MashupAdvisor: A Recommendation Tool for Mashup Development

Hazem Elmeleegy
Department of Computer Science
Purdue University
West Lafayette, IN 47907
hazem@cs.purdue.edu

Anca Ivan, Rama Akkiraju, Richard Goodwin
Business Service Informatics
IBM T.J. Watson Research Center
Hawthorne, NY 10532
{*ancaivan,akkiraju,rgoodwin*}*@us.ibm.com*

Abstract

Mashup editors, like Yahoo Pipes and IBM Lotus Mashup Maker, allow non-programmer end-users to "mash-up" information sources and services to meet their information needs. However, with the increasing number of services, information sources and complex operations like filtering and joining, even an easy to use editor is not sufficient. MashupAdvisor aims to assist mashup creators to build higher quality mashups in less time. Based on the current state of a mashup, the MashupAdvisor quietly suggests outputs (goals) that the user might want to include in the final mashup. MashupAdvisor exploits a repository of mashups to estimate the popularity of specific outputs, and makes suggestions using the conditional probability that an output will be included, given the current state of the mashup. When a suggestion is accepted, MashupAdvisor uses a semantic matching algorithm and a metric planner to modify the mashup to produce the suggested output. Our prototype was implemented on top of IBM Lotus Mashup Maker and our initial results show that it is effective.

1 Introduction

With Web 2.0 technologies spreading rapidly across the Internet, mashups have recently attracted much attention as a promising approach for ad-hoc information and service integration. HousingMaps.com is an example of a Website that "mashes-up" two other popular Websites: Craig's List and Google Maps. Over 2,000 more online mashups are listed in ProgrammableWeb.com [9], a Website serving as a registry of online mashups and the APIs they use. Yahoo Pipes, Yahoo's mashup editing Website, contains over 20,000 mashups created by Web users. Statistics both from ProgrammableWeb and Yahoo Pipes Websites show that there is a rapidly growing population of users who are both using the existing mashups as well as creating new ones. In addition to public mashups, enterprise users are creating mashups to integrate information from various enterprise sources and applications, often linking in public sources from the Internet, to address a particular business need. Such mashups are referred to as *enterprise mashups*. Building either public and enterprise mashups presents similar challenges.

A key challenge in building a mashup editor for non-programmer users is making the process of mashup creation as simple and "programming-free" as possible. Several mashup editors have already been introduced by the industry, including Yahoo Pipes [10], Google Mashup Editor [4], Microsoft PopFly [7], Intel Mash Maker [6], and IBM QEDWiki [5]. These products allow their users to create mashups without any programming involved, by dragging-and-dropping services, operators, feeds and/or user inputs and visually connecting them. However, the knowledge required is not trivial because users are still expected to know exactly what the mashup inputs and outputs are, and to figure out all the intermediate steps needed to generate the outputs from the inputs. These tasks can be very difficult, especially with the large number of available services and information sources, and complex filtering and merging operations. Moreover, many users may have a general idea of what they are trying to achieve, but not know the specifics of what they want or what is possible.

This paper presents a recommendation tool, called MashupAdvisor, that addresses these issues by providing design-time assistance to the user. MashupAdvisor relies on a repository of mashups developed by a community of users to generate recommendations. For example, if the user has included in the mashup a business news source, MashupAdvisor might offer to plot company stock prices, map store locations or display the founder's bio of companies featured in an article. If the user accepts a suggestion, MashupAdvisor automatically selects and configures the required inputs and adds the required filters, joins and service calls to the mashup to produce the desired result. A

978-0-7695-3310-0/08 $25.00 © 2008 IEEE
DOI 10.1109/ICWS.2008.128

user might accept a suggestion because it was their original intent, in which case MashupAdvisor saves them time, or they might accept it because it provides useful information that they had not thought of or did not know was possible. The recommendations are computed based on a probabilistic model that ranks the mashup outputs based on their popularity (i.e., usage in the mashup repository). Once the user selects an output, AI metric planning techniques are used to find the maximum utility plan to generate the desired output. The utility function used by the AI planner is directly proportional to the popularity of the plan steps as inferred from the mashup repository. In addition to modeling the popularity, MashupAdvisor can take the semantic similarity between the attribute names into account when making the recommendations.

The MashupAdvisor has three key benefits: (1) saving the development time, (2) recommending higher quality plans for user outputs, and (3) recommending interesting and relevant outputs to the user that he/she would not have thought of independently.

The rest of the paper is organized as follows. Section 2 describes a motivating scenario. Section 3 gives a detailed description of the architecture and its components. In Section 4, the experiments illustrate the efficiency and effectiveness of the MashupAdvisor. Section 5 discusses the related work. Section 6 finally concludes the paper and gives suggestions for future work.

2 Motivating Scenario

The following scenario illustrates how MashupAdvisor can be used to solve a problem and the value it brings to the mashup developer. In this scenario, Bob is an HR employee and he is assigned the task of building a Web application that enables other employees to quickly find nearby businesses such as restaurants, gas stations, banks, etc. Bob uses a mashup editor to build the necessary mashup. The editor has a repository of services, RSS feeds, user inputs, and special operators such as "aggregate feeds", "filter feed", and "sort feed".

Initially, Bob uses the mashup editor without the recommendation engine.

1. Bob creates a user input ("email"), where employees should enter the email address.

2. Bob selects the *BluePages* service and connects its "email" parameter to the "email" user input. The *BluePages* service outputs detailed information about a person, given the email address.

3. Bob selects the *Yahoo Local Search* service (which takes the street address as input), extracts the city and state information from the composite XML output of the *BluePages* service using two XPath queries, and finally connects the extracted city and state information to the "city" and "state" parameters of the *Yahoo Local Search* service.

4. Bob repeats step 3 several times; each time Bob enters in the "keyword" parameter of *Yahoo Local Search* service keywords such as "restaurant", "gas station", "bank", etc.

The outputs of the *Yahoo Local Search* services constitute the contents of the desired Web application. Note that the output of the *BluePages* service does not contain a "street" element, and therefore Bob used the *Yahoo Local Search* service to find nearby business near the employee's city as opposed to his exact street address. Thus, the results of *Yahoo Local Search* service may not be the closest to the employee's location.

Later, Bob re-creates the mashup with the help of MashupAdvisor.

1. Bob creates a user input ("email"), where employees should enter their email addresses.

2. MashupAdvisor recommends a list of outputs to be added to Bob's mashup. In order, they are: *BluePages* output, *Yahoo Local Search* output, and *Facebook Search* output.

3. Bob selects the *Yahoo Local Search* output from the MashupAdvisor recommendations.

4. Mashup Advisor computes the best plan to generate the *Yahoo Local Search* output given the email address already available in Bob's mashup. The computed plan connects the "email" user input to the "email" parameter of the IBM *BluePages* service. Then, it extracts the "zipcode" information from the composite XML output of the *BluePages* service using an XPath query and connects it to the "zipcode" parameter of the *Yahoo Local Search* service (which takes the zipcode as input).

5. Bob makes several copies of the *Yahoo Local Search* service; each time, Bob enters in the "keyword" parameter of each copy keywords such as "restaurant", "gas station", "bank", etc.

6. MashupAdvisor recommends a new list of outputs to be added to Bob's mashup. In order, they are: *Yahoo Traffic Report* output, *Yahoo Weather* output, and *Facebook Search* output.

7. Bob selects the *Yahoo Traffic Report* output from the MashupAdvisor recommendations.

8. MashupAdvisor computes the best plan to generate the *Yahoo Traffic Report* output given the information already present in Bob's mashup. The computed plan connects the "zipcode" information extracted from the *BluePages* service output to the "zip" parameter of the *Yahoo Traffic Report* service.

In this scenario, the outputs of the *Yahoo Local Search* service copies along with the output of the *Yahoo Traffic Report* constitute the contents of the desired Web application.

The recommendations offered by MashupAdvisor were decided based upon the information already available in Bob's mashup in addition to a repository of mashups developed by a community of users. The key benefits of the MashupAdvisor are:

Shorter development time: Bob did not need to search through the repository of services to find the services required to build his mashup. He did not need to ensure that they can be connected together, nor did he have to find the best way to connect them (including browsing through the composite XML outputs of some of the services to generate the XPath queries necessary for the extraction of certain fields).

Higher quality plans: The plan recommended by the MashupAdvisor to generate the *Yahoo Local Search* output used a version of the *Yahoo Local Search* service that takes the zip code as input rather than the street address. This resulted in more accurate results compared to the plan Bob used when MashupAdvisor was turned off, where the search was performed using the city and state information only.

Unthought-of-yet-interesting outputs: Bob did not think of adding a traffic report to his mashup before. But, when MashupAdvisor recommended it a possible output, Bob realized that it would be useful for the employees to be aware of the traffic report while deciding which restaurant to go to.

3 Mashup Advisor

MashupAdvisor is a recommendation tool which provides design-time assistance to non-programmer mashup creators. It can automatically provide recommendations when a change occurs to the partial mashup, or it can be invoked upon user request. The recommendation process takes place in two steps: (1) the MashupAdvisor generates a ranked list of recommended outputs that can be added to the user's mashup, and (2) when the user selects a recommended output, the MashupAdvisor computes the best plan to generate the selected output given the partial mashup. The following subsections describe the overall architecture of MashupAdvisor, and explain the details of its components.

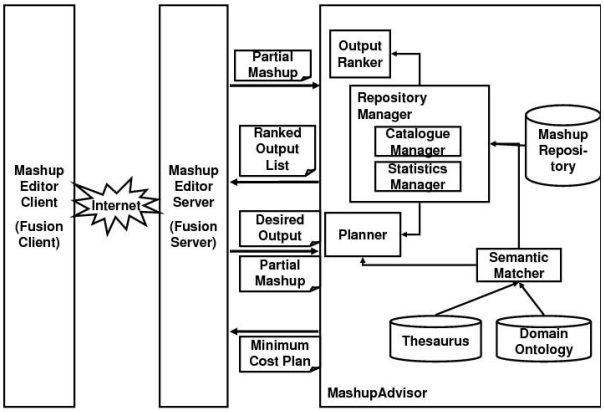

Figure 1. System Architecture of MashupAdvisor

3.1 System Architecture

The MashupAdvisor has four components (Figure 1): a *repository manager*, a *semantic matcher*, an *output ranker*, and a *planner*. The *repository manager* answers requests regarding the repository of mashups. It has two subcomponents: a *catalog manager*, which keeps track of all the mashups, services, and service outputs in the repository; and a *statistics manager* which determines the probability distributions for the occurrence of different repository concepts and their co-occurrence patterns. The *semantic matcher* computes the semantic similarity score for any pair of concepts based on domain independent thesaurus (e.g. wordnet) and domain dependent ontologies. The *output ranker* and the *planner* components interface with the client. They use the repository manager and the semantic matcher to generate recommendations. In particular, the output ranker is responsible for the first step in the recommendation process (i.e. generating a ranked list of outputs), while the planner is responsible for the second step (i.e. computing the best plan to generate the selected output given the partial mashup).

3.2 Repository Manager

The *repository manager* analyzes the repository of mashups and computes statistical information for use by other components in the system. A mashup is modeled as a composition of services and information sources. An information source can be an online feed or a user supplied input. A service is defined by its name, inputs and outputs. For each service, the inputs are divided into formal and ac-

tual inputs. Formal inputs are the identifiers used in the service definition, while actual inputs refer to sources or other service outputs that are linked to the service's formal inputs in a specific mashup. A concept A is denoted A^f, A^a, or A^o if it is used as a formal input, an actual input, or an output respectively. If A represents a user supplied input, it is denoted A^u. The notation $A \rightarrow B$ denotes that the concept A is linked to concept B, where A would be the actual input and B would be the formal input.

The statistics manager computes statistics about the usage of concepts within the repository:

- $P(A^a)$: probability that concept A is used as an actual input.
- $P(A^o)$: probability that concept A is used as an output.
- $P(A^a|B^o)$: probability that concept A is used as an actual input, given that concept B is used as an output.
- $P(A^o|B^a)$: probability that concept A is used as an output, given that concept B is used as an actual input.
- $P(A \rightarrow B^f|A^o \bigcup A^u)$: probability that concept A is linked to concept B, which is a formal input, given that concept A is used as an output or a user supplied input

The first two non-conditional probabilities are computed for each concept A by counting the number of mashups having A^a or A^o, and then dividing those counts by the total number of mashups. The next three conditional probabilities are computed for every pair of concepts A and B, by counting the number of mashups having A^a and B^o, A^o and B^a, or $A^o \rightarrow B^f$/$A^u \rightarrow B^f$ for each of the three probabilities respectively. These three counts are then divided by the count of mashups having B^o, B^a, or A^o/A^u respectively.

The statistics manager can be configured to take semantics into account when calculating the probabilities. Instead of only counting mashups containing the exact match of a concept A, the statistics manager can also count the mashups having a semantically similar concept A'. However, their count will be weighted by the similarity score between A and A', as given by the *semantic matcher*. If the probability calculation involves two concepts A and B, then the count of mashups having similar concepts A' and B' will be weighted by the product of similarity scores between A and A' and between B and B'.

This is best explained by the following example.

Consider a repository of 10 mashups, where three mashups have the concept "zip" used as an actual input, and two other mashups have the concept "postalCode" used as an actual input as well. If semantics is taken into account and the semantic similarity between "zip" and "postalCode" is 0.7, then the probability that "zip" is used as an actual input is given by

$$P(\text{“}zip\text{”}^a) = \frac{3 \times 1 + 2 \times 0.7}{10} = 0.44$$

where 3 and 2 are the counts of the mashups having "zip" and "postalCode" used as an actual input respectively, while 1 and 0.7 are the assigned weights.

3.3 Semantic Matcher

The Semantic matcher [12] is used by both the *output ranker* and the *planner* to find the "best" matches for a given concept by computing a "similarity score". The similarity score between two concepts A and B is given by Formula 1:

$$Score(A, B) = synNum/max(len(A), len(B)) \quad (1)$$

where $synNum$ is the number of matching words (i.e., the tokens that are synonyms), and $len(X)$ is the number of tokens in the tag X. The score captures the closeness of the two tags. For example, the similarity score between *CustomerCare* and *ClientSearch* is 0.5, because *Customer* and *Client* are synonyms, but *Care* and *Search* are not.

3.4 Output Ranker

The responsibility of the *output ranker* is threefold: (1) it identifies a set of candidate outputs that can be added to the user's existing partial mashup, (2) it assigns a relevance score to each candidate output, and (3) it ranks the candidate outputs based on their scores.

The output ranker selects the candidate outputs from the mashup repository, excluding the concepts appearing in the user's partial mashup either in the form of service outputs or in the form of direct user inputs. For each candidate output, the score is an estimate of the conditional probability that the output will be included in the final mashup, given the existing partial mashup. Consequently, if A is the candidate output, while B_1, B_2,..., B_n are the mashup concepts of the partial mashup, then a suitable scoring function for the output ranker is given by the probability that the concept for the candidate output is used as an output given that *any* of the concepts of the sources and service outputs in the partial mashup are used as actual inputs (Formula 2). Note that while Formula 2 would ideally use the joint probabilities, this is not practical both because the repository may not include enough examples, and the need to calculate the probabilities in near real time.

$$S_{or}(A) = P(A^o|\bigcup_{i=1}^{n} B_i^a) \quad (2)$$

The calculation of the scoring function was simplified with the assumption that the event that a concept A appears as an actual input in a specific mashup is independent from the event that another concept B appears as an actual input in the same mashup. There are no assumptions regarding the relationship between inputs and outputs, even though

outputs would normally depend on which inputs exist in the mashup. Formula 3 is obtained by applying Bayes rule to Formula 2.

$$S_{or}(A) = \frac{P(A^o)P(\bigcup_{i=1}^{n} B_i^a|A^o)}{P(\bigcup_{i=1}^{n} B_i^a)} \quad (3)$$

Because of the inclusion-exclusion principle:

$$P(\bigcup_{i=1}^{n} B_i^a|A^o) = \sum_{i=1}^{n} P(B_i^a|A^o) - \sum_{1\le i_1<i_2\le n} P(B_{i_1}^a \bigcap B_{i_2}^a|A^o) + \ldots + P(\bigcap_{i=1}^{n} B_i^a|A^o) \quad (4)$$

Many of the probabilities in Formula 4 cannot be obtained from the statistics manager, and therefore they require scanning the repository each time an output's score is calculated. However, taking the independence assumption into consideration, 4 can be re-written as follows:

$$P(\bigcup_{i=1}^{n} B_i^a|A^o) = \sum_{i=1}^{n} P(B_i^a|A^o) - \sum_{1\le i_1<i_2\le n} P(B_{i_1}^a|A^o)P(B_{i_2}^a|A^o) + \ldots + \prod_{i=1}^{n} P(B_i^a|A^o) \quad (5)$$

Similarly,

$$P(\bigcup_{i=1}^{n} B_i^a) = \sum_{i=1}^{n} P(B_i^a) - \sum_{1\le i_1<i_2\le n} P(B_{i_1}^a)P(B_{i_2}^a) + \ldots + \prod_{i=1}^{n} P(B_i^a) \quad (6)$$

Note that all the probabilities used in Formulas 5 and 6 are known to the statistics manager. Additionally, since $P(A^o)$ is known to the statistics manager; thus, the output ranker can avoid performing expensive scans of the repository during the calculation of $S_{or}(A)$, as would have been the case had the independence assumption not been made.

One problem is that the number of concepts in Formulas 5 and 6 is exponential in n, where n is the number of concepts in the user's partial mashup. In order to reduce the computation time, the formulas take only into consideration the most relevant n_{max} concepts. The relevance of a concept B in the partial mashup to the candidate output A is defined by $P(A^o|B^a)$, which again can be obtained from the statistics manager.

3.5 Planner

Once the output ranker recommends a number of concepts as relevant outputs, the user may select any concept and the MashupAdvisor computes the best combination of services that will output the selected concept. The MashupAdvisor uses an AI planner [2] in order to compute the service composition with the highest utility that will achieve a goal (selected concept) given the state of the world (partial mashup concepts). The utility function used to guide the search process depends on the plan popularity and the quality of semantic matching in the plan. The plan popularity score S_p is the product of all probabilities of the form $P(A \rightarrow B^f|A^o \bigcup A^u)$ for every binding $A \rightarrow B$ existing in the plan. The plan semantic score S_s is the sum of all similarity matching scores between attributes A and B for each binding $A \rightarrow B$ existing in the plan. The plan overall score is $S = w_p S_p + w_s S_s$, where w_p and w_s are configurable weights.

4 Experiments and Results

The goal of the experiments is to show that both the output ranker and the planner provide high-quality information to the users, and scale with the number of mashups and services in the repository.

Data sets. In the absence of a real repository of mashups that could be consumed by the MashupAdvisor, the experiments were ran on a automatically generated data set. The data set simulates the repository of mashups from the ProgrammableWeb.com [9], and was generated by following an 80/20 geometric distribution for both the service popularity and the mashup size. Figure 2 shows that most mashups contain a small number of services (350 mashups contains less than 5 services), and the largest mashup created by a user contains 22 services. Figure 3 shows that

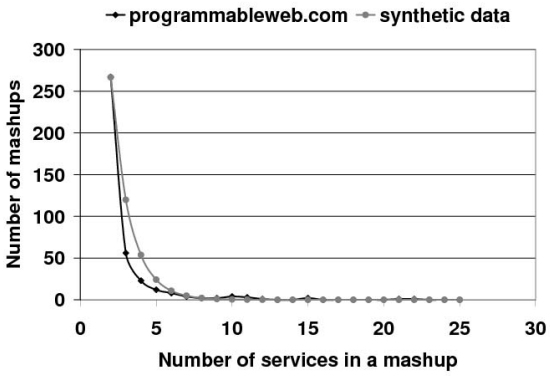

Figure 2. Distribution of Mashup size from the Programmable Web

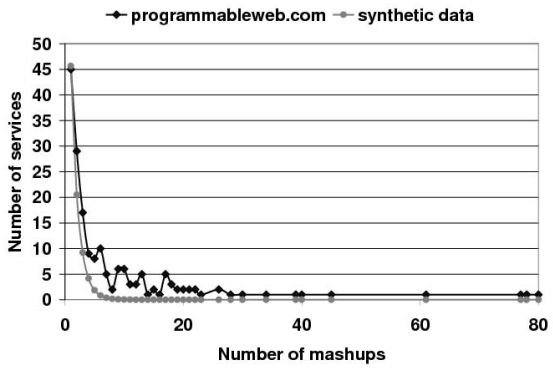

Figure 3. Distribution of Service popularity from the Programmable Web

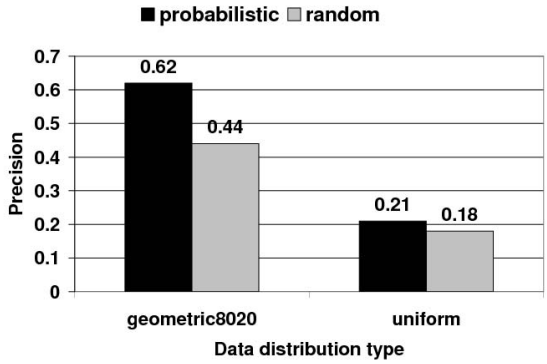

Figure 4. Comparison of output ranker quality

a large number of services are used in a small number of mashups (almost 100 services are used in at most 3 mashups), while a very small number of services are very popular (less than 10 services). In addition to this data set, the MashupAdvisor performance was also measured on a data set created by following a uniform distribution. The comparison between the two data sets is showing how the quality of results depends on the initial data set.

Experimental Setup. The measures for both the output ranker and the planner were taken by running simulations on the *geometric80/20* and *uniform* data sets. Each data set contains was divided into two sets, one to represent the repository and one to use as tests. For each test mashup, the intermediary outputs were used as inputs to the recommendation system. The quality of the output ranker is measured as:

$$hit\ ratio = number\ of\ hits / total\ number\ of\ queries$$

. A query to the output ranker is considered a *hit* if the chosen output is in the list of outputs recommended by the output ranker. To test the planner, the experiments use the inputs and final outputs and generate a plan. The plan efficiency is evaluated by measuring the time taken to generate the plan and the number of steps in the plan.

All experiments were ran on a IBM Thinkpad T60 with 2.33GHz Intel Centrino Duo processor and 3GB of RAM.

4.1 Output ranker quality

The quality of output ranker was assessed by comparing the quality of the recommended outputs against the quality given by a random recommender. By definition, a random recommender chooses a number of outputs at random and suggests them as recommendations. Both the MashupAdvisor and the random recommender were ran on the two data sets (geometric80/20 and uniform). Figure 4 shows that the MashupAdvisor is marginally better than the random recommender when used on the uniform data set. However, the improvement is significant in the case of the geometric80/20 data set, because the output ranker is able to re-use the user experience accumulated in the repository (the experience is captured as service/concept popularity).

4.2 Output ranker execution performance

As described in Section 3.4, the output ranker spends most of the time computing Formula 6, which is exponential in the number of inputs. In order to prevent the response time from increasing and making the output ranker unusable, the output ranker allows setting a maximum number of inputs that will be used during the computation. Figure 5 shows how the running time increases until the number of inputs reaches the maximum limit, and then flattens out. Overall, in a repository with 10,000 services and 10,000 mashups, the output ranker takes less than 100ms to respond to a user request, if the limit on relevant inputs is 12.

4.3 Planner quality

In order to evaluate the quality of the plans created by the planning module, the following experiment was conducted. For each test mashup, extract half of the mashup outputs and the final output. For each output and the user

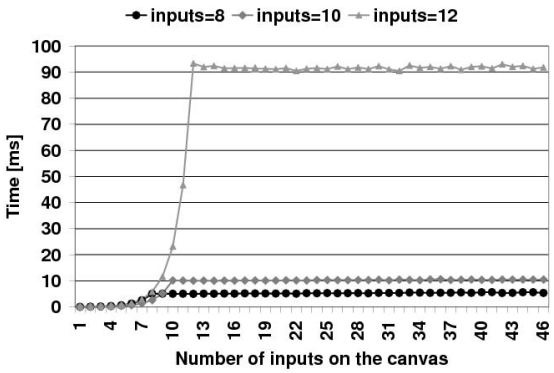

Figure 5. Output ranker performance

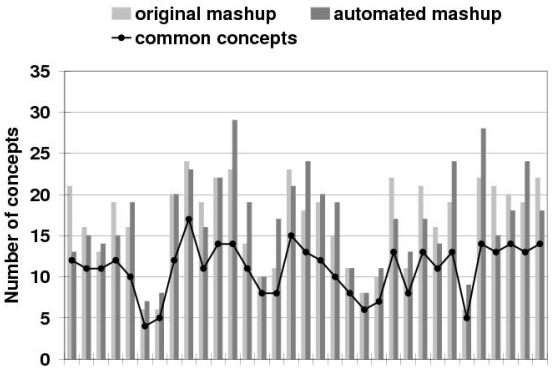

Figure 6. Planner quality - concept coverage

inputs, ask the planner to find a plan that finds that same output given the user inputs and incrementally build a mashup. The first conclusion is that the mashups created through this simulation are almost the same size as the original mashups (the biggest difference in size was 4 services). In particular, the original mashup was bigger than the automated mashup in 40% of case, it was smaller in 40% of cases, and had the same size in 20% of cases. Figure 6 illustrates two important insights: (1) automated mashups provide almost the same outputs as the original mashups (the number of common outputs is very high in all cases), and (2) automated mashups also provide new and relevant outputs that the user might not have considered otherwise, because these outputs were computed based on their popularity (other users considered that these outputs were relevant).

4.4 Planner execution performance

The goal of this experiment is to verify whether the planner execution time scales with the number of services in the repository. The evaluation was done by running multiple queries against repositories of different sizes (1000 to 8000 services). The limit on the planning search space was set to 50000 states. Figure 7 shows the running time for the planner for a series of queries. For easy reading, all planning times were displayed in increasing order. For most queries, the planner finds plans in less than 50 seconds. For repositories of medium to large size (5000-8000 services), the planner takes 2-3 minutes. Clearly, most users would find a delay of minutes unacceptable. One of the next research directions is to optimize the planner or create a customized planner instead of using a generic AI planner.

5 Related Work

In the last couple of years, there has been a lot of activity in the industry to build mashup development tools. Some of these tools are Yahoo Pipes [10], Google Mashup Editor [4], Microsoft PopFly [7], Intel Mash Maker [6], and IBM QEDWiki [5]. Except for Intel Mash Maker, all these tools are editors, which are focused on allowing users to visually create mashups by connecting services and sources together in various ways. MashupAdvisor can be easily integrated with any of such tools. Similar to MashupAdvisor, Mash Maker, which is implemented as an extension to the Web browser, also provides recommendations to the users. In particular, Mash Maker's recommendations are based on

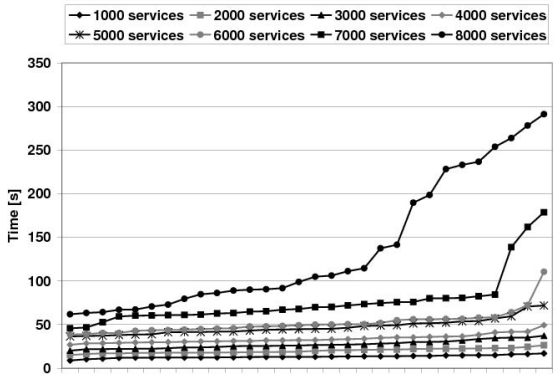

Figure 7. Planner performance

the content of the pages users are currently browsing. However, unlike MashupAdvisor's recommendations, they do not take into account the mashups, which other users have previously created. Moreover, the additional information that Mash Maker may recommend to the Web user is information that can be reached through a single step, rather than through a complete plan of any arbitrary size, as is the case with MashupAdvisor.

Another related body of work concerns the problem of service composition. Many approaches proposed in the literature use AI planning techniques to solve this problem. These approaches can be compared to the planner component of MashupAdvisor. A detailed survey of such approaches is presented in [8]. However, none of the surveyed approaches attempted to model and incorporate the popularity of plans and subplans within the utility function, which guides the search for the optimal plan. Furthermore, MashupAdvisor can handle RESTful services and data feeds in addition to web services.

Several research efforts are directed towards the problem of the semantic matching of Web services. The systems described in [3], [11], and [2] are examples of such efforts. The main goal of this body of work is to go beyond the exact matching of the inputs and outputs during service matching, and instead rely on the semantic similarity. MashupAdvisor builds upon the results of this body of work, as evident from the incorporation of the semantic matcher component.

MashupAdvisor can be considered to belong to the large area of recommender systems. A survey of this area is given in [1], where recommender systems are classified as content-based, collaborative, or hybrid systems. MashupAdvisor can be classified as a hybrid recommender system, since it depends on the popularity of the recommended outputs and plans among a collection of users, in addition to the semantic similarity between the recommended elements and the elements already included in the user's mashup. Moreover, most of the collaborative or hybrid recommender systems assume the availability of a matrix showing the rating of different users to the different possible recommendations. However, the information available to MashupAdvisor has a quite different structure. In particular, the collection of mashups, where each mashup is essentially a graph connecting information sources and services together, required MashupAdvisor to employ techniques that are significantly different from the ones used by standard collaborative recommender systems.

6 Conclusions and Future Work

In this paper we have presented MashupAdvisor, a system for assisting mashup composers by suggesting relevant outputs to augment partially constructed mashups. The system aims to reduce the amount of time needed to create mashups, to improve the efficiency of the resulting mashups and to improve the quality of the resulting mashups. The studies that we have done support these goals. There is however some work to be done on optimizing the planner for large service repositories. We also plan to run user studies to validate the results.

References

[1] G. Adomavicius and A. Tuzhilin. Towards the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. In *IEEE Transactions on Knowledge and Data Engineering*, volume 17, June 2005.

[2] R. Akkiraju, B. Srivastava, A. Ivan, R. Goodwin, and T. Syeda-Mahmood. Semaplan: Combining planning with semantic matching to achieve web service composition. In *Proceedings of ICWS*, 2006.

[3] X. Dong, A. Havey, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proceedings of VLDB*, 2004.

[4] Google. *Google Mashup Editor*, April 2008. http://code.google.com/gme/.

[5] IBM. *IBM Mashup Starter Kit*, October 2007. http://www.alphaworks.ibm.com/tech/ibmmsk.

[6] Intel. *Intel Mash Maker*, April 2008. http://mashmaker.intel.com/.

[7] Microsoft. *Popfly*, April 2008. http://www.popfly.com/Overview/.

[8] J. Peer. Web service composition as ai planning - a survey. Technical report, 2005.

[9] ProgrammableWeb.com. *ProgrammableWeb.com*, May 2007. http://www.programmableweb.com.

[10] ProgrammableWeb.com. *Yahoo Pipes - A case study of using the Regex Module*, September 2007. http://theytookmystapler.blogspot.com/2007/09/yahoo-pipes-case-study-of-using-regex.html.

[11] E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of web services using semantic descriptions. In *1st Workshop on Web Services: Modeling, Architecture and Infrastructure*, pages 17–24. ICEIS Press, April 2003.

[12] T. Syeda-Mahmood, G. Shah, R. Akkiraju, A. Ivan, and R. Goodwin. Searching service repositories by combining semantic and ontological matching. In *Proceedings of ICWS2005*, 2005.