



POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA

Effective Identification and Reuse of Model Patterns in Service Orchestration Modeling

by Florian Daniel, florian.daniel@polimi.it

SummerSOC, Crete, Greece, June 26, 2017

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

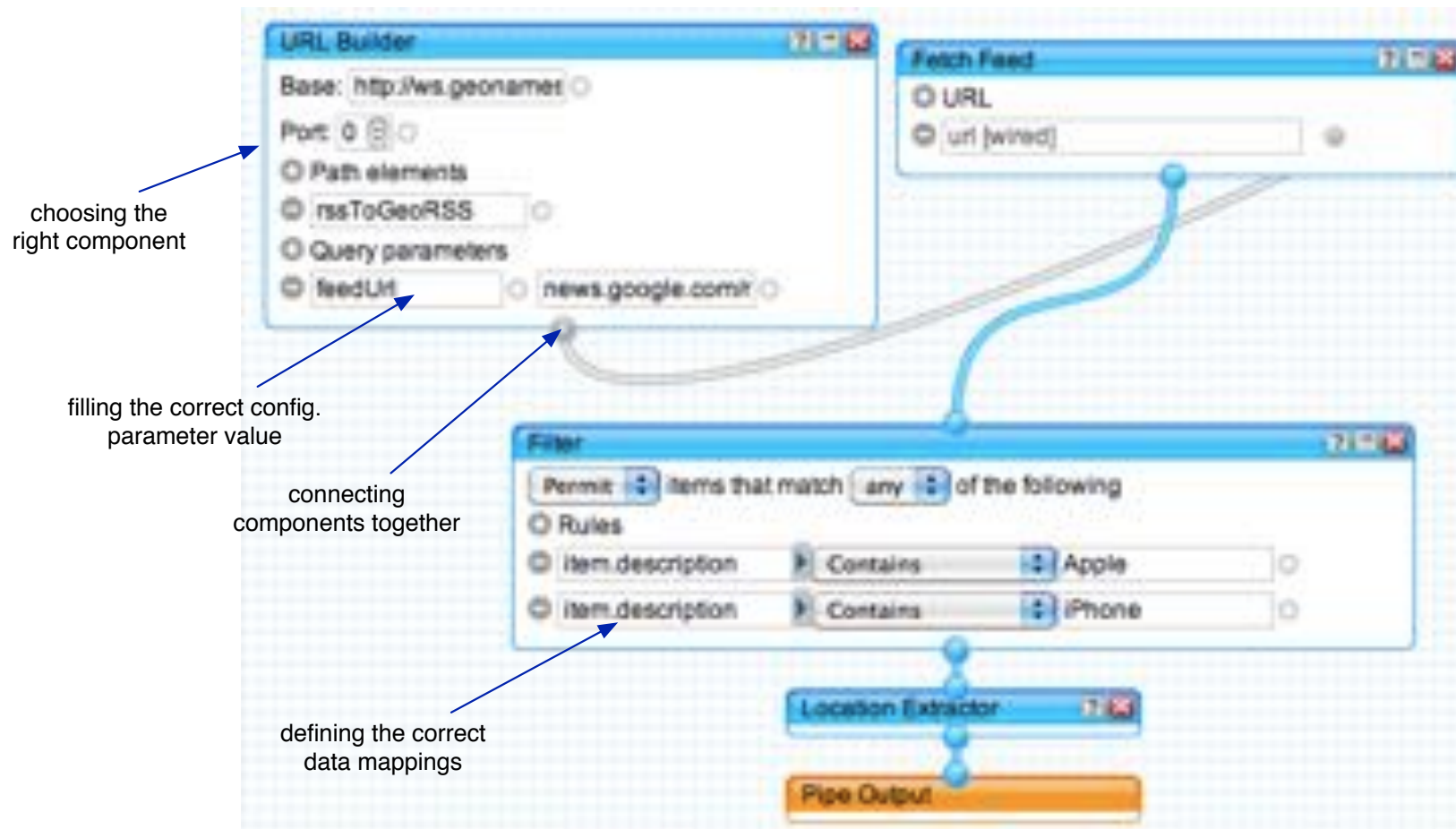
Research question

Does recommending model patterns really help developers
model faster / better?

A typical **mashup model pattern** (Yahoo! Pipes)

Mashup model: $m = \langle name, C, F, M, P \rangle$

Composition pattern model: $cp = \langle C, F, M, P, usage, date \rangle$



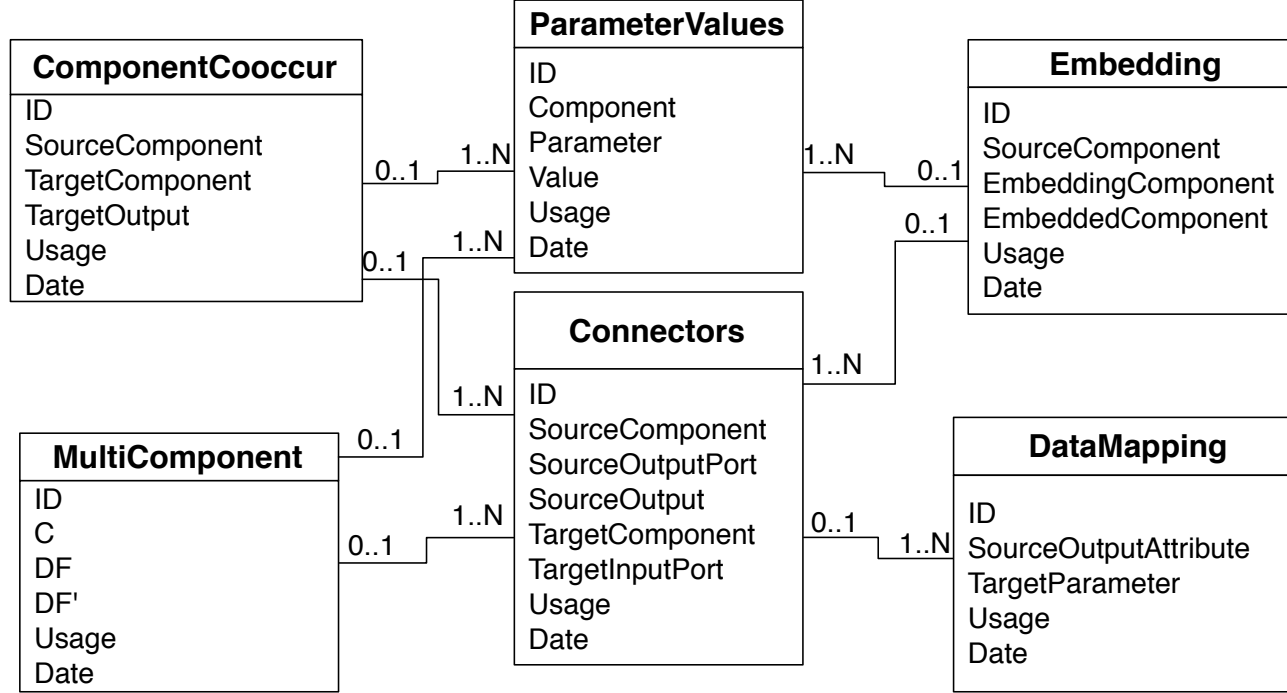
The pattern tells how to enrich an RSS feed with geo-coordinates and plot its items on a map

Pattern knowledge base

Algorithm 2: getRecurrentConnectors
Data: repository of mashup models M , minimum support of data flow connectors ($minsup_{df}$) Result: set of recurrent connectors F_{df}
<pre> 1 $DB_{df} = \text{array}()$; // database of data flow connector instances 2 foreach $m_i \in M$ do 3 $\text{append}(DB_{df}, m_i.DF)$; // fill with instances 4 $F_{df} = \text{set}()$; // set of recurrent data flow connectors 5 foreach $df_{xy} \in DB_{df}$ do 6 if $\text{computeSupport}(df_{xy}, DB_{df}) \geq minsup_{df}$ then 7 $F_{df} = F_{df} \cup \{df_{xy}\}$; 8 return F_{df}; </pre>

Algorithm 4: mineComponentCooccurrences
Data: repository of mashup models M , minimum support of data flow connectors ($minsup_{df}$), data mappings ($minsup_{dm}$) and parameter value assignments ($minsup_{pv}$) Result: set of component co-occurrence patterns with their corresponding dataflow connectors, data mappings and parameter values ($\{df_{xy}, VA_{xy}, VA_{xy}, DM_{xy}\}$)
<pre> 1 $F_{df} = \text{getRecurrentConnectors}(M, minsup_{df})$; 2 $DB = \text{array}()$; // database of recurrent connector instances 3 $Patterns = \text{set}()$; // set of component co-occurrence patterns 4 foreach $df_{xy} \in F_{df}$ do 5 $DB[df_{xy}] = \text{getConnectorInstances}(M, df_{xy})$; 6 // create databases for frequent itemset mining 7 $DBVA_{xy} = \text{array}()$; 8 $DBDM_{xy} = \text{array}()$; 9 foreach $df_{tsxy} \in DB[df_{xy}]$ do 10 $c_x = \text{source component of } df_{tsxy}$; 11 $c_y = \text{target component of } df_{tsxy}$; 12 $\text{append}(DBVA_{xy}, c_x.VA)$; 13 $\text{append}(DBVA_{xy}, c_y.VA)$; 14 $\text{append}(DBDM_{xy}, c_x.DM)$; 15 $F_{tsx} = \text{mineFrequentItemsets}(DBVA_{xy}, minsup_{pv})$; 16 $F_{tsy} = \text{mineFrequentItemsets}(DBVA_{xy}, minsup_{pv})$; 17 $F_{tsd} = \text{mineFrequentItemsets}(DBDM_{xy}, minsup_{pv})$; 18 // keep only those combinations of value assignments and 19 // data mappings that frequently occur together 20 $Coo = \text{set}()$; 21 foreach $(VA_x, VA_y, DM_x) \in F_{tsx} \times F_{tsy} \times F_{tsd}$ do 22 if $\text{computeSupport}((VA_x, VA_y, DM_x), DB[df_{xy}]) \geq minsup_{df}$ then 23 $Coo = Coo \cup \{(VA_x, VA_y, DM_x)\}$; 24 return $Patterns$; </pre>

Algorithm 3: getConnectorInstances
Data: repository of mashup models M , reference connector df_{xy} Result: array of connector instances $DB_{df_{xy}}$
<pre> 1 $DB_{df_{xy}} = \text{array}()$; // database of data flow connector instances 2 foreach $m_i \in M$ do 3 $\text{append}(DB_{df_{xy}}, m_i.DF \cap \{df_{xy}\})$; // fill with instances 4 return $DB_{df_{xy}}$; </pre>



Algorithm 1: mineParameterValues
Data: repository of mashup compositions M and minimum support ($minsup$) for the frequent itemset mining Result: set of parameter value patterns grouped by type of component.
<pre> 1 $C = \text{set of component instances in } M$; 2 $P = \text{array}()$; // list of parameter value patterns. 3 foreach $type$ of component t in C do 4 $C^t = \text{set of } c_i.VA$ with $c_i \in C$ such that $c_i.type = t$; //we 5 //get all the parameter value assignments of component 6 //instance of type is t. 7 $P[t] = \text{mineFrequentItemsets}(C^t, minsup)$; 8 $P[t] = P[t]'$; 9 return P; </pre>

Algorithm 1: mineConnectors
Data: repository of mashup models M , minimum support of data flow connectors ($minsup_{df}$) and data mappings ($minsup_{dm}$) Result: set of connectors with their corresponding data mappings ($\{df_{xy}, DM_{xy}\}$)
<pre> 1 $F_{df} = \text{getRecurrentConnectors}(M, minsup_{df})$; 2 $DB = \text{array}()$; // database of recurrent connector instances 3 $Patterns = \text{set}()$; // set of connector patterns 4 foreach $df_{xy} \in F_{df}$ do 5 $DB[df_{xy}] = \text{getConnectorInstances}(M, df_{xy})$; 6 // create database for frequent itemset mining 7 $DBDM_{xy} = \text{array}()$; 8 foreach $df_{tsxy} \in DB[df_{xy}]$ do 9 $c_y = \text{target component of } df_{tsxy}$; 10 $\text{append}(DBDM_{xy}, c_y.DM)$; 11 $F_{tsy} = \text{mineFrequentItemsets}(DBDM_{xy}, minsup_{dm})$; 12 // construct the connector patterns 13 foreach $DM_y \in F_{tsy}$ do 14 $Patterns = Patterns \cup \{(df_{xy}, DM_y)\}$ 15 return $Patterns$; </pre>

Algorithm 7: mineComponentEmbeddings
Data: repository of mashup compositions M , minimum supports for component embeddings ($minsup_{comp}$), data mappings ($minsup_{dm}$) and parameter value ($minsup_{pv}$) Result: list of component embedding patterns with their corresponding dataflow connectors, data mappings and parameter value assignments ($\{df_{xy}, DM_{xy}, DM_{xy}, VA_{xy}\}$)
<pre> 1 $DB_{comp} = \text{array}()$; 2 foreach $m_i \in M$ do 3 foreach $df_{xy} \in m_i.DF$ do 4 if c_x includes c_y and c_y includes c_x then 5 $\text{embed}(c_x, c_y, df_{xy})$; 6 $DB_{comp} = \text{append}(DB_{comp}, \text{embed}(c_x, c_y, df_{xy}))$; 7 return DB_{comp}; 8 if $\text{computeSupport}(DB_{comp}, minsup_{comp}) \geq minsup_{comp}$ then 9 $F_{comp} = \text{mineFrequentItemsets}(DB_{comp}, minsup_{comp})$; 10 $F_{comp} = \text{array}()$; 11 foreach $m_i \in M$ do 12 foreach $df_{xy} \in m_i.DF$ do 13 foreach $df_{tsxy} \in m_i.DF \times m_i.DF$ do 14 if $m_i.DF \cap df_{tsxy} \neq \emptyset$ then 15 $DM_x = \text{map}(c_x, c_y, df_{tsxy})$; 16 $DM_y = \text{map}(c_y, c_x, df_{tsxy})$; 17 $DM_{xy} = \text{map}(c_x, c_y, df_{tsxy})$; 18 $DB_{comp} = \text{append}(DB_{comp}, DM_x \cup DM_y \cup DM_{xy})$; 19 $F_{comp} = \text{mineFrequentItemsets}(DB_{comp}, minsup_{comp})$; 20 $F_{comp} = \text{array}()$; 21 foreach $m_i \in M$ do 22 foreach $df_{xy} \in m_i.DF$ do 23 foreach $df_{tsxy} \in m_i.DF \times m_i.DF$ do 24 if $m_i.DF \cap df_{tsxy} \neq \emptyset$ then 25 $VA_x = \text{map}(c_x, c_y, VA)$; 26 $VA_y = \text{map}(c_y, c_x, VA)$; 27 $VA_{xy} = \text{map}(c_x, c_y, VA)$; 28 $DB_{comp} = \text{append}(DB_{comp}, VA_x \cup VA_y \cup VA_{xy})$; 29 $F_{comp} = \text{mineFrequentItemsets}(DB_{comp}, minsup_{comp})$; 30 $C_{comp} = \text{set}()$; 31 foreach $(df_{xy}, DM_{xy}, VA_{xy}) \in F_{comp} \times F_{comp} \times F_{comp}$ do 32 if $\text{computeSupport}((df_{xy}, DM_{xy}, VA_{xy}), minsup_{comp}) \geq minsup_{comp}$ then 33 $C_{comp} = C_{comp} \cup \{(df_{xy}, DM_{xy}, VA_{xy})\}$; 34 return C_{comp}; 35 foreach $(df_{xy}, DM_{xy}, VA_{xy}) \in C_{comp}$ do 36 $Patterns = Patterns \cup \{(df_{xy}, DM_{xy}, VA_{xy})\}$ 37 return $Patterns$; </pre>

Patterns **mined** from a dataset of 970 “most popular” pipes models of Yahoo! Pipes (association rule mining, frequent itemset mining + predefined topologies)

C. Rodriguez, S. Roy Chowdhury, F. Daniel, H.R. Motahari Nezhad and F. Casati. Assisted Mashup Development: On the Discovery and Recommendation of Mashup Composition Knowledge. In *Web Services*, Springer, 2014, Pages 683-708.

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

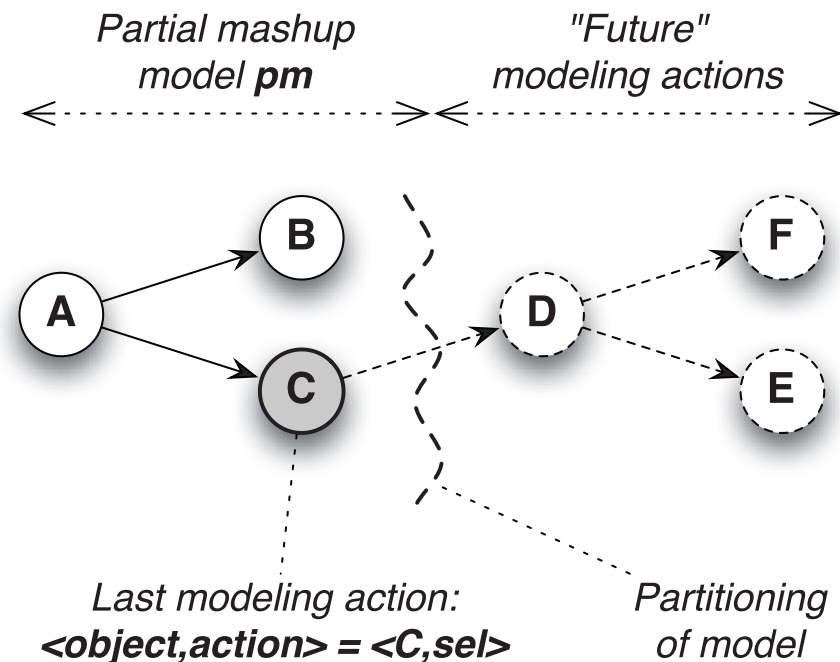
Expert: ranks contextual recommendations according to experts' past component preferences; cloning h-index

Modeling **test cases**

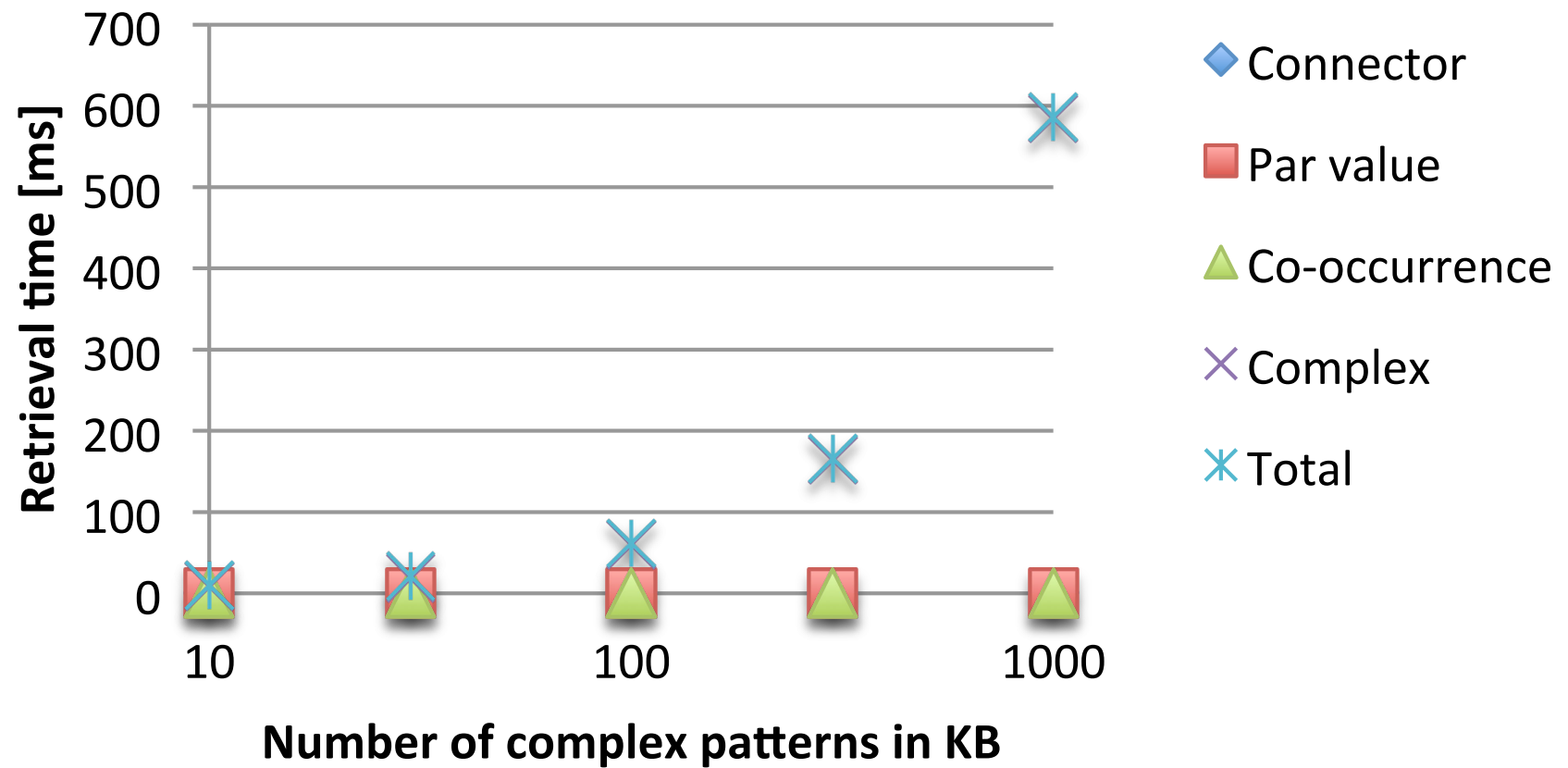
100 pipes models, different from the ones used to mine patterns

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

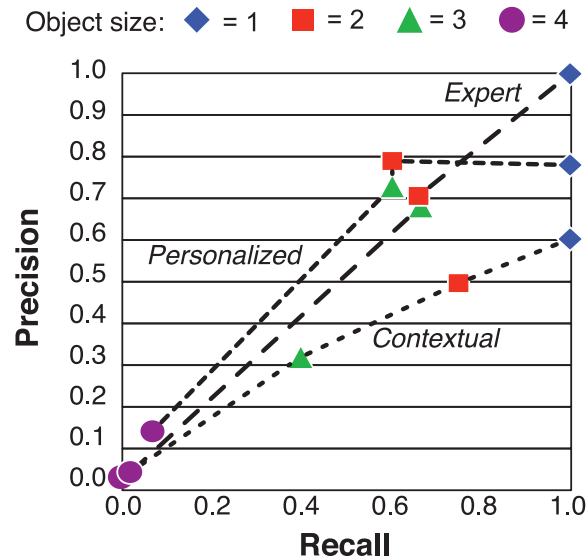
- 356 with object size 1
- 227 with object size 2
- 212 with object size 3
- 61 with object size 4



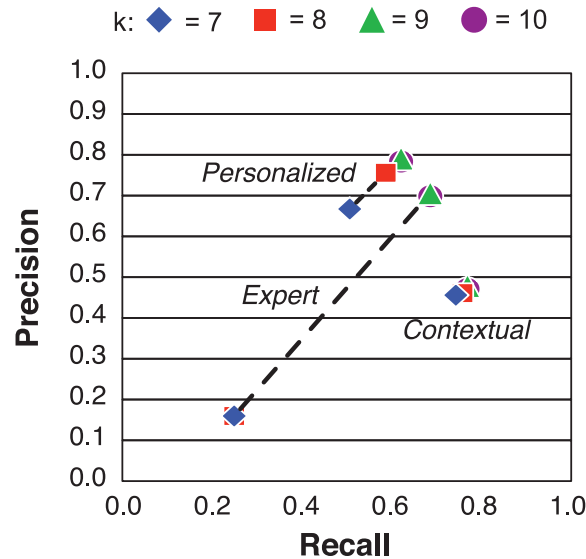
In-browser **performance** of recommendations



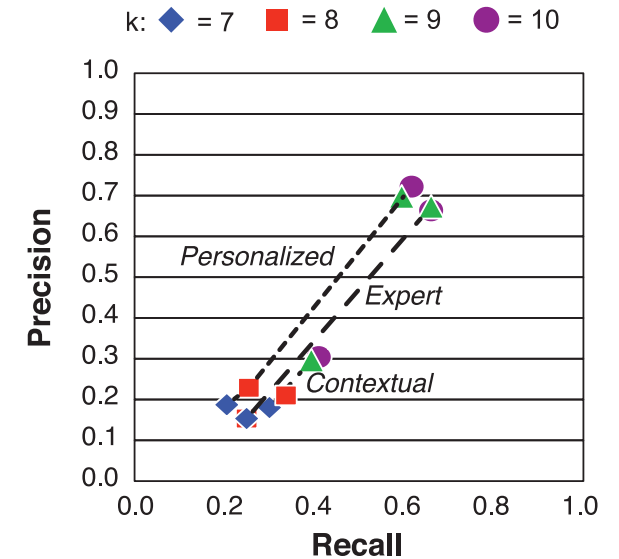
Precision and recall: $P = \frac{|TP|}{|TP|+|FP|}$ $R = \frac{|TP|}{|TP|+|FN|}$



(a) P/R for **varying object size** ($k=10$)



(b) P/R for **varying k** (object size is 2)

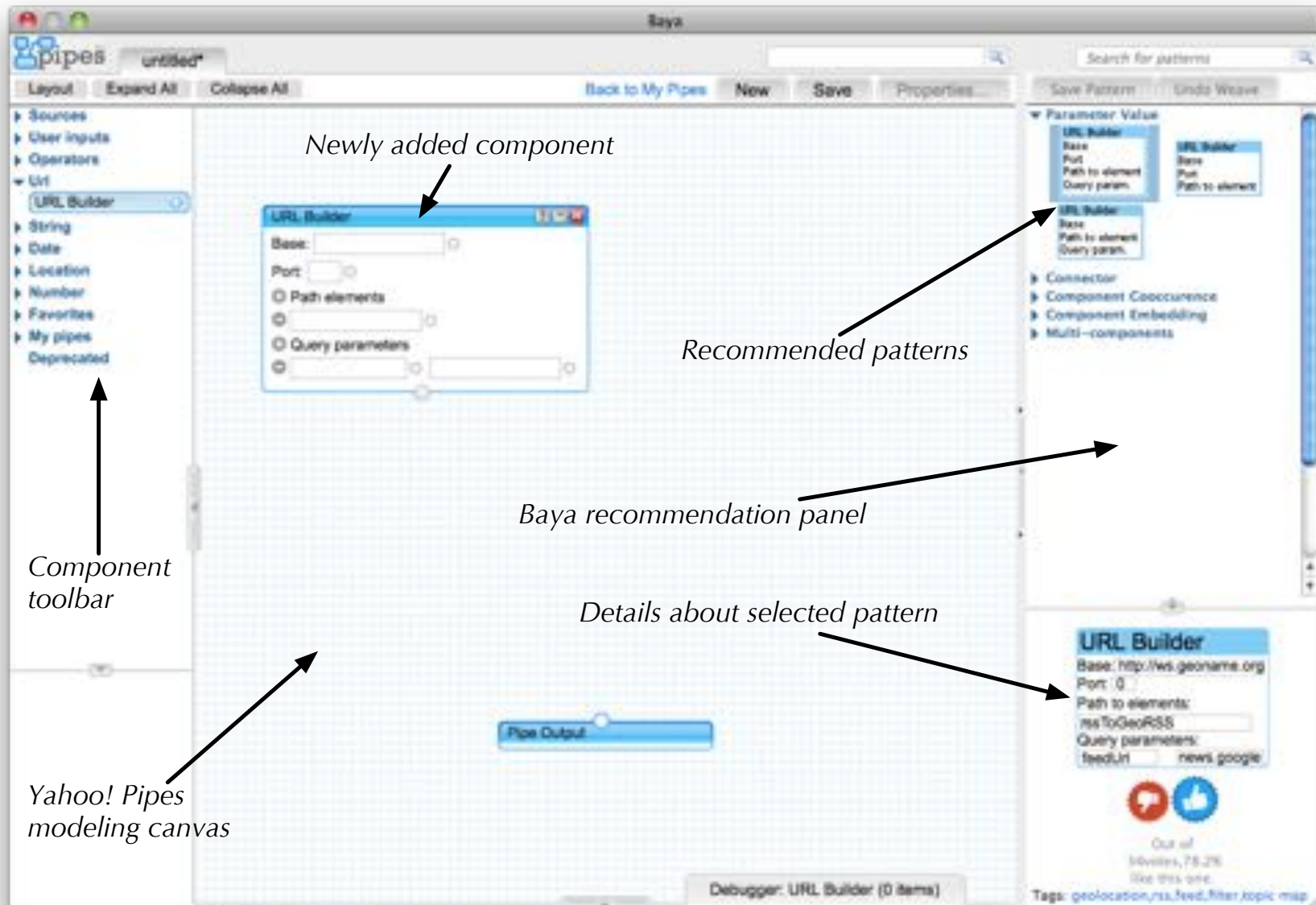


(c) P/R for **varying k** (object size is 3)

High performance in response to stepwise modeling actions
Retrieve at least 8-9 recommendations

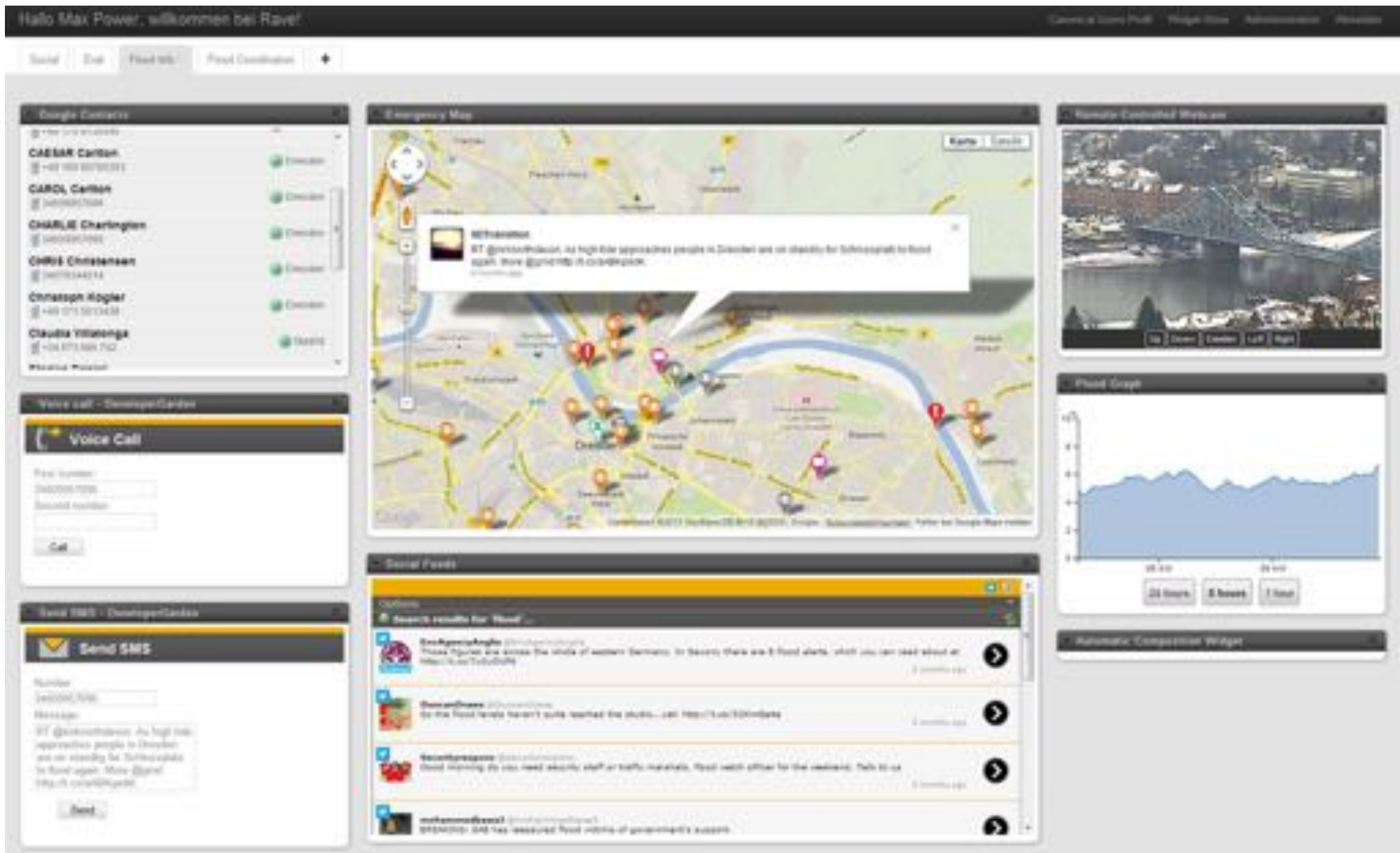


= assisted development in Yahoo! Pipes



for developers

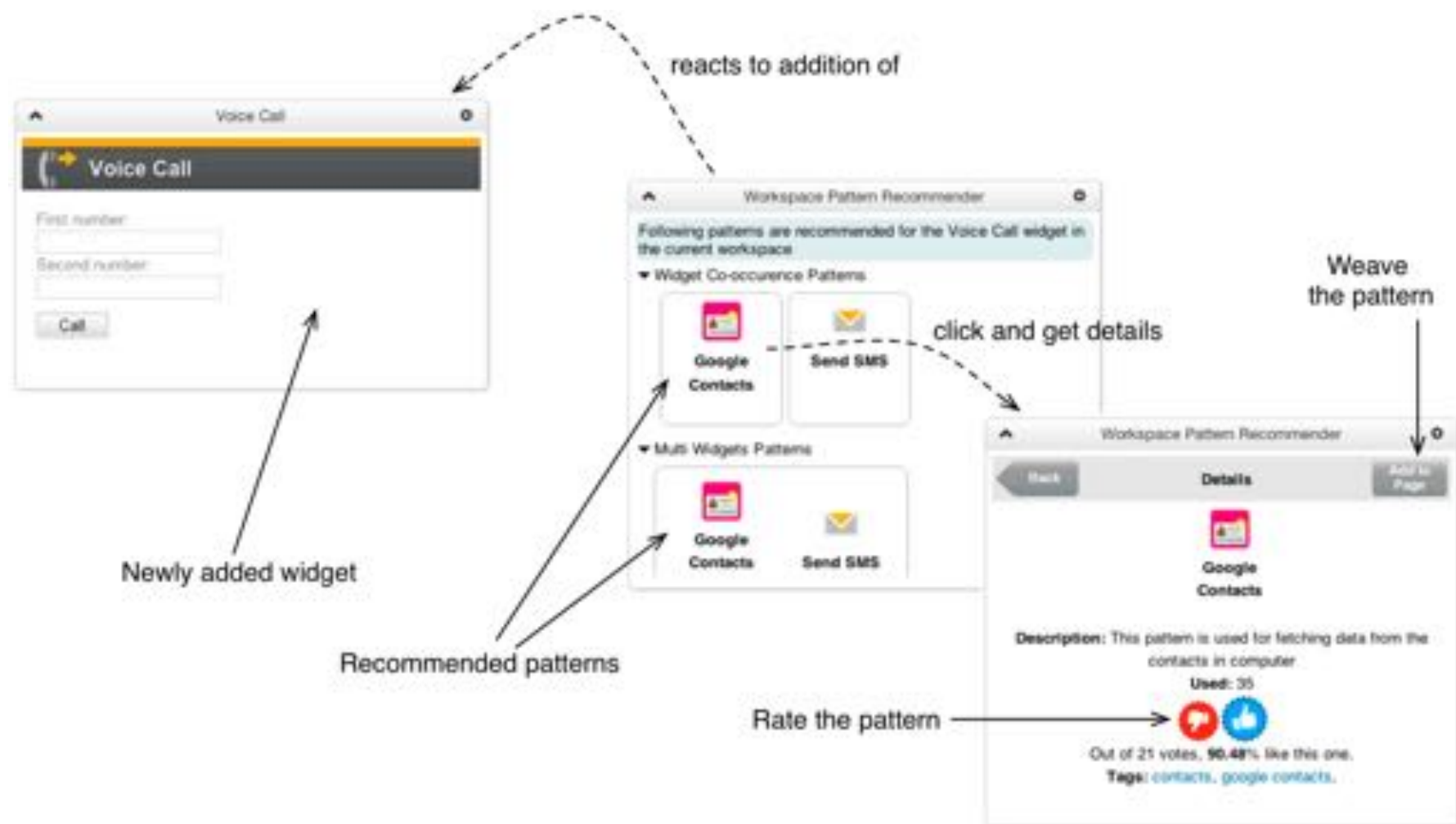
 **melette** = extension of Apache Rave for UI mashups



The screenshot displays the melette UI mashup interface, which is a dashboard for emergency response coordination. The interface is organized into several sections:

- Header:** A dark bar at the top with the text "Hallo Max Power, willkommen bei Rave!" and navigation links for "Current & Recent Posts", "Project Home", "Administration", and "About".
- Navigation:** A row of tabs below the header: "Social", "Data", "Feed Info", and "Feed Coordinates", with a plus sign for additional options.
- Google Contacts:** A list of contacts on the left side, including Caesar Carlson, Carol Carlson, Charlie Charleston, Chris Christensen, Christoph Koppier, and Claudia Yllanonga, each with a status indicator.
- Voice Call - Desktop/Landline:** A section for making voice calls, featuring a "Voice Call" button, input fields for "First Number" and "Second Number", and a "Call" button.
- Send SMS - Desktop/Landline:** A section for sending SMS messages, featuring a "Send SMS" button, input fields for "Number" and "Second Number", a "Message" field, and a "Send" button.
- Emergency Map:** A central map showing a river and surrounding areas, with numerous red and orange location pins. A pop-up window displays a tweet from @kriszofhauert about high tide approaching people in Dresden.
- Social Feeds:** A section at the bottom left displaying a list of tweets related to flood alerts and emergency response, including mentions of @kriszofhauert and @kriszofhauert.
- Flood Graph:** A line graph on the right side showing flood levels over time, with a blue area representing the data. It includes a legend for "24 hours", "8 hours", and "1 hour".
- Automatic Compression Widget:** A small widget at the bottom right.

for end-users



User studies

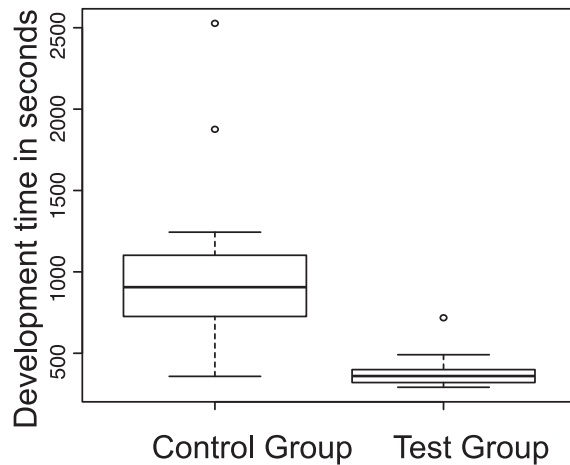
H1: Baya speeds up mashup development

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)

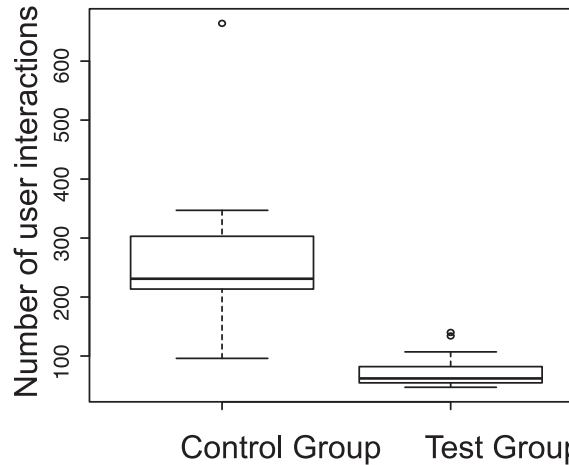


(a) mean development time in designing a pipe without and with Baya

$$\mu_{dev,ctrl} = 1027.1s$$

$$\mu_{dev,test} = 384.9s$$

Retain H1
(p=0.00045)

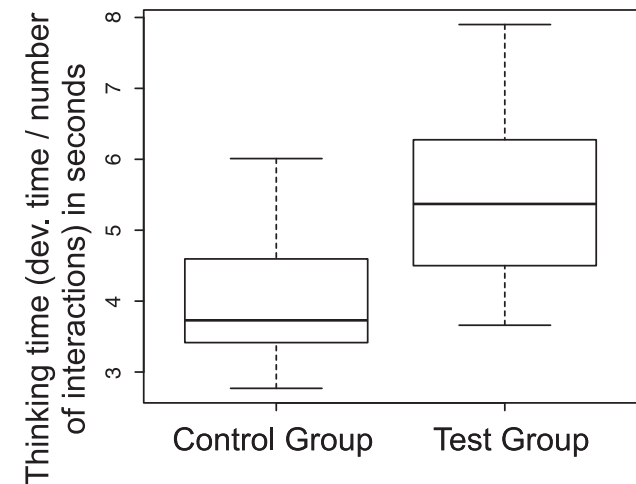


(b) mean number of user interactions in designing a pipe without and with Baya

$$\mu_{int,ctrl} = 258.9$$

$$\mu_{int,test} = 74.3$$

Retain H2
(p=0.00009)



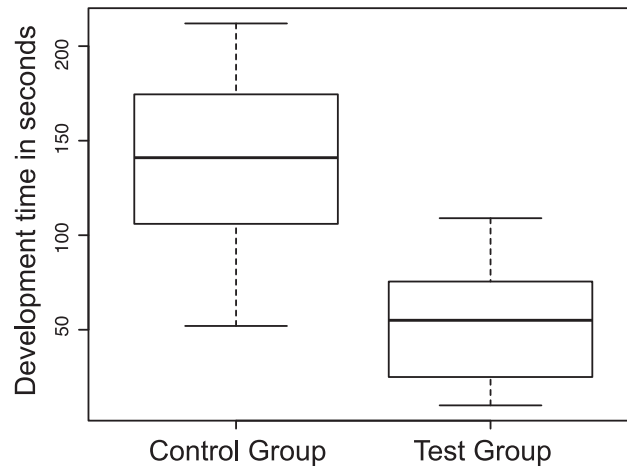
(c) mean thinking time for designing a pipe without and with Baya

$$\mu_{th,ctrl} = 4.0s$$

$$\mu_{th,test} = 5.5s$$

Reject H3
(p=0.00209)

Independent user studies by partners in the EU FP7 project **melette**: Baya for Apache Rave, 44 participants (admins)



(a) user study with Apache Rave (China)

$$\mu_{dev,ctrl} = 139.8s$$

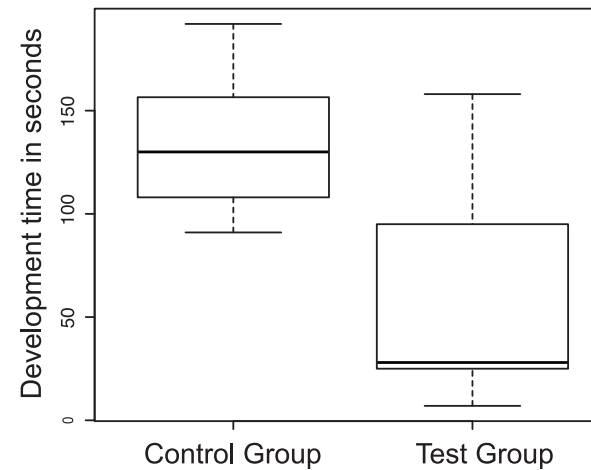
$$\mu_{dev,test} = 54.6s$$

Retain H1

(p=0.0001)



T-Systems Multimedia Solutions



(b) user study with Apache Rave (Germany)

$$\mu_{dev,ctrl} = 132.1s$$

$$\mu_{dev,test} = 58.9s$$

Retain H1

(p=0.0007)

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

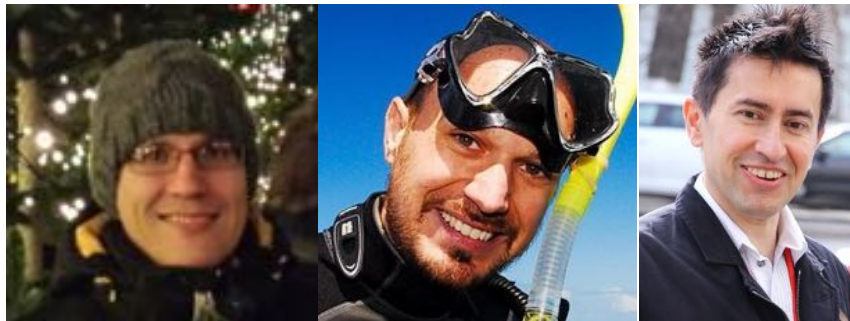
In conclusion

- 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

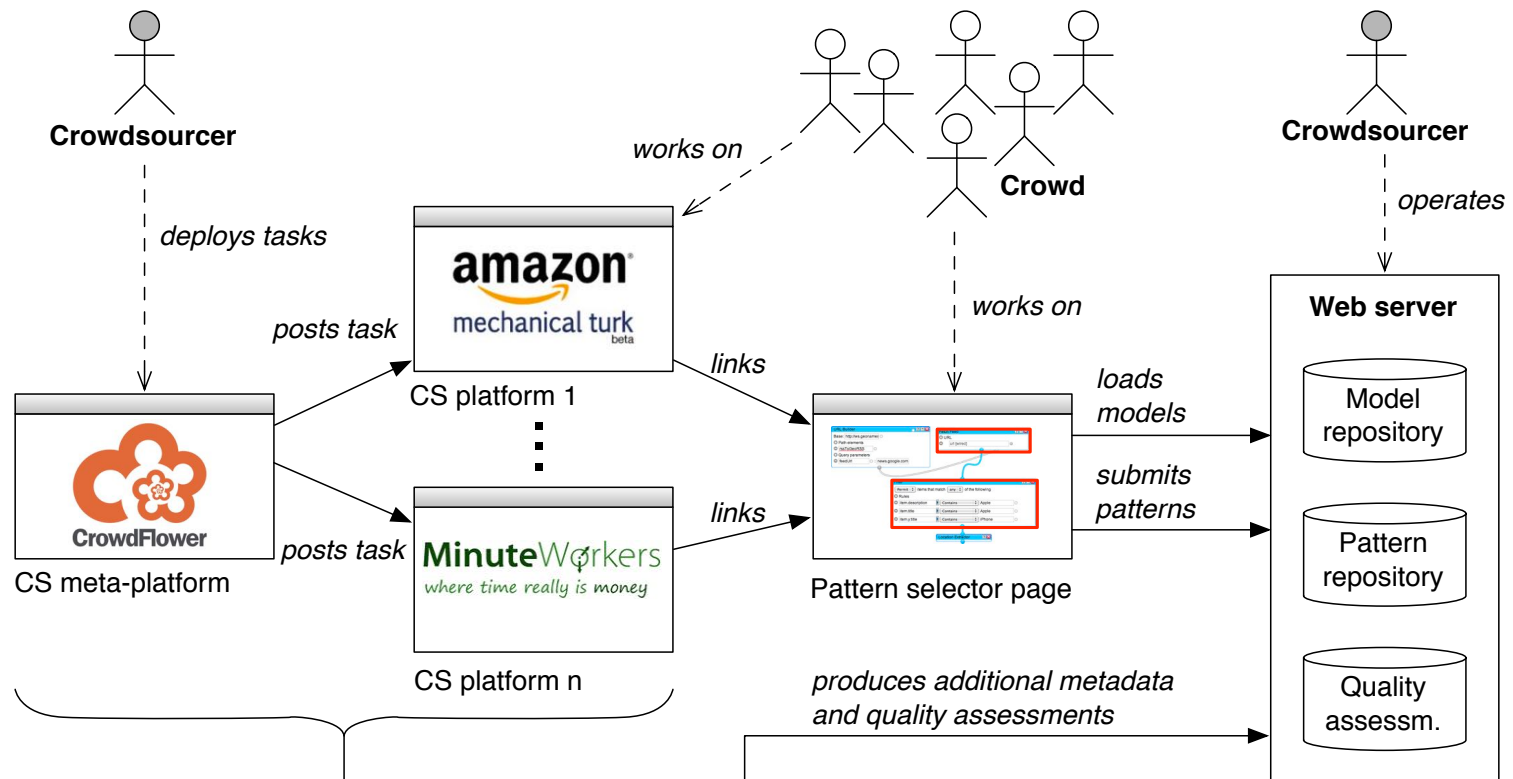
Pattern mining, that is, the automated discovery of patterns from data, is a mathematically complex and computationally demanding problem that is generally not manageable by humans. In this article, we focus on small datasets and study whether it is possible to mine patterns with the help of the crowd 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 crowdsource the **quality assessment** of identified patterns?

Architecture



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

* C. Rodriguez, S. Roy Chowdhury, F. Daniel, H.R. Motahari Nezhad and F. Casati. Assisted Mashup Development: On the Discovery and Recommendation of Mashup Composition Knowledge. In *Web Services Foundations*, Springer, 2014, Pages 683-708.

Screen shot of the **Naive** task design

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.

The screenshot displays the 'Naive' task design interface, which is divided into two main sections: a form for specifying pipe metadata and a visual pipe model diagram.

Form Section (Left):

- 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
- Submit** button

Pipe Model Diagram (Right):

The diagram illustrates a sequence of pipe components connected by arrows, representing a data flow. The components are:

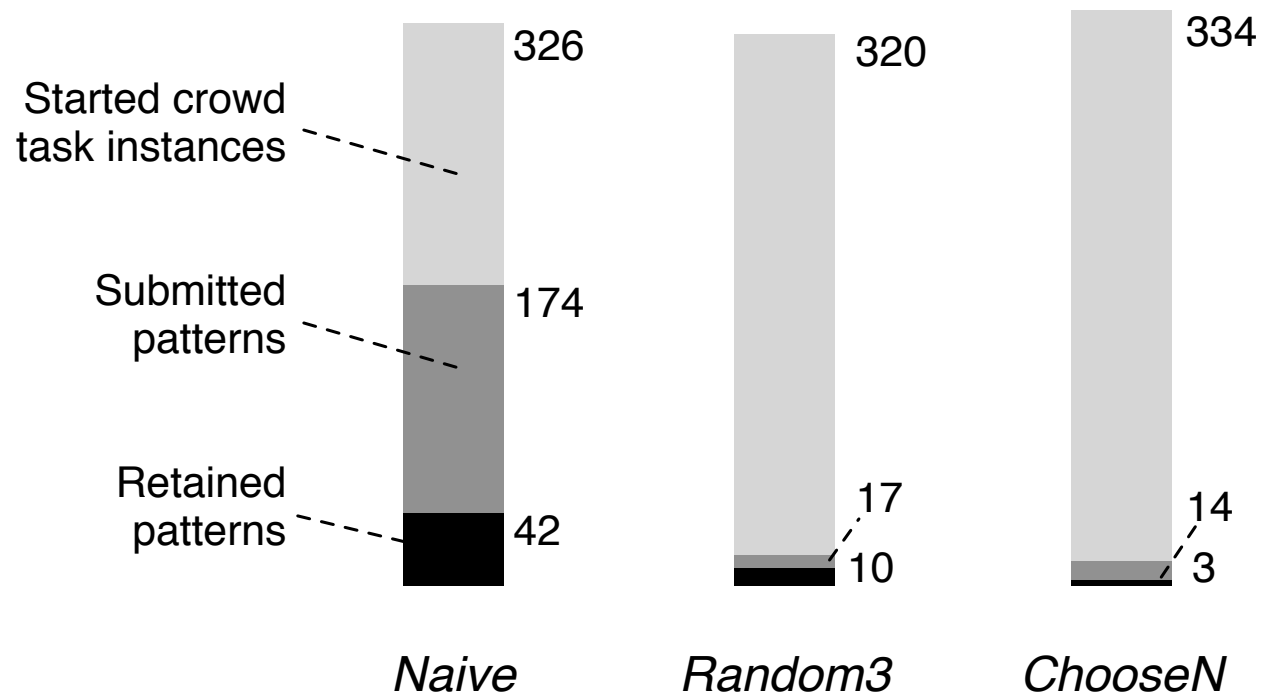
- Fetch Feed:** A component that retrieves feeds from a URL. It has a 'URL' field and a 'Fetch' button.
- Sort:** A component that sorts the fetched feeds. It has a 'Sort by' field (set to 'item.published') and a 'Sort order' field (set to 'ascending').
- Take:** A component that takes a subset of the sorted feeds. It has a 'Take after position' field (set to 'first').
- Item Count (number):** A component that counts the number of items. It has a 'Name' field (set to 'N') and a 'Default' field (set to '1').
- Label feed:** A component that labels the feeds. It has a 'Name' field (set to 'Label') and a 'Default' field (set to '1st item').
- String Builder:** A component that builds a string from the labeled feeds. It has a 'String' field (set to 'item.title') and a 'String' field (set to '\$1').

The diagram also includes a 'Pipe Output' button at the bottom.

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

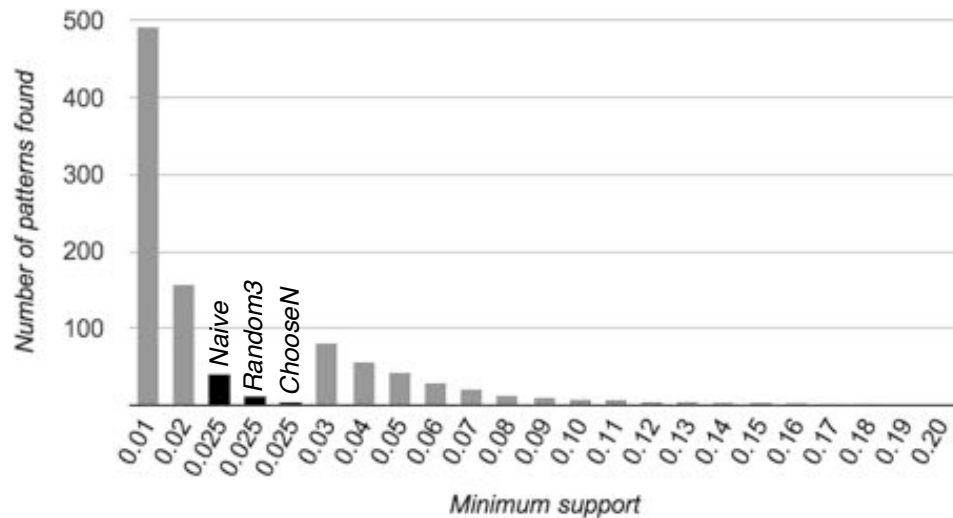
Results

Crowd task instances vs. patterns collected

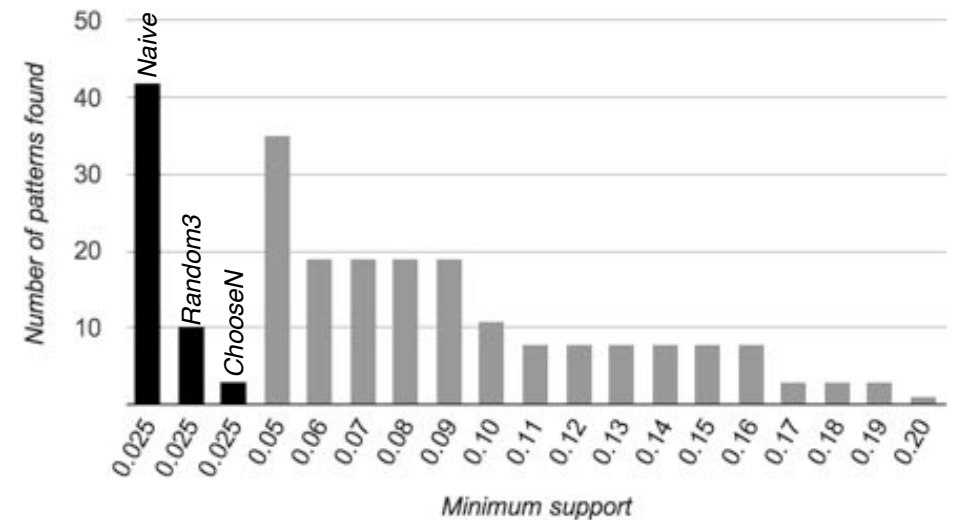


Retained patterns = **valid patterns**, manually checked

Number of patterns identified



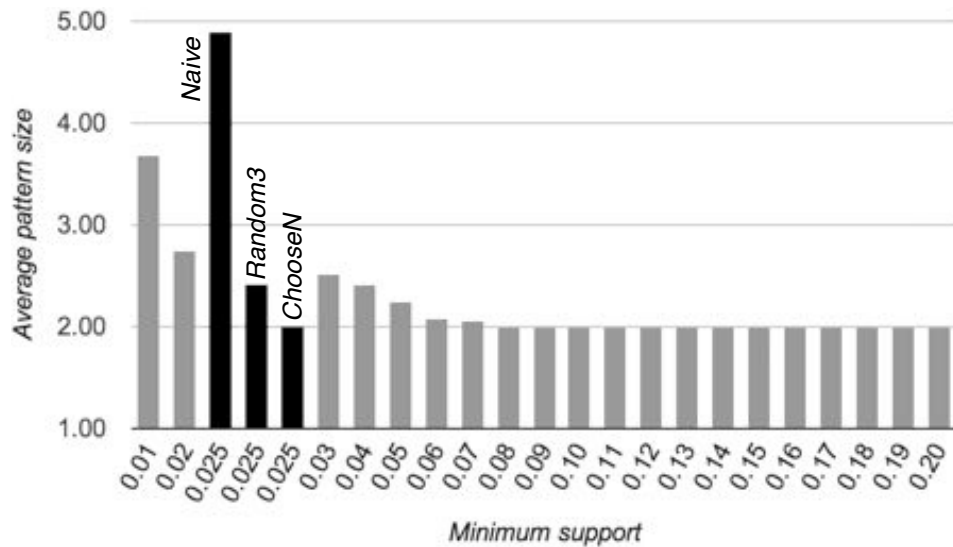
(a) Machine⁹⁹⁷ (gray) compared to the crowd (black)



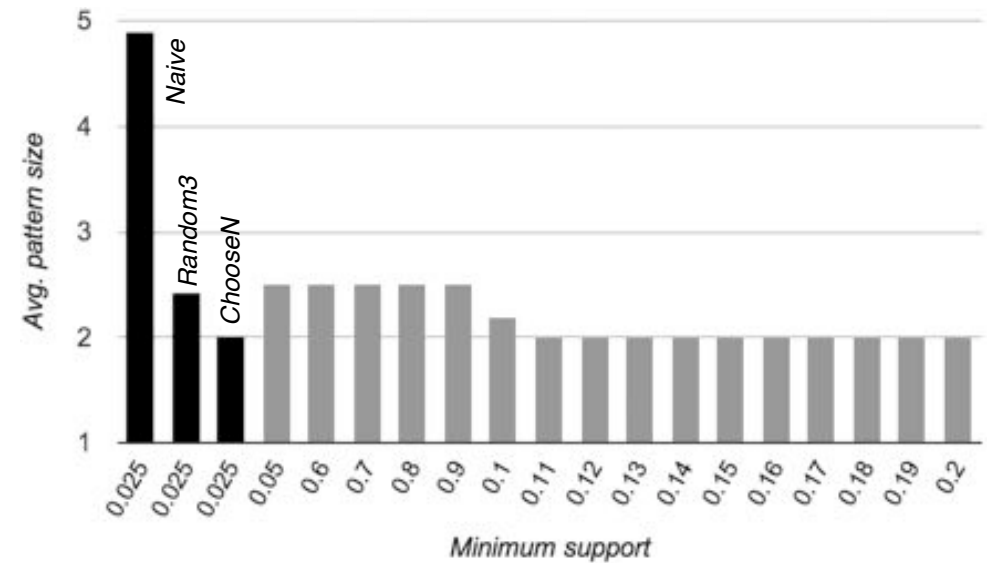
(b) Machine⁴⁰ (gray) compared to the crowd (black)

—> Yes, it is possible to identify patterns with the crowd

Average pattern sizes



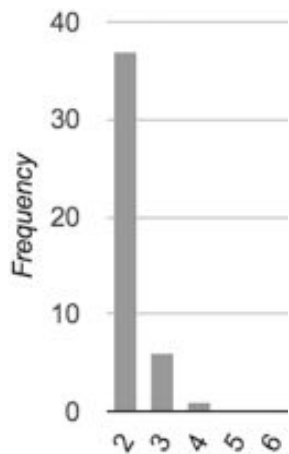
(a) *Machine*⁹⁹⁷ (gray) compared to the crowd (black)



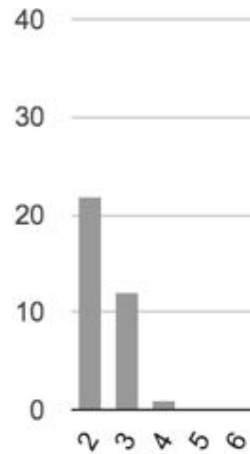
(b) *Machine*⁴⁰ (gray) compared to the crowd (black)

—> The patterns identified by the crowd are in average bigger

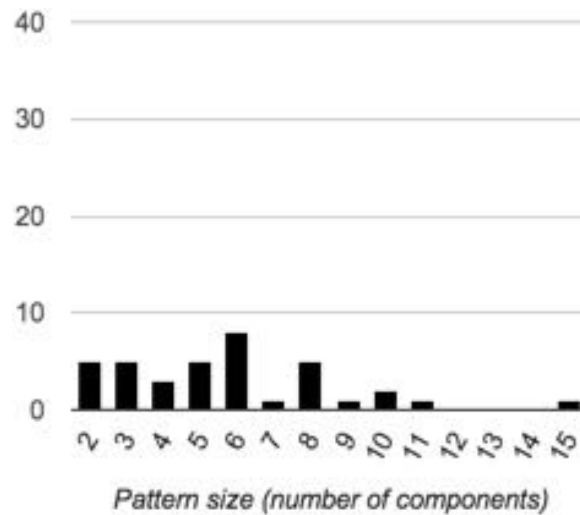
Distribution of pattern sizes



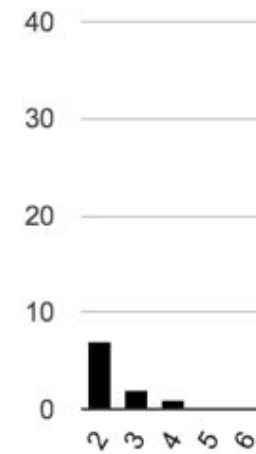
(a) *Machine*⁹⁹⁷:
44 patterns with
support of 0.05



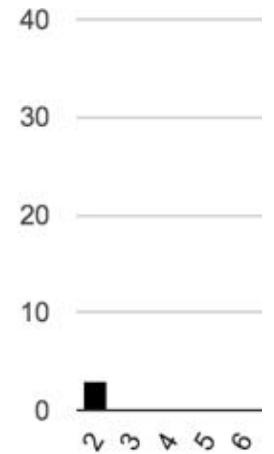
(b) *Machine*⁴⁰:
35 patterns with
support of 0.05



(c) *Naive*
with 42 patterns



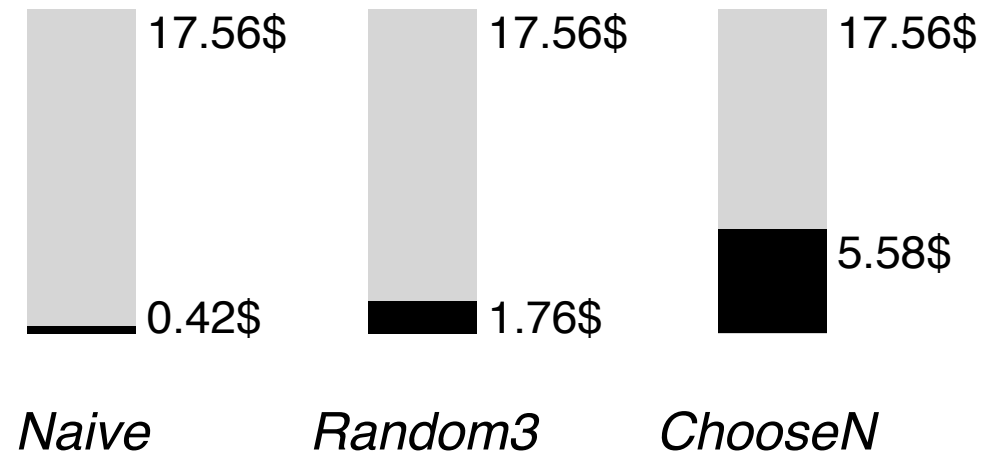
(d) *Random3*
with 10 patterns



(e) *ChooseN*
with 3 patterns

—> The domain knowledge captured by Naive is even complex

Cost



—> 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**

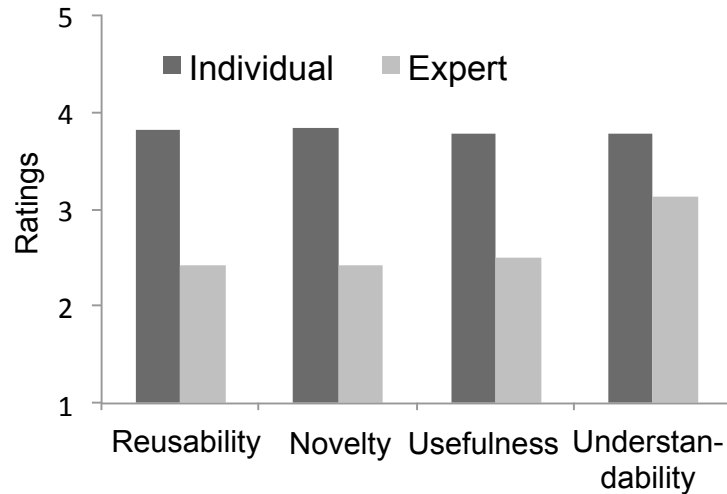
Individual: asks for assessment of the for metrics, given one pattern

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

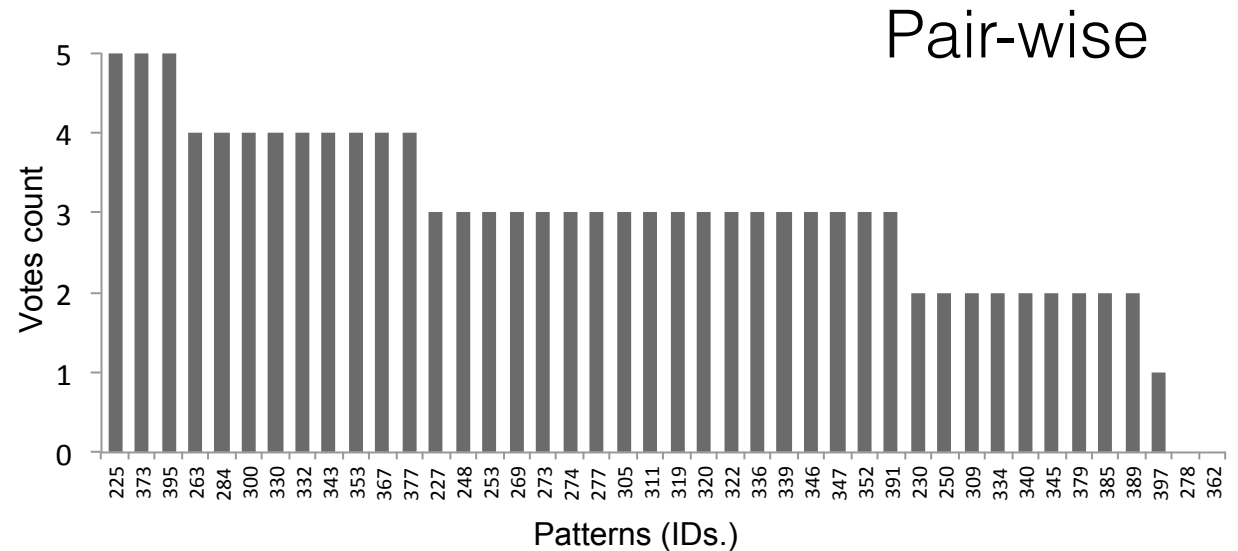
+ **Expert assessment** for comparison

Results

Replaceability of experts



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



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

Criteria	Mann-Whitney's test	Spearman's correlation coefficients ρ	
		<i>Expert vs. Individual</i>	<i>Expert vs. PairWise</i>
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

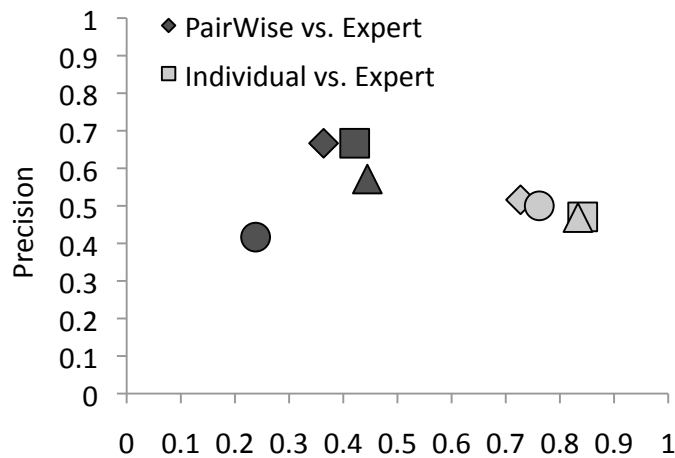
—> Individual does **not** produce anything like the experts

—> Pair-wise does **not** produce anything like the experts

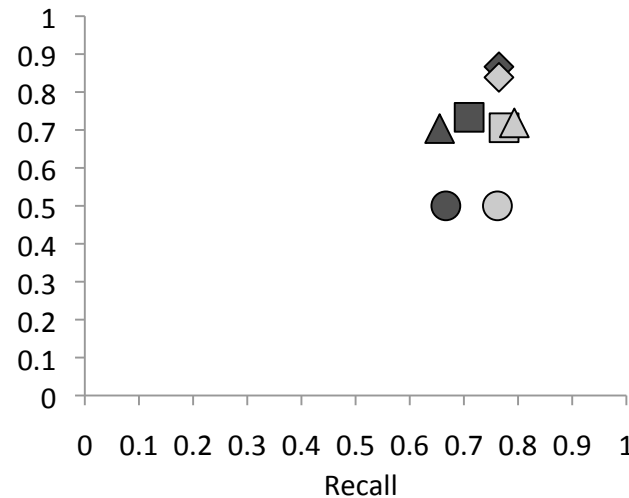
Precision and recall

$$P = \frac{TruePos}{TruePos + FalsePos}$$

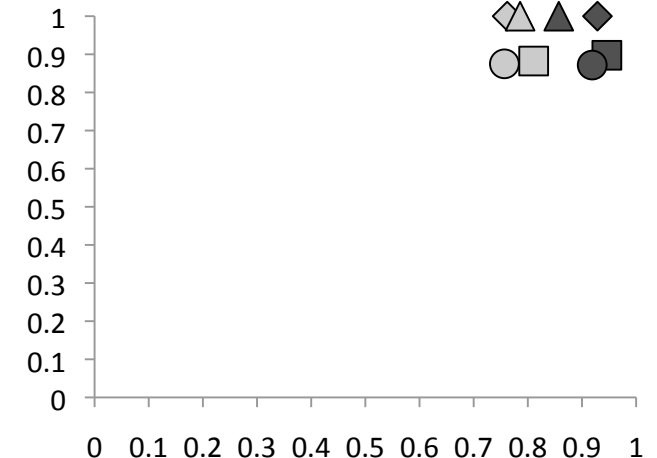
$$R = \frac{TruePos}{TruePos + FalseNeg}$$



(a) 25th percentile



(b) 50th percentile



(c) 75th percentile

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