2012. Quality-aware mashup composition: Issues, techniques and tools. In *The International Conference on the Quality of Information and Communications Technology (QUATIC'12)*. IEEE, 10–19.

Michael Pierre Carlson, Anne H. Ngu, Rodion Podorozhny, and Liangzhao Zeng. 2008. Automatic mash up of composite applications. In *The International Conference of Service Oriented Computing (ICSOC'08)*. Springer, 317–330.

Huajun Chen, Bin Lu, Yuan Ni, Guotong Xie, Chunying Zhou, Jinhua Mi, and Zhaohui Wu. 2009. Mashup by surfing a web of data APIs. *VLDB* 2, 2 (August 2009), 1602–1605.

Florian Daniel and Maristella Matera. 2014. *Mashups: Concepts, Models and Architectures*. Springer.

Ewa Deelman, Dennis Gannon, Matthew S. Shields, and Ian Taylor. 2009. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation of Computer Systems* 25, 5 (2009), 528–540.

Remco Dijkman, Marlon Dumas, Boudewijn Van Dongen, Reina Käärik, and Jan Mendling. 2011. Similarity of business process models: Metrics and evaluation. *Information Systems* 36, 2 (2011), 498–516.

Hazem Elmeleegy, Anca Ivan, Rama Akkiraju, and Richard Goodwin. 2008. Mashup advisor: A recommendation tool for mashup development. In *The International Conference on Web Services (ICWS'08)*. IEEE Computer Society, 337–344.

Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. 2011. CrowdDB: Answering queries with crowdsourcing. In *SIGMOD*. 61–72.

Liqiang Geng and Howard J. Hamilton. 2006. Interestingness measures for data mining: A survey. *Computer Surveys* 38, 3 (2006), 9.

Dan Gillick and Yang Liu. 2010. Non-expert evaluation of summarization systems is risky. In *The NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk (NAACL-HLT'10)*. Association for Computational Linguistics, 148–151.

Ohad Greenshpan, Tova Milo, and Neoklis Polyzotis. 2009. Autocompletion for mashups. *VLDB* 2, 1 (August 2009), 538–549.

Jeff Howe. 2008. *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*. Crown Publishing Group, New York, NY.

Till Janner, Robert Siebeck, Christoph Schroth, and Volker Hoyer. 2009. Patterns for enterprise mashups in b2b collaborations to foster lightweight composition and end user development. In *The International Conference on Web Services (ICWS'09)*. 976–983.

Hyun Joon Jung and Matthew Lease. 2011. Improving consensus accuracy via Z-score and weighted voting. In *The AAAI Conference on Human Computation (AAAIWS'11)*. 88–90.

Christian Keimel, Julian Habigt, Clemens Horch, and Klaus Diepold. 2012. Qualitycrowd—A framework for crowd-based quality evaluation. In *Picture Coding Symposium (PCS'12)*. IEEE, 245–248.

Faiza Khan Khattak and Ansaf Salleb-Aouissi. 2011. Quality control of crowd labeling through expert evaluation. In *The Neural Information Processing Systems Workshop on Computational Social Science and the Wisdom of Crowds (NISP'11)*.

Aniket Kittur, Ed H. Chi, and Bongwon Suh. 2008. Crowdsourcing user studies with mechanical turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'08)*. ACM, 453–456.

Christian Kohls. 2011. The structure of patterns: Part II - qualities. In *The Conference on Pattern Languages of Programs (PLoP'11)*. 27:1–27:18.

Wenchao Li, Sanjit A. Seshia, and Somesh Jha. 2012. CrowdMine: Towards crowdsourced human-assisted verification. In *The Annual Design Automation Conference (DAC'12)*. IEEE, 1250–1251.

Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. 2011. Crowdsourced databases: Query processing with people. In *The Conference on Innovative Data Systems Research (CIDR'11)*. 211–214.

Winter Mason and Duncan J. Watts. 2010. Financial incentives and the performance of crowds. *ACM SigKDD Explorations Newsletter* 11, 2 (2010), 100–108.

Thomas J. McCabe. 1976. A complexity measure. *IEEE Transactions on Software Engineering* SE-2, 4 (Dec. 1976), 308–320.

Richard M. C. McCreadie, Craig Macdonald, and Iadh Ounis. 2010. Crowdsourcing a news query classification dataset. In *The SIGIR Workshop on Crowdsourcing for Search Evaluation (CSE'10)*. 31–38.

OMG. 2011. Business Process Model and Notation (BPMN) version 2.0. http://www.bpmn.org. (2011).

Object Management Group (OMG). 2014. *The Interaction Flow Modeling Language (IFML), Version 974 1.0*. OMG standard specification. Object Management Group, http://www.ifml.org.

OMG. 2014. Unified Modeling Language (UML). http://www.uml.org/. (2014).

Aditya Parameswaran and Neoklis Polyzotis. 2011. Answering queries using humans, algorithms and databases. In *The Conference on Innovative Data Systems Research (CIDR'11).* 160–166.

Anton V. Riabov, Eric Boillet, Mark D. Feblowitz, Zhen Liu, and Anand Ranganathan. 2008. Wishful search: Interactive composition of data mashups. In *The International Conference on World Wide Web (WWW'08)*. ACM, 775–784.

Carlos Rodríguez, Florian Daniel, and Fabio Casati. 2014a. Crowd-based mining of reusable process model patterns. In *The International Conference on Business Process Management (BPM'14)*. Springer, 51–66.

Carlos Rodríguez, Soudip Roy Chowdhury, Florian Daniel, Hamid R. Motahari Nezhad, and Fabio Casati. 2014b. Assisted mashup development: On the discovery and recommendation of mashup composition knowledge. In *Web Services Foundations*. Springer, 683–708.

Soudip Roy Chowdhury, Florian Daniel, and Fabio Casati. 2014. Recommendation and weaving of reusable mashup model patterns for assisted development. *ACM Transactions on Internet Technologies* 14, 2–3 (2014), Article 21.

Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. 2008. Get another label? Improving data quality and data mining using multiple, noisy labelers. In *The International Conference on Knowledge Discovery and Data Mining (KDD'08)*. ACM, 614–622.

Kathryn T. Stolee and Sebastian Elbaum. 2010. Exploring the use of crowdsourcing to support empirical studies in software engineering. In *The ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'10)*. ACM, 35.

Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. 2005. *Introduction to Data Mining*. Addison-Wesley.

Stefano Tranquillini, Florian Daniel, Pavel Kucherbaev, and Fabio Casati. 2014. Modeling, enacting and integrating custom crowdsourcing processes. *ACM Transactions on the Web* 9, 2 (2014).

Luis Von Ahn. 2006. Games with a purpose. *Computer* 39, 6 (2006), 92–94.

Luis Von Ahn and Laura Dabbish. 2004. Labeling images with a computer game. In *The SIGCHI Conference on Human Factors in Computing Systems (CHI'04)*. ACM, 319–326.

Mathias Weske. 2007. *Business Process Management: Concepts, Languages, Architectures*. Springer.

Received March 2016; accepted March 2016