YAM: a Schema Matcher Factory
Fabien Duchateau, Remi Coletta, Zohra Bellahsene, Renée J. Miller

► To cite this version:
RR-09018, 2009. lirmm-00399184

HAL Id: lirmm-00399184
https://hal-lirmm.ccsd.cnrs.fr/lirmm-00399184
Submitted on 25 Jun 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
YAM: a Schema Matcher Factory

Fabien Duchateau, Remi Coletta, Zohra Bellahsene
LIRMM
Univ. Montpellier 2
34392 Montpellier - France
firstname.name@lirmm.fr

Renée J. Miller
Univ. of Toronto
40 St. George Street
Toronto ON M5S 2E4, Canada
miller@cs.toronto.edu

ABSTRACT
In this paper, we present YAM, a schema matcher factory.
YAM (Yet Another Matcher) is not (yet) another schema
matching system as it enables the generation of a la carte
schema matchers according to user requirements. These re-
quirements include a preference for recall or precision, a
training data set (schemas already matched) and provided
expert correspondences. YAM uses a knowledge base that
includes a (possibly large) set of similarity measures and
classifiers. Based on the user requirements, YAM learns
how to best apply these tools (similarity measures and clas-
sifiers) in concert to achieve the best matching quality.
In our demonstration, we will let users apply YAM to build
the best schema matcher for different user requirements.

1. OVERVIEW OF YAM

YAM (Yet Another Matcher) is a meta-matching tool,
which generates a dedicated schema matcher, i.e., a
schema matcher which is best suited, in terms of quality,
for a given schema matching scenario and according to user
inputs. The motivations leading to our work are:

- There is no schema matching tool which performs best
  for all matching scenarios. Although matching tools en-
  able the user to tune some parameters (strategies, weights,
  thresholds, etc.), the algorithm used by the matching tool
  stays the same, for instance, COMA++'s aggregation func-
  tion [1] or Similarity Flooding's graph propagation algo-
tools by tuning the input parameters used by the matcher.
Specifically, eT uner finds the best parameter settings for a
given matching algorithm. On the contrary, YAM is able to
produce the best schema matcher for a given scenario. Each
generated schema matcher may use a different algorithm (an
aggregation function, Bayes network, decision tree, etc.) and
different similarity measures.

- YAM uses some user input(s), but of a different form.
Specifically, YAM can use an (optional) preference between
precision and recall, and some expert correspondences (that
is, a small number of correct matches). This small amount of
input enables the use of supervised learning to create a ded-
cicated schema matcher. YAM is able to convert user time
spent to give preferences into better quality results. Indeed,
most schema matching tools focus on a better precision, but
this is not the best choice in terms of post-matching effort,
I.e., the quantity of work required by an expert to correct dis-
covered correspondences. Technically speaking, it is easier
for the expert to validate (or not) a discovered correspon-
dence than to manually browse two large schemas for new
correspondences that the tool may have missed.

1.1 Architecture

Figure 1 depicts the YAM architecture. YAM has two
phases: a learning phase that produces a dedicated schema
matcher and a matching phase in which the dedicated matcher
is used over new input schemas. The learning process can
use a preference for precision and recall tradeoff and some
expert correspondences (from a domain of interest, or for the
schemas to be matched). The Knowledge Base (KB) stores
a set of classifiers, a set of similarity measures, and pairs
of schemas which have already been matched (with corre-
spondences). Thanks to these inputs and the KB, the learn-
ing process is able to generate a dedicated schema matcher.
This step is described with more details in Section 1.2. Note
that using the whole content of the KB enables learning of
a robust schema matcher, i.e., a default matcher that pro-
vides the best average results. This robust schema matcher
is useful when YAM must directly work as a generic schema
matcher, i.e., when the user has not provided any inputs.
The second phase performs the matching. It requires as in-
put the schemas to be matched, and the dedicated schema
matcher that has been previously generated. It outputs a
list of discovered correspondences between the schemas. The
current version of YAM includes 20 classifiers from the Weka
library\footnote{http://www.cs.waikato.ac.nz/~ml/weka} and 30 similarity measures, including all the mea-
sures from the Second String project\footnote{http://secondstring.sourceforge.net}. YAM is able to parse
edge-labeled trees (a simple abstraction that can be used for
XML schemas, web interfaces, etc.) and the knowledge base
contains a large set of real schemas from various domains.
(betting, hotel booking, dating, etc.) gathered from the web [3]. The current KB contains more than 250 schemas, among which 200 pairs have already been matched.

1.2 Learning Process

Similarly to machine learning classifiers, a matching tool classifies each pair of schema elements (or correspondence), by labelling them either as relevant or irrelevant. In YAM, we train a large set of classifiers on schemas which have already been matched, i.e. training schemas. Two errors can occur while training: discovering an irrelevant correspondence (a.k.a. false positive) and missing a relevant correspondence (a.k.a. false negative). The first error decreases precision while the second one decreases recall. Classifiers usually assign the same weight for both errors. But it is possible to promote one of them. Thus, YAM also takes as input a user preference for either recall or precision. It is able to generate a schema matcher that favors this preference.

Among the 20 classifiers that have been learned, the final step consists of selecting the best one. To fulfill this goal, a cross-validation is applied against the training schemas. The classifier which manages to discover, between the training schemas, most of these expert correspondences and the fewest irrelevant correspondences is selected as the dedicated schema matcher.

2. DEMONSTRATION SCENARIOS

In this demo, we show the capability of our tool for generating a dedicated schema matcher, according to user inputs.

We have used 10 scenarios from various domains (hotel booking, currency, courses, etc.). They are available with the demo. We run our experiments 30 times to limit impact of randomness during learning.

Without input. Let us imagine that we would like to match two hotel booking webforms. A user selects the schemas to be matched by clicking on the add schemas button. In this first case, (s)he does not want to give more input and (s)he clicks the learn and match button both to generate a schema matcher and to use it to match the schemas. As the user did not give any expert correspondences, YAM uses the whole KB to learn the most robust schema matcher, which is based on C4.5, a decision tree. First, this robust schema matcher is displayed in the bottom left part of YAM’s interface. Then, the discovered correspondences are shown in the right part of YAM using lines between schema elements. The robust schema matcher achieves an average 58% f-measure on the 10 scenarios.

Promoting recall. Suppose the user thinks the recall is too low. That is, (s)he feels the matcher has missed some correspondences. (S)He/She adjusts the precision / recall slide to promote recall and restarts the learning and matching process. After that the matcher promoting recall, based on NNge, a nearest-neighbor-like classifier, has been generated, it discovers more relevant correspondences (13% increase for recall). Promoting recall slightly decreases precision, but average f-measure reaches 62%.

Training with similar schemas. Now, let us imagine that the user has some similar schemas than the ones to be matched, for instance other schemas dealing with hotel booking, currency, courses, etc. domains. (S)He/She decides to train on those similar scenarios instead of the whole KB by clicking the choose training scenarios button. (S)He/She then selects the similar scenarios and runs the learning and matching. Another schema matcher has been generated, based on JRip, a propositional rule learner. This matcher achieves an average 66% f-measure. Thus, training the matchers on similar schemas can improve the results.

Providing expert feedback. As the user is not satisfied with these results, (s)he finally provides expert feedback as input. Thus, (s)he selects some discovered correspondences (5% in our experiments, which represents at most 5 correspondences to provide), and validates them by clicking the validate button. These validated correspondences (along with the schemas) are then added to the KB, and are also automatically added to the input expert correspondences panel. The user then clicks the learn and match button. With this new input3, YAM generates the dedicated schema matcher, Bayes Network in this case, in 630 seconds. Since it does not need to focus on the input expert correspondences, this matcher enables an f-measure improvement up to 89%.

Table 1 sums up the results and provides a comparison with other reputed matching tools, COMA++ and Similarity Flooding (SF). We notice that YAM robust (without user input) achieves similar results than those of COMA++ and SF. However, when user spends some time to provide some inputs, YAM is able to strongly improve the matching quality.

Table 1: Average results of the different matchers on the 10 scenarios

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>YAM-robust</td>
<td>52%</td>
<td>65%</td>
<td>58%</td>
</tr>
<tr>
<td>YAM-recall</td>
<td>51%</td>
<td>78%</td>
<td>62%</td>
</tr>
<tr>
<td>YAM-similar</td>
<td>55%</td>
<td>81%</td>
<td>66%</td>
</tr>
<tr>
<td>YAM-feedback</td>
<td>88%</td>
<td>90%</td>
<td>89%</td>
</tr>
<tr>
<td>COMA++</td>
<td>74%</td>
<td>45%</td>
<td>56%</td>
</tr>
<tr>
<td>SF</td>
<td>64%</td>
<td>54%</td>
<td>58%</td>
</tr>
</tbody>
</table>

3. CONCLUSION

In this paper, we have presented YAM, a factory of schema matchers. The main contributions of our work are: (i) implementation of the first tool capable of producing a dedicated schema matcher, and (ii) impact of various user inputs on the quality. By means of demonstration, we will show how YAM generates different schema matchers according to user requirements. More information (scenarios and demo) can be found at http://www.lirmm.fr/~duchateau/yam.

4. REFERENCES


3If there is not enough input expert correspondences to train correctly, YAM adds some randomly training schemas from the KB.