Effective Identification and Reuse of Model Patterns in Service Orchestration Modeling

by Florian Daniel, florian.daniel@polimi.it
Goal

Assisting developers by recommending development knowledge
Recommendation and Weaving of Reusable Mashup Model Patterns for Assisted Development

SOUDIP ROY CHOWDHURY, INRIA Saclay
FLORIAN DANIEL and FABIO CASATI, University of Trento

With this article, we give an answer to one of the open problems of mashup development that users may face when operating a model-driven mashup tool, namely the lack of modeling expertise. Although commonly considered simple applications, mashups can also be complex software artifacts depending on the number and types of Web resources (the components) they integrate. Mashup tools have undoubtedly simplified mashup development, yet the problem is still generally nontrivial and requires intimate knowledge of the components provided by the mashup tool, its underlying mashup paradigm, and of how to apply such to the integration of the components. This knowledge is generally neither intuitive nor standardized across different mashup tools and the consequent lack of modeling expertise affects both skilled programmers and end-user programmers alike.

In this article, we show how to effectively assist the users of mashup tools with contextual, interactive recommendations of composition knowledge in the form of reusable mashup model patterns. We design and study three different recommendation algorithms and describe a pattern weaving approach for the one-click reuse of composition knowledge. We report on the implementation of three pattern recommender plugins for different mashup tools and demonstrate via user studies that recommending and weaving contextual mashup model patterns significantly reduces development times in all three cases.
Idea
Development knowledge -> model patterns
Proactively assist development -> recommend patterns
Don’t distract developers -> context + speed
Make knowledge operational -> weave patterns

Research question
Does recommending model patterns really help developers model faster / better?
A typical **mashup model pattern** (Yahoo! Pipes)

Mashup model:  \( m = \langle \text{name}, C, F, M, P \rangle \)
Composition pattern model:  \( cp = \langle C, F, M, P, \text{usage}, \text{date} \rangle \)

The pattern tells how to enrich an RSS feed with geo-coordinates and plot its items on a map
Patterns mined from a dataset of 970 “most popular” pipes models of Yahoo! Pipes (association rule mining, frequent itemset mining + predefined topologies)

Recommendation algorithms

**Contextual:** candidate patterns contain the object of the last modeling action; exact and approximate matching

**Personalized:** ranks contextual recommendations according to users’ past component preferences

**Expert:** ranks contextual recommendations according to experts’ past component preferences; cloning h-index
Modeling **test cases**

100 pipes models, different from the ones used to mine patterns

Generated **856** test cases with different object sizes:

- 356 with object size 1
- 227 with object size 2
- 212 with object size 3
- 61 with object size 4
In-browser **performance** of recommendations

![Graph showing retrieval time vs number of complex patterns in KB for Connector, Par value, Co-occurrence, Complex, and Total categories.](image-url)
Precision and recall:

\[
P = \frac{|TP|}{|TP| + |FP|} \quad \text{and} \quad R = \frac{|TP|}{|TP| + |FN|}
\]

High performance in response to stepwise modeling actions
Retrieve at least 8-9 recommendations
assisted development in Yahoo! Pipes
melette = extension of Apache Rave for UI mashups
8.1. Rationale for server-side mashups

Mashup development depends on the ability to combine loosely related components into a new, hybrid application, whose requirements determine which artifacts (graphical interfaces, data sources or functionalities) are to be combined. Thus, from a high level perspective, hybridization in a mashup can refer to different technological domains, which allows employing the term 'mashup' with at least three different orientations: presentation, data and processes. Different languages, programming methodologies and techniques are applied to each of these areas [Hanson2009]:

- **Presentation-oriented mashups** add various user interfaces to provide a new application or product. This type of mashup usually seeks to create an application that displays all together the various user interface components in a similar way to a...
User studies

**H1:** Baya speeds up mashup development  
**H2:** Development with Baya requires fewer user interactions  
**H3:** Development with Baya requires less thinking time
Crowdsourced user study (Amazon Mechanical Turk): 30 participants equally split into control and test group (developers)

- Development time in seconds
  - Control Group
  - Test Group
  - Mean development time without Baya: $\mu_{dev,ctrl} = 1027.1s$
  - Mean development time with Baya: $\mu_{dev,test} = 384.9s$

- Number of user interactions
  - Control Group
  - Test Group
  - Mean number of interactions without Baya: $\mu_{int,ctrl} = 258.9$
  - Mean number of interactions with Baya: $\mu_{int,test} = 74.3$

- Thinking time
  - Control Group
  - Test Group
  - Mean thinking time without Baya: $\mu_{th,ctrl} = 4.0s$
  - Mean thinking time with Baya: $\mu_{th,test} = 5.5s$

**Retain H1**
(p=0.00045)

**Retain H2**
(p=0.00009)

**Reject H3**
(p=0.00209)

2-sample t-test (Welch): normally distributed samples, unequal variances
Independent user studies by partners in the EU FP7 project **omelette**: Baya for Apache Rave, 44 participants (admins)

(a) user study with Apache Rave (China)

\[ \mu_{dev,ctrl} = 139.8s \]
\[ \mu_{dev,test} = 54.6s \]

(b) user study with Apache Rave (Germany)

\[ \mu_{dev,ctrl} = 132.1s \]
\[ \mu_{dev,test} = 58.9s \]

**Retain H1**
(p=0.0001)

**Retain H1**
(p=0.0007)

2-sample t-test (Welch): normally distributed samples, unequal variances
In conclusion

• Recommending and weaving model patterns can really make modelers **more efficient**!
• Baya is a concrete proof of concept and a flexible starting point for others
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, modelers can benefit from when designing own models.

Research questions

1. Is the crowd able to **discover** meaningful, reusable mashup model patterns?
2. Is it possible to crowdsourcing the **quality assessment** of identified patterns?
Architecture

Web server
- Model repository
- Pattern repository
- Quality assessm.

Crowdsourcer
- deploys tasks
- posts task

CS meta-platform
- posts task

CS platform 1
- links
- works on

Crowd
- works on

Pattern selector page
- loads models
- submits patterns
- produces additional metadata and quality assessments

Crowdsourcer
- operates
Experiment 1: pattern identification

Similar **dataset** as in previous study: 997 pipes models
- 40 randomly picked models for the crowd (*Crowd*)
- 997 for the automated algorithm (*Machine*)

**Evaluation metrics**
- Number of patterns identified
- Avg pattern size (# components)
- Distribution of pattern sizes
- Cost per pattern
Crowd task designs

**Naive**: shows one pipe and asks for a pattern

**Random3**: shows 3 pipes and asks for a pattern

**ChooseN**: shows 10 pipes and asks to choose N pipes and to identify a pattern

+ Automated mining **algorithm** for comparison

---

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 a best effort 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 automated algorithm is run with different support levels and dataset sizes and the results are compared to ones obtained by the crowd-based approach.

4.1. Mining Tasks/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 crowd-sourcer 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 and realistic 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 pipe and to construct patterns by simply clicking on the respective components.

Additional input fields for the specification of pattern name, description and meta-data

Name and description of pipe sources from Yahoo! Pipes

The pipe model to be analyzed by the worker. The model is a clickable image map that allows the worker to define a pattern by selecting its components.
Results
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 of 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 multi-faceted and may comprise 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 8 different workers. Running Random3 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 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.
Number of patterns identified

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

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

$\Rightarrow$ Yes, it is possible to identify patterns with the crowd
Average pattern sizes

(a) Machine$^{997}$ (gray) compared to the crowd (black)
(b) Machine$^{40}$ (gray) compared to the crowd (black)

→ The patterns identified by the crowd are in average bigger
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); Random3 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 \(\text{sup}_{\text{min}} = 0.05\) and produced 44 and 35 patterns, respectively. Figures 8(a) and (b) show that automatically mined patterns are generally small (sizes range from 2–4), with a strong prevalence of the most simple and naive patterns (size 2). 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 Naive. 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 worst among all studied approaches.

---

(a) Machine\(^{997}\): 44 patterns with support of 0.05
(b) Machine\(^{40}\): 35 patterns with support of 0.05
(c) Naive with 42 patterns
(d) Random3 with 10 patterns
(e) ChooseN with 3 patterns

---

The domain knowledge captured by Naive is even complex.
4.3.3. Cost Effectiveness.

The above results for Naive show that crowd-based pattern mining can outperform machine-based mining for small datasets in terms of productivity (more specifically, the ratio of number of patterns found per number of pipes in input are 35/40 = 0.86 and 42/40 = 1.05 for Machine 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. 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 above 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 1, 3 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 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 assure 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 however of good quality, as reported in previous sections. We also noticed a natural selection phenomenon: the majority of patterns was submitted by only few workers. We assume these were workers with good knowledge.

—if> The approach is cost-effective
Experiment 2: quality assessment

**Dataset** = output of best crowd mining approach of Exp 1

Pattern assessment **metrics**

- Reusability
- Novelty
- Usefulness
- Understandability
Crowd task **designs**

**Individual**: asks for assessment of the for metrics, given one pattern

**Pair-wise**: asks for each metric to choose which of two given patterns is better

+ **Expert assessment** for comparison
Results
Replaceability of experts

(a) Individual vs. Expert rating (avg of Likert ratings)

(b) Understandability pattern ranking in decreasing order of aggregated votes

---

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Mann-Whitney's test</th>
<th>Spearman's correlation coefficients ( \rho )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Expert vs. Individual</td>
</tr>
<tr>
<td>Reusability</td>
<td>( p = 5.787 \times 10^{-9}; U = 8029 )</td>
<td>-0.0783</td>
</tr>
<tr>
<td>Novelty</td>
<td>( p = 3.287 \times 10^{-13}; U = 8741 )</td>
<td>0.1212</td>
</tr>
<tr>
<td>Usefulness</td>
<td>( p = 6.392 \times 10^{-10}; U = 8197 )</td>
<td>0.0257</td>
</tr>
<tr>
<td>Understandability</td>
<td>( p = 5.744 \times 10^{-4}; U = 6870 )</td>
<td>0.0732</td>
</tr>
</tbody>
</table>

---

\( \Rightarrow \) Individual does **not** produce anything like the experts

\( \Rightarrow \) Pair-wise does **not** produce anything like the experts
5.2.2. Metrics and Statistical Tests.

We use four criteria to assess the quality of patterns. The two questions we want to study in the experiment design in Figure 16 in Appendix B.

We specifically compute a (partial) ranking, each pattern is given 6 chances to be voted. The task setting makes sure that each pattern appears exactly 6 times in different contexts. Finally, we order all patterns in decreasing order for each of the three approaches and quality criteria individually and convenience.

Individual vs. Expert

PairWise vs. Expert

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

The approaches could be used to filter out the worst patterns
Conclusion

• Using suitable task designs, the crowd is able to identify meaningful model patterns.
• More visibility into the dataset (e.g., to spot repetitions) does not help, to the contrary.
• We were not able to obtain reliable quality assessments from the crowd.
The real conclusion

• Model patterns can **really help** if suitably recommended and used
• The key problem is **finding** good patterns
• The **crowd** may be a viable alternative (or complement?) to computational approaches in identifying patterns