An Effcient and Scalable Ranking Technique for Mashups Involving RSS Data Sources

Article *in* Journal of Network and Computer Applications · March 2014

DOI: 10.1016/j.jnca.2013.06.004

CITATIONS
2

READS
10

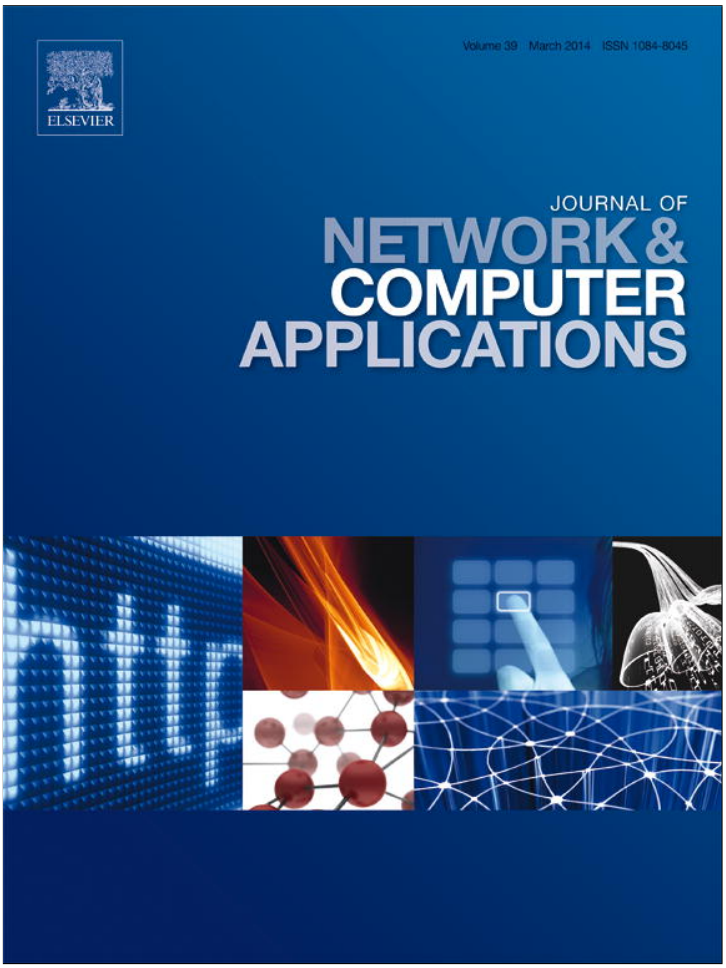
4 authors:

Osama Al-Haj Hassan
Isra University, Jordan
23 PUBLICATIONS **197** CITATIONS
SEE PROFILE

Thamer Al-Rousan
Isra University, Jordan
39 PUBLICATIONS **100** CITATIONS
SEE PROFILE

Anas abu taleb
Princess Sumaya University for Technology
25 PUBLICATIONS **180** CITATIONS
SEE PROFILE

Adi A. Maaita
Isra University, Jordan
20 PUBLICATIONS **40** CITATIONS
SEE PROFILE



Journal of Network and Computer Applications 39 (2014) 179–190

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

An efficient and scalable ranking technique for mashups involving RSS data sources

Osama Al-Haj Hassan *, Thamer Al-Rousan [1], Anas Abu Taleb [1], Adi Maaita [1]

Isra University, 11622 Amman, Jordan

ARTICLE INFO

Article history:
Received 2 October 2012
Received in revised form
5 April 2013
Accepted 15 June 2013
Available online 27 July 2013

Keywords:
Mashup
Ranking
Personalization
Vector space model
Web service
Recommendation

ABSTRACT

Mashups are key category of Web 2.0 personalized applications. Due to personalization property of Web 2.0 applications, number of mashups hosted by a mashup platform is increasing. End-users are overwhelmed by the increasing number of mashups. Therefore, they cannot easily find mashups of their interest. In this paper, we propose a novel mashup ranking technique based on the popular Vector Space Model (VSM) for mashups that use RSS feeds as data sources. Mashups that are ranked higher would be more interesting to end-users. In order to evaluate our mashup ranking technique, we implement it in a prototype where end-users select mashups that they consider interesting. We implicitly collect the end-user mashup selections and record the outcome of our ranking technique, and then we analyze them. Recorded R-Precision value in our technique is on an average 30% higher than R-Precision value in binary ranking technique which shows an improvement in capturing mashups that resemble end-user interest. In our design, we make sure our mashup ranking technique scales well to increasing number of mashups.

1. Introduction

The main interest of Internet users is shifting towards Web 2.0 (Murugesan, 2007) applications. One of the main features of Web 2.0 that attracts Internet users is personalization. Web 2.0 personalized applications provide tools customized per end-user specific needs.

One of the icons of Web 2.0 personalized applications is mashups. Mashups enable end-users to remix feeds and further process them forming their own Web services (Beemer and Gregg, 2009). This is why mashups in concept are Web services created by end-users. Mashups allow end-users to query the Web in a more personalized, flexible, and effective way. They enable end-users to extract data from multiple data sources and then combine data and apply further processing and refinement to it. Each end-user can create his/her own set of mashups; this is why mashups aid towards enhancing the world of personalization. A mashup example is found in Fig. 1. We state here that our work only considers RSS feeds as data sources for mashups.

Mashup platforms are those applications that offer a graphical user interface through which the end-user can perform the following tasks. First, the end-user can design his/her own mashups. Second, the end-user can browse other end-user mashups. Third, the end-user can execute mashups of his/her own or other end-users'publicly available mashups. Examples of mashup platforms on the Web are Yahoo Pipes (Yahoo Inc, 2007) and Intel MashMaker (Intel Corp, 2007).

Another application that uses mashups in its core is mashup reader. Mashup readers are applications that allow end-users to design their own mashups and at the same time search and subscribe for mashups of their interest. Accordingly, end-users are able to follow updates in the result of execution of their favorite mashups.

One drawback of mashup platforms and mashup readers is that they do not provide a ranking mechanism for mashups. In mashup platforms, suppose the end-user is interested in browsing available mashups related to sport news of '*LA Lakers*' and '*Kobe Briant*'. The result of such browse request might involve large number of mashups and that makes it difficult for the end-user to select best mashups that satisfy his/her request. Therefore, it is important for end-users to get the result of their search request ordered based on mashup ranking which is unfortunately not provided by mashup platforms. In mashup readers, the end-user may care about following periodical updates about papers published by '*Springer*' and '*Elsevier*' only if they fulfill certain criterion such as having '*Web Services*' topic and concerning '*Quality of Service*'. Similarly, large number of mashups might satisfy this query. Consequently, mashup ranking is a necessity for end-users so that they can subscribe to mashups that conform to their demands.

* Corresponding author. Tel.: +96 2799834001; fax: +96 264711505.
E-mail addresses: osamahajhassan@hotmail.com, osama.haj@ipu.edu.jo (O.-H. Hassan), thamer.rousan@ipu.edu.jo (T. Al-Rousan), anas.taleb@ipu.edu.jo (A.A. Taleb), adimaaita@ipu.edu.jo (A. Maaita).
[1] Fax: +96 264711505.

1084-8045/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jnca.2013.06.004

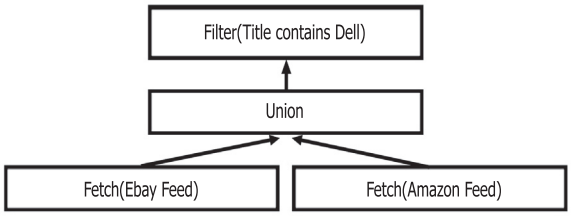


Fig. 1. Mashup example.

Some of the current ranking techniques in recommendation systems depend on keywords and URLs as the base of their techniques. Examples of such techniques are Chen et al. (2007), Cingil et al. (2000), Liu et al. (2002), Mobasher et al. (1999), Mobasher et al. (2001), and Samper et al. (2008). Since the previous techniques rely on keywords and URLs, they would not be suitable for mashups. This is because mashups are represented as trees of operators arranged in a specific logical order. Therefore, the previously mentioned mashup ranking techniques in recommendation systems do not suit mashups because they do not take into consideration the mashup structure and relationship between operators.

The previous paragraphs show the resemblance between mashups and Web services. Hence, it is logical to ask whether Web services ranking techniques work for mashups. Existing ranking techniques in Web services such as Caverlee et al. (2004), Dong et al. (2004), Paolucci et al. (2002), and Skoutas et al. (2007) cannot be directly applied to mashups. This is justified by the fact that mashups are created by end-users and this implies that the number of mashups hosted by a mashup platform is potentially higher than the number of Web services hosted by a Web service portal. Therefore, Web services ranking techniques cannot cope with the scalability requirement of mashup platforms.

1.1. Contribution

Mashup centric platforms lack efficient mashup ranking capabilities. Therefore, we focus on efficiency of mashup ranking. Our paper contribution is laid out in the following points.

- We design a novel ranking mechanism for mashups; this technique utilizes the popular Vector Space Model (VSM).
- We design our mashup ranking technique so that it handles mashup platforms scalability requirements.
- We evaluate our mashup ranking technique by implementing it in a simple prototype where end-users are required to select mashups of their interest.

The rest of the paper is organized as follows. First, we discuss literature related to our work in Section 2. Then, in Section 3 we overview our ranking technique and point out challenges we faced during its design. After that, in Section 4 we show how mashups are represented, organized, and accessed in our system. Following that, in Section 5, we discuss our mashup ranking technique in details. Next to that, in Section 6 we discuss complexity related issues in our ranking technique. Finally, in Section 7 we thoroughly evaluate our mashup ranking technique and we conclude our work in Section 8.

2. Related work

Our work considers ranking for mashups so that a mashup platform can provide end-users with recommendation of mashups they might be interested in. Therefore, we classify this section into four subsections, namely, recommendation systems, ranking for Web services, ranking for mashups, and mashup platforms.

2.1. Recommendation systems

The main theme of Web 2.0 applications is personalization which is achieved by narrowing the huge window of information available in the Web by only presenting the information that end-users actually care about. Therefore, many research papers target personalization topic (Cingil et al., 2000; Liu et al., 2002; Mobasher et al., 1999, 2001; Datta et al., 2001; Mobasher et al., 2000; Shahabi and Chen, 2003). One of the main topics that involve personalization is recommendation systems and we point out that one of the main features of a Web 2.0 application is to provide a recommendation mechanism through which the system can provide the end-user with objects that closely resemble his/her interest. A recommendation technique can be designed using the means of collaborative filtering, content based filtering, defining end-user profiles, tracing end-user search activity, and by using semantic Web principles such as meta data and taxonomies. Our work adopts user profiles as the basis of its recommendation technique. CRESDU (Chen et al., 2007) is a recommendation system with the purpose of delivering RSS advertisements to end-users who are more likely to be interested in them. CRESDU builds user profile based on end-user client-side private data. Cingil et al. (2000) store the click stream of end-users in log files in order to build a profile for each end-user. This profile is compared with end-user behavior for the purpose of generating recommendations for him/her. Liu et al. (2002) use category profiles which can be compared to end-user navigation history. Accordingly, the end-user query can be transformed to a more meaningful one based on comparison with category profiles. Mobasher et al. (1999) use URL clustering approach to generate recommendations. Their work classifies URLs into clusters and compares the URLs in end-user browsing activity with URL clusters in order to produce recommendations. Mobasher et al. (2001) provide real time recommendations for end-users by detecting patterns in end-user Web activity using association rules. NectaRSS (Samper et al., 2008) is a feed reader that utilizes user profiles as means of presenting the end-user with RSS items he/she might be interested in. Our work is similar to NectaRSS in terms of using Vector Space Model in its ranking technique. However, the technique of applying Vector Space Model on feeds (such as in NectaRSS case) is different than applying the same technique on mashups. On one hand, in feed ranking case, the technique mainly depends on matching the content of feeds (Exact or Partial) with the criterion specified by the end-user. On the other hand, mashups go beyond being a source of information (Feed); they are designed as trees which are built on top of feeds and they represent a logical order of executing operators. The relationship between subtrees across mashups and the logical ordering of operators are information that can be exploited towards a ranking technique that best suits mashups. Most of the previous recommendation systems rely on keywords and URLs as their core. Mashups on the other hand are built as subtrees of operator execution, and therefore, a ranking process for mashups should take relationships between operators into account. Accordingly, the previous techniques cannot be directly applied to mashup ranking. As a result, a careful consideration for the specifics of mashup platforms is needed when designing a recommendation technique for mashups.

2.2. Ranking for web services

BASIL (Caverlee et al., 2004) is a ranking technique for Web services. The technique assumes that a source Web service is looking for a similar target Web service. The similarity between

the source and target Web services is based on similarity between summaries of both Web services. The key feature for BASIL is that the similarity process is biased towards the source service because summary of target Web service is estimated based on the summary of the source Web service. Our ranking technique is different because we aim towards finding similarity between the end-user interest and mashups while BASIL focuses on similarity between two Web services. Dong et al. (2004) use semantics of Web service description and compare it with the search keyword provided by the end-user, this would deliver to the end-user Web services that reflect his/her interest. Paolucci et al. (2002) find similarity between Web service output and service request parameters such that four types of similarity are defined, namely, exact, plug-in, subsumes, and fail. Those types of similarity are decided based on the level of subsumption between service and request parameters. In the previous work, an ontology is used for classifying Web services into classes. This way, if a Web service and a service request belong to the same class, the relationship between them is considered 'exact'. Skoutas et al. (2007) propose a ranking technique that is based on similarity between the end-user request and Web service advertisements. The similarity is performed based on parameters of request and input/output parameters of Web services. Their ranking technique also utilizes a domain ontology that helps in finding the degree of similarity between request and Web service. Our work is different than the previously mentioned work in Web service ranking in two points. First, the previous works concentrate on Web services functionality and input/output parameters in their ranking techniques. However, in our ranking technique, we take into consideration mashup structure and relationships between operators. Second, because of personalization property of mashups, the number of mashups hosted by a mashup platform is higher than the number of Web services in a Web service portal. Therefore, the scalability and efficiency requirements in our case are more stringent than Web services case. Therefore, we utilize an indexing data structure that increases the efficiency of our ranking technique. Such efficiency enhancements are missing from the previously mentioned works in Web service ranking.

2.3. Ranking for mashups

Although mashups research area is still new, few research attempts target ranking of mashup. A technique that helps end-users build their own mashups is proposed in Greenshpan et al. (2009) and Abiteboul et al. (2009). Basically, the end-user selects initial components for his/her mashups. Then, the proposed system uses the initial end-user selection to look for other components to complete end-user mashup. Several components can be candidates for completion, so, this existing work proposes an algorithm to present to the end-user the 'k' highest ranked candidates for completion. Ranking quantity is composed of three parts. First, an initial importance of components. Second, how frequent a component is used in other end-users mashups. Third, an inheritance importance which works on the idea that if a given component has a certain importance value and it is used in another subtree, then the subtree takes the importance of the used component. This existing work focuses in helping end-users build their own mashups while our work focuses on helping end-users search for mashups of their interest. Pietschmann et al. (2011) take care of enriching mashups by adding context aware semantics. Therefore, their approach helps end-users in completing composition of their mashups by providing them with ranked templates that semantically match the mashup they are trying to build. Ranking in tagged maps is proposed in Zhang et al. (2010). This existing work focuses on maps which contain tags (such as in Google maps). It proposes a system through which end-users can find places on maps that are relevant to their query. This existing work takes care of efficiency of the proposed system by using R-tree and inverted indexes for keeping map tags. This is suitable for geographical data. The previous existing work also proposes a ranking technique that uses tf-idf scheme but adapted to geographical data. We find similarity between this work and our work in terms of using indexes for efficiency purposes, but this existing work is only suitable for GIS and maps domain. Zemke et al. (2012) discuss ranking indicators for mashing Web services. The existing proposed technique is implemented in Omelette Project (2013). This technique works by creating a mashup graph that depicts relationships between the mashing operation and the Web services they use. The proposed social indicators mainly work on the idea of how many times a Web service is used in mashing processes and how close a Web service is to mashing processes it is involved with in the graph. A main shortcoming of this work is that their techniques do not specify whether a Web service is relevant to the end-user in the first place. They assume that a set of Web services relevant to the end-user is given and then they rank Web services in that set. Another system that helps end-users in building their mashups is proposed in Chowdhury (2012). When the end-user chooses one component as part of his/her mashup, a proposal of the next component to be used is supplied by the system to the end-user. The next proposed component is chosen based on ranking mashup components. This existing work only mentions that it uses ranking in the proposed system. But it does not provide any details on the type of ranking it uses and it is stated in that paper that providing details on ranking mashups is part of future work. Ranking techniques for patterns of mashups are discussed in Radeck et al. (2012). This existing work uses collaborative filtering by observing the repetition of a pattern in other end-user mashups. The approach also uses semantic similarity by finding out mashup components that can replace each other. When an end-user is composing a mashup, the system proposes patterns that he/she can use based on the ranking value. Our work is different, it does not rank patterns of using mashup operators, but we rank mashup subtrees. Also, our ranking technique helps the end-user to select the mashup that closely reflects the end-user search criterion. While the previous work uses recommendation to help the end-user in composing mashups. Another key difference is that on the contrary to our system, the previous work does not take efficient mashup access into account. Ranking mashup components using quality measures for the sake of helping end-users in mashup composition is investigated in Picozzi et al. (2010). A mashup component that enhances the quality of the mashup it is going to be deployed in would be ranked higher. Ranking is based on the context of mashup components and the role they play in the mashup. This work does not focus on the efficiency of accessing mashups.

2.4. Mashups

Mashup platforms are flourishing in the era of Web 2.0 personalized applications. Several mashup platforms have been proposed in literature. One of the most popular mashup platforms is Yahoo Pipes (Yahoo Inc, 2007). It is a platform in which the end-user can use several types of operators to build his/her own mashups. These operators are used to fetch several data sources such as RSS and ATOM feeds. In addition, they are used to manipulate the fetched data. Marmite (Wong and Hong, 2007) is a Firefox plug-in that is designed to help the end-user in aggregating and updating several data sources in addition to directing the output of the aggregated data to other files and websites. Building mashups by example is proposed in Tuchinda et al. (2008). This platform enables end-users to extract data from

several websites. After that, the data is visualized to the end-user as tables. The end-user can manipulate tables and define rules between them. MashMaker (Ennals and Garofalakis, 2007) is a mashup platform that enables end-users to extract data sources and visually build queries on them. MARIO is a mashup platform that provides end-users with a cloud of tags that can be used to design mashups. MARIO also provides an algorithm that finds the best execution plan for a given mashup. Liu et al. (2007) propose a mashup platform that works on the principles of service oriented architecture where mashup builder, mashup user, and mashup server represent provider, consumer, and broker, respectively. Data integration services for situational applications are proposed via DAMIA (IBM Corp, 2013). Kulathuramaiyer (2007) uses semantic meta-data in a mashup platform that allows end-users to search digital libraries. JackBe Corp (2011) is a mashup platform that eases the creation of secure enterprise mashups via a visual interface. Jung (2012) proposes a platform through which end-users can collaborate in building mashups that browse information from several sources across the Web. Liu et al. (2011) introduce a service oriented architecture that enables the end-user to build mashups using integration patterns. Ngu et al. (2010) use semantic meta-data to mash non-service components that can functionally cooperate. Di Lorenzo et al. (2009) introduce a model for comparing mashup platforms. To the best of our knowledge, none of the previous platforms explicitly discuss the issue of mashup ranking.

The previous discussion of related works exposes the following drawbacks. First, existing ranking techniques mainly rely on keywords and URLs. This is not suitable for mashups because mashups are built as a hierarchy of mashup operators where relationship between operators must be taken into account. Second, existing ranking techniques for Web services and mashups do not focus on efficiency which does not work for mashups due to the high pressure imposed by mashup platforms which is explained in Section 3. As a result, an efficient ranking mechanism for mashups is needed.

3. Overview and challenges

In this section, we show how ranking can be incorporated within a mashup centric platform. Moreover, we lay out the challenges we faced in designing our mashup ranking technique.

A typical mashup (Beemer and Gregg, 2009) fetches data from several data sources distributed over the Web, and then it processes the fetched data using operations such as filtering, truncating, and sorting. We implement a basic mashup based prototype in which we deploy our mashup ranking technique. We use this prototype for the purpose of evaluating our mashup ranking technique. In our prototype, we present several mashups for the end-user. The end-user selects mashups of his/her interest and subscribes to them. Accordingly, the end-user is notified with any updates in the result of executing the mashups he/she subscribed to. In our prototype, the end-user uses keywords in his/her search attempt and the mashups that satisfy his/her search appear in the application window. Each displayed mashup has a label showing the ranking value of this mashup. The higher this value is, the closer this mashup in representing the end-user interest. Mashups are shown for the end-user in descending order based on their ranking value. This would bring the highly ranked mashups to end-user attention first. Each mashup has a button labeled '*Subscribe*', which enables the end-user to subscribe to that mashup. Upon subscribing to any mashup, our prototype adds that mashup to the list of mashups the end-user subscribed to, and consequently, our prototype is responsible for periodically executing that mashup and showing any updated result to the end-user.

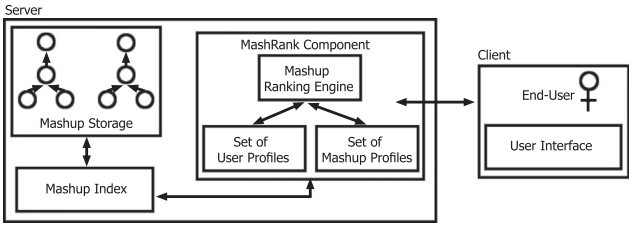


Fig. 2. Mashup ranking component integrated with a mashup centric platform.

Figure 2 shows how our mashup ranking component fits into a mashup centric platform.

Several challenges arise in designing a mashup ranking technique. First, since mashups are designed by end-users (not by developers), the number of mashups hosted by any mashup based system is potentially very high. In other words, developers design a Web service for a specific group of people who have interest in that Web service (one Web service per group of end-users). But in mashups case, each end-user has its own needs; and therefore, the end-user designs his/her own mashup (one mashup per end-user). Ranking mashups requires processing existing mashups in the system. Therefore, our mashup ranking technique is required to deal efficiently with this large number of mashups. Otherwise, the mashup platform in which our mashup ranking technique is employed would suffer from degraded performance. Second, when we compare a feed reader to a mashup reader, feed readers usually rank feeds based on the popularity of content of those feeds. The case is not the same in mashup readers because mashups do not only represent a source of information, but they also represent operators built on top of feeds to be executed in a logical order. The logical order of operator execution can be exploited toward a better ranking technique of mashups. Accordingly, our mashup ranking technique should work on the operator level of mashups and that makes its design more challenging than a feed ranking technique. Third, mashups are designed by end-users who might have common interests. This indicates that mashups designed by end-users might have identical parts. For example, suppose we have two end-users; each of them designed one mashup and it happens that the two mashups have identical operators in common. The number of operators in the two mashups is greater than the number of *unique* operators. Accordingly, if we do not consider commonality between mashups that would increase the number of stored operators in the system and in turn hurt the efficiency of our mashup ranking technique.

4. Processing mashups

Web 2.0 applications impose high pressure on servers (Sobel et al., 2008) which indicates that Web 2.0 platforms have to be efficient. As mashups belong to Web 2.0 category, we have to pay closer attention to the efficiency of this type of applications. Working with mashups requires two aspects. First, efficiently accessing mashup operators. Second, efficiently executing mashup operators. A technique that tackles the later aspect is proposed in Lin et al. (2012). The proposed technique sheds light on optimizing mashup trees via means of relational algebra. Our work handles the former aspect. A better mashup platform would incorporate both aspects.

In this section we describe how we access mashups in an efficient manner which makes our mashup ranking efficient. First, we explain how we represent mashups. Basically, each mashup consists of a set of operators ordered in a tree shape such as the mashup shown in Fig. 1. The leaves of the tree would be the first operators to be executed and they represent fetching data from

data sources. The root of the tree would be the last operator to be executed after which the final result of mashup execution is dispatched to the end-user. The operators in between are the data processing operators such as filter, truncate, and sort operators.

The operators we use in our system work on RSS feeds. RSS feeds contain list of items where each item consists of specific fixed properties such as title, description, author, and link. The 'title' property reflects the headline of the item. The 'description' property provides more details about the item. The 'author' property specifies who wrote this item. The 'link' property represents the hyperlink through which this item can be found on the Web. Consequently, our operators process the previously mentioned properties of items in RSS feeds. For example, as shown in Fig. 1, the union operator combines two RSS feeds into one. The filter operator keeps only those RSS items in which the 'title' property contains the keyword 'Dell'.

We use the following operators in our system. Fetch operator pulls the content of RSS feeds from the Web. Filter operators keep in RSS feeds only those items that satisfy the filter condition. Union operator combines two RSS feeds into one. Sort operator sorts the items of a RSS feed based on one of its properties such as title. Tail operator keeps only the last N items in a feed. Truncate operator keeps only the first N items in a feed. Unique operator removes any duplicate items. Count operator returns how many items exist in a feed. Subelement operator keeps only a given property of RSS items such as title and removes the other properties. Reverse operator reverses the order of items in a feed.

Each operator has a basic string representation that is formed of concatenating the identification number that we assign for each of its parameters. For example, '15|24|33|*Dell*' is the basic representation for a filter operator ($ID=15$) which filters the input RSS feed based on property '*title*' ($ID=24$) contains '*Dell*', where '33' is the ID of the '*contains*' operator. Table 1 shows the representation of operators of the mashup in Fig. 1.

Our mashup ranking technique depends on the Vector Space Model (VSM), therefore, our ranking technique would require accessing mashup operators for the sake of counting repetitions of operators. To find out the number of times an operator is repeated, the operator representation needs to be compared with the representation of all other operators in the system. Therefore, a lookup operation is needed to search for operators which have similar representation. Performing this process sequentially is inefficient. Because of that we use a B+ tree index to organize mashup operators. The keys of the index are formed from the representation of each operator. Using the B+ tree index minimizes the number of steps needed for comparing an operator representation with other operator representations in the system. Figure 3 shows an example of B+ tree index that stores operators of the mashup in Fig. 1.

As explained in Section 3, storing identical operators multiple times can hurt our mashup ranking performance, therefore, if we have a newly designed mashup in hand, we use the B+ tree index explained above to check if there is any currently existing operator in the system that matches the representation of one of the operators in the new mashup. If that happens, then the key (representation) of the new operator is not inserted into the index again because it already exists. For example, if we have a new mashup identical to the one presented in Fig. 1 except that the filter operator is different. In this case, searching the index for the two fetch operators and the union operator would result in a match; and therefore, we do not need to store a duplicate of these operators. But, since the filter operator is different than the one existing in the index, then it would be recognized as a new operator and inserted in the index.

Using mashup index and detecting identical operators help in minimizing the number of operators in the system and aids towards fast operator lookup operations. This provides a solid ground for designing an efficient mashup ranking technique.

Table 1
The representation of operators of Fig. 1.

Operator	Representation	Explanation
Fetch (Buy.com Feed)	09	09: ID of Buy.com feed
Fetch (Amazon Feed)	04	04: ID of Amazon feed
Union	SU|09|MU|04|EU	SU, MU, EU are separators between the two unioned components
Filter (Title contains Dell)	15|24|33|Dell	15: ID of filter operator 24: ID of property 'Title' 33: ID of 'contains' operator

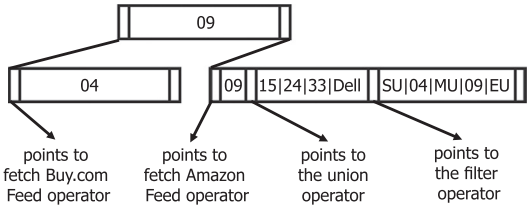


Fig. 3. Example of B+ index which stores operators of the mashup in Fig. 1.

5. Ranking mashups

Due to large number of mashups that satisfy the end-user search query, it is impossible for him/her to decide which of them best suits his/her needs. Consequently, it is the responsibility of the mashup centric platform to rank mashups such that highly ranked mashups are recommended for the end-user first.

5.1. Mashup ranking technique

We will exploit Vector Space Model (VSM) (Salton, 1971) in ranking mashups. In its primitive form, VSM depends on the existence and repetition of keywords in documents. In mashup ranking, that would be adapted as the existence and repetition of subtrees in mashups. This helps us to exploit the relationship between mashup operators in our ranking process.

In order for VSM to work, two characteristic vectors (profiles) are needed. The first profile is a user profile ($\mathbb{P}_{user}$) resulting from his/her selection of mashups he/she is interested in. The second profile is a mashup profile. By comparing a user profile with a given mashup profile, we can find how similar the mashup is to the user profile, and therefore, the degree of relevance of this mashup to the user interest can be decided. Figure 4 illustrates how mashup ranking is integrated with search process.

5.1.1. Constructing user profile

In a given session, the user selects several mashups which he/she is interested in. Those selected mashups will be used in constructing and updating user profile. Suppose that $\mathbb{M}_{user}=\{m_1, m_2, \ldots, m_n\}$ is the set of mashups selected by user in a given session where $n=|\mathbb{M}_{user}|$. Suppose $\mathbb{ST}_{user}=\{st_1, st_2, \ldots, st_r\}$ is the set of different subtrees existing in $\mathbb{M}_{user}$ mashups. Using the same notation, $\mathbb{ST}_{m_i}=\{st_1, st_2, \ldots, st_z\}$ is the set of different subtrees in mashup m_i. Each selected mashup m_i in $\mathbb{M}_{user}$ has a characteristic vector ($\mathbb{W}_{m_i}$) which represents the weights of different subtrees of $\mathbb{ST}_{m_i}$ in mashup m_i such that $\mathbb{W}_{m_i}=\{w^i_{st_1}, w^i_{st_2}, \ldots, w^i_{st_z}\}$. In the previous equation, $w^i_{st_z}$ corresponds to weight of subtree st_z in mashup m_i, such that $z=|\mathbb{ST}_{m_i}|$. This value is computed based on term frequency (tf) scheme as shown in Eq. (1) such that $tf_{i,z}$ corresponds

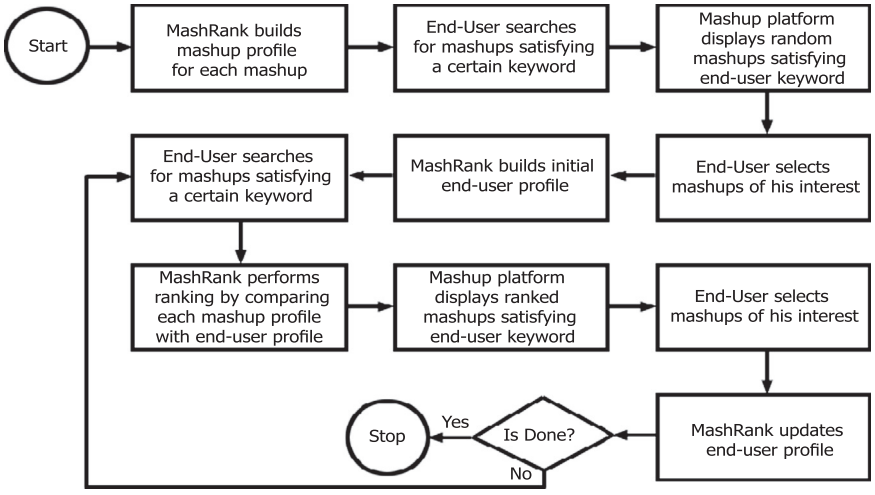


Fig. 4. A search process which involves mashup ranking.

to the frequency of subtree st_z in mashup m_i. The term frequency in this case measures the correlation between the frequency of subtree st_z in mashup m_i and the frequency of the same subtree in all other selected mashups $\mathbb{M}_{user}$.

$$w_z^i = \frac{tf_{i,z}}{\sum_{k=1}^{n} tf_{k,z}} \tag{1}$$

We note here that the summation of weights for a 'given subtree' in 'all' mashups is equal to one. This is an example to clarify this point. Suppose we have 3 mashups, namely m_1, m_2, and m_3. Assume we have a given subtree 'x'. Now, suppose subtree x appeared 2, 3, and 1 times, respectively, in mashups m_1,m_2, and m_3. So, the term in Eq. (1) is computed 3 times for subtree 'x' (once per mashup). The verbal explanation of the term is 'Number of times a subtree exists in a mashup divided by number of times a subtree exists in all mashups'. So, the terms for subtree 'x' would be as follows. In m_1, the term is $\frac{2}{6}$. In m_2, the term is $\frac{3}{6}$. In m_3, the term is $\frac{1}{6}$. The summation of the previous terms for subtree x in all mashups is equal to 1.

Now, the user profile is represented as $\mathbb{P}_{user} = \{q_{st_1}, q_{st_2}, \ldots, q_{st_r}\}$ where q_{st_r} represents the weight for subtree st_r in all selected mashups $\mathbb{M}_{user}$. This value is computed as shown in the following equation:

$$q_{st_r} = \frac{\sum_{u=1}^{n} w_{st_r}^u}{n} \tag{2}$$

The way we model the previously mentioned mashup and user profiles is as text files. Recall that a user profile ($\mathbb{P}_{user}$) consists of a set of values where each value is the weight of a subtree of the mashups he/she previously selected. Therefore, the user profile is modeled as a text file containing pairs of the following format (*ID of Subtree*, *weight*). Similarly, a mashup profile is the characteristic vector of the mashup $\mathbb{W}_{m_i}$ and it consists of a set of weights of each subtree in the same mashup. Therefore, a mashup profile is also modeled as a text file containing pairs of the format (*ID of subtree*, *weight*).

5.1.2. Similarity between user profile and mashup profile

The ranking process works by computing a similarity value[2] between the user profile characteristic vector ($\mathbb{P}_{user}$) and the mashup characteristic vector ($\mathbb{W}_{m_i}$), the higher the similarity value is, the mashup is expected to be more relevant to user. We refer to similarity value as $Sim(\mathbb{W}_{m_i}, \mathbb{P}_{user})$. This value is computed using the cosine measure (Salton, 1989) as shown in Eq. (3) where z is the number of different subtrees in mashup m_i.

$$Sim(\mathbb{W}_{m_i}, \mathbb{P}_{user}) = \frac{\mathbb{W}_{m_i} \cdot \mathbb{P}_{user}}{|\mathbb{W}_{m_i}| \cdot |\mathbb{P}_{user}|} = \frac{\sum_{h=1}^{z} (w_{st_h}^i \cdot q_{st_h})}{\sqrt{\sum_{h=1}^{z} (w_{st_h}^i)^2} \cdot \sqrt{\sum_{h=1}^{z} (q_{st_h})^2}} \tag{3}$$

Since we model mashup and user profiles as two text files that contain pairs of values (*ID of subtree*, *weight*), then Eq. (3) works by performing a comparison between these weight values of pairs in user profile and mashup profile that belong to the same subtree. Given a set of mashups, the similarity value between each of them and the user profile is computed. After that, mashups are ordered in descending style based on computed similarity values and the mashups will be presented to the user in the same descending order. In other words, the mashup that is first introduced to user has the highest similarity value and it is supposed to be more relevant to the end-user.

5.1.3. Updating user profile

After one end-user session ends, an initial value of the user profile is built. Now, when the user next session ends, the user profile $\mathbb{P}_{user}$ has to be updated in order to reflect changes in interest. When the end-user is using our prototype in a new session, suppose that he/she selected a new mashup m_{n+1}. This event causes the following actions. First, creating a new mashup profile $\mathbb{W}_{m_{n+1}}$ for mashup m_{n+1}. Second, updating user profile $\mathbb{P}_{user}$.

Regarding the new mashup profile, a new characteristic vector $\mathbb{W}_{m_{n+1}}$ should be created for the new mashup containing a weight w_z^{n+1} for each subtree st_z in the newly selected mashup m_{n+1}. Notice that the weight w_z^{n+1} is computed the same way as for w_z^i which is previously shown in Eq. (1).

Regarding user profile $\mathbb{P}_{user}$, two types of subtrees could exist in the newly selected mashup. A given subtree st_r exists in the current user profile $\mathbb{P}_{user}$ and another subtree st_{r+1} does not exist in that profile. In the later case, the subtree st_{r+1} does not have a weight in the user profile characteristic vector $\mathbb{P}_{user}$. In other words, the end-user has never selected a mashup with this subtree before. This might reflect a new interest for the end-user. In this case, the weight $q_{st_{r+1}}$ of the new subtree is evaluated and added to the user profile $\mathbb{P}_{user}$. In the former case, subtree st_r has a weight q_{st_r} which already exists in the user profile $\mathbb{P}_{user}$, this weight has to be recomputed and it will replace the old q_{st_r} value in the user profile. Both q_{st_r} and $q_{st_{r+1}}$ are computed based on Eq. (2) which is previously explained in Section 5.1.2.

[2] The similarity value is often referred to as Retrieval Status Value (RSV).

5.2. Binary ranking of mashups

In this subsection, we discuss a simple scheme of mashup ranking which depends on the primitive binary ranking. We discuss this scheme for the purpose of comparing it to our mashup ranking scheme. In binary ranking of documents, if a keyword exists in a document, then the document is considered relevant to the end-user. Otherwise, it is not relevant. Adapting this to mashups, if a mashup contains any mashup subtree that exists in any previously selected mashup by the end-user, the mashup is considered relevant to the end-user. On the other hand, if the mashup does not contain any subtree that exists in any previously selected mashup, the mashup is considered irrelevant to the end-user.

5.3. Enhancements on mashup ranking

Sometimes, the parameters of mashup operators might deceive the mashup ranking process. One downside to this is that mashups which are not related to end-user interest might be considered as relevant to him/her. For example, the user might be looking for mashups with keyword 'Big Apple', and he/she literally means a big apple fruit. If keyword semantics is not taken into consideration during ranking process, the user might be offered with mashups that deal with 'Big Apple' as the nickname for New York City which is not what the user is interested in. This can actually occur in the other way around wherein mashups which are actually relevant to the end-user would not be considered important to him/her. For example, suppose one end-user is interested in mashups related to 'Football'. Some mashups might process information related to 'Soccer'. 'Soccer' is a synonym for 'Football'; and therefore, if keyword synonyms are not taken into account, mashups related to 'Soccer' would not be considered relevant to end-users.

To eliminate these problems, we utilize a simple data repository that contains words and their corresponding synonyms and semantics. Basically, when the end-user uses a keyword to search for mashups, the mashup platform retrieves related semantics and synonyms of that keyword from the data repository. If the keyword has synonyms, then these synonyms are added to end-user query and that will enrich the result that might be produced from search process. Similarly, if the keyword has different semantics, then these semantics are displayed to the end-user so that he/she picks the right semantic that matches his/her intention. This way, any irrelevant mashups will be excluded from search result. From now on, we refer to mashup ranking which applies the aforementioned enhancements as 'Enhanced Mashup Ranking'.

The repository consists of two text files. The first one contains a set of lines such that each line consists of a keyword and list of its synonyms. The second text file contains a set of lines such that each line consists of a keyword and list of its semantics. However, we believe that using ontologies would be the best implementation of this concept.

6. Using keywords instead of mashup subtrees

One possible way to rank mashups is by ranking the news items and summaries resulting from the mashup execution. This is similar to the work in NectaRSS (Samper et al., 2008) which ranks news headlines in a feed reader using vector space model. To make notions clear, we note that a headline is equivalent to title of a RSS feed item and a summary is equivalent to description of a RSS feed item.

We argue that ranking the news items resulting from mashup execution is not suitable for mashups. This is related to one of the disadvantages of vector space model, namely the increasing complexity resulting from large number of keywords that have to be taken into consideration in the ranking process. Here, it is worthy to mention that number of operators in a single mashup is significantly smaller than the number of keywords that exist in the news headlines and summaries resulting from the mashup execution. For example, one mashup can fetch data from 'Buy.com' and 'Amazon' feeds, merge the data, and filter merged data such that it contains the keyword 'Dell' and sort data in ascending order. This mashup consists of only 5 operators. On the other hand, the number of different keywords contained in the headlines and summaries of the mashup execution is definitely more than 5 keywords.

Assume that 6 headlines resulted from execution of the previously mentioned mashup with 5 different keywords in each headline. We assumed low numbers so that we maintain the fairness of comparison. This means that the vector size for each headline is 5. Therefore when the vector space model is computed, the similarity function between user profile vector and headline vectors has to be computed 6 times (once between the user profile vector and each headline vector). This is computationally expensive.

In addition, if headline summaries are involved in the ranking process, then number of unique keywords increases dramatically because summaries are much longer than headlines. This will even increase the complexity of the ranking process. On the other hand, in mashup ranking, the vector of the mashup consists of different subtrees that exist in the mashup. In our example, the number of different subtrees is 10. This implies that the size of user profile vector is going to be smaller.

Also, the similarity is computed between user profile vector and mashup vector (one computation in this case compared to 6 computations in headlines case). In addition, the average number of operators in a mashup is relatively small. In contrast, the number of keywords in summaries is relatively large; this implies that using mashup operators for ranking has lower complexity than using keywords of summaries.

Moreover, one extra advantage of operator based mashup ranking is that relationship between operators is used in the ranking process. In other words, the ranking process not only considers different operators, but it also considers different subtrees. This is important because if the user is interested in a mashup that fetches data from 'Amazon' then filters data based on 'Dell' computers, then he/she is most likely interested in other mashups that contain the same subtree of execution.

7. System evaluation

In this section, we evaluate our mashup ranking technique by performing real experiments with 12 end-users. This number is considered acceptable given that it is hard to find end-users willing to participate in experiments. Each end-user is asked to complete 40 sessions using our experimental prototype. In each session, the end-user is presented with a set of 10 different mashups chosen from a pool of 30 mashups. Out of these 10 mashups, the end-user selects mashups of his/her interest. The 40 sessions which the end-user is requested to go through are classified into the following groups based on the technique employed in ranking.

- Random ranking. Here the end-user goes through one initial session followed by 10 sessions. In all these sessions, mashups displayed for the end-user are chosen randomly.
- Binary ranking. In this case, the end-user performs one initial session followed by 10 sessions. In the initial session, mashups displayed for the end-user are chosen randomly as an initialization step. Based on the end-user selections, the initial user

profile is built. Now, in the following 10 consecutive sessions, mashups are compared to user profile and hence ordered using binary ranking technique. Then, the 10 highest ranked mashups are presented to the end-user.

- Mashup ranking. In this part, the end-user undergoes one initial session followed by 10 sessions. As an initialization step, mashups displayed for the end-user are chosen randomly in the first session. Based on the end-user mashup choices, the initial user profile is built. In the following 10 consecutive sessions, mashups are compared to the user profile and hence ordered using our mashup ranking technique. Then, the 10 highest ranked mashups are presented to the end-user.
- Enhanced mashup ranking: This part is similar to Mashup Ranking experiment. The only difference is that the enhancements discussed in Section 5.3 are employed here. In other words, keyword synonyms and semantics are utilized in ranking technique. Similar to other experiments, the end-user undergoes one initial session followed by 10 consecutive sessions.

Notice that the 10 mashups displayed for the end-user in the first initial session of each category are the same. This is important so that all experiments start with the same initial input and therefore they can be compared together.

7.1. Building mashups

The 30 mashups used in our experiments are built as follows. The average number of operators per mashup is 10. The full list of operators is explained in Section 4. It is noticeable that one source of variety of our mashups comes from the data sources used in the fetch operators. We use a set of data sources extracted from Syndic8 (Barr and Kearney, 2001) based on 5 categories, namely, sport, health, travel, music, and politics. We extracted 20 data sources from each category. This way, we have a pool of 100 data sources from which we randomly choose to build our mashups. The way we extract this pool of data sources is as follows.

- First, as shown in Fig. 5 we manually go to Syndic8 home page and in the 'search for feed' textbox, we search for feeds pertaining to the previously mentioned 5 categories (5 search attempts). The result of each search attempt is a page containing a list of feeds satisfying the search category. An example of such a page is shown in Fig. 6.
- Second, we developed a java program that extracts the content of each feed found in each search result page. The idea is that the search result contains HTML patterns in which information of feeds are listed (see Fig. 7). The important information we need is Feed URL and type of feed. We only care about RSS feeds because the mashup operators we implemented only deal with RSS feeds.
- Third, whenever the type contains the keyword 'RSS', we simply extract the feed URL and then using our java program, we pull down the content of each feed URL page and store it offline locally on our machine. An example of how pulled page looks like is shown in Fig. 8 and an example of how the code for that page looks like is presented in Fig. 9.

Fig. 5. Part of Syndic8 home page which facilitates searching for feeds.

Feed URL	Feed ID	Site Name	Created	Approved	Feed Changed
XML	19743	Food and Farm News	2002-10-25	2002-11-02	2013-01-31
XML	34643	Kansas City infoZine Food Headlines	2003-10-03	2003-12-18	2013-01-31
XML	42965	Basilikumspirit	2004-01-05	2004-01-14	2013-01-09

Fig. 6. Part of search result showing feeds related to category 'Food'.

The result of the previous steps is a pool of 100 RSS feed files which constitute the pool of feeds from which we select the data sources of the fetch operators in our mashups.

The second source of variety of our mashups comes from the keywords used in the filter operators. We use Google SK tool (Google Inc, 2008) to extract keywords. Since we have 5 categories of mashups, it is important for our keywords to be applicable for those categories. For example, if our mashup uses a 'Food' related feed, then it makes sense for the filter operator to have a food related keyword. Accordingly, for each of the 5 categories, we manually search for keywords related to that category. Google SK Tool has the flexibility of downloading the resulting keywords in different formats including XML which we use here. This way we have a pool of 50 keywords (10 per category) from which filter operators are built. Figure 10 shows how Google SK Tool is used to find and download keywords related to 'food' category. The key parts of the figure are marked in red circles and they are self-explanatory.

```
<td class="infotablecell">
   <center><a href="http://rss.infozine.com/kc/food.xml">
   <img src="xml.gif" alt="http://rss.infozine.com/kc/food.xml"
   type="application/rss+xml"></a></center>
</td>
<td class="infotablecell">
   <a href="feedinfo.php?FeedID=34643"
   title="Kansas City infoZine Food Headlines
   (http://rss.infozine.com/kc/food.xml)">34643</a></td>
<td class="infotablecell">
   <a href="feedinfo.php?FeedID=34643">Kansas City infoZine Headlines</a>
</td>
<td class="infotablecell">2003-10-03  </td>
<td class="infotablecell">2003-12-18  </td>
<td class="infotablecell">2013-01-31  </td>
```

Fig. 7. Part of the HTML code for the webpage in Fig. 6. The red circle shows the URL of the feed. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

7.2. Evaluation metrics

To evaluate our mashup ranking technique, we use two common metrics, namely, Retrieval Status Value (RSV) and R-precision which we explain next.

7.2.1. Average retrieval status value

Previously, we showed how a similarity value is computed for each mashup. This value shows how close the mashup is to the end-user interest. A higher similarity value means that it is more likely that the end-user would be interested in that mashup. An alternative name for similarity value is Retrieval Status Value (RSV). The average RSV can be used to test how good our mashup technique is in reflecting end-user interest. Assume that the end-user selected N mashups in a given session. The ideal scenario happens if the N selected mashups are the same as the first N mashups presented for the end-user. Let M_n corresponds to the first N mashups presented to the end-user. Consequently, $AvgRSV_{M_{user}}$ corresponds to average RSV for the N mashups selected by the end-user. Similarly, $AvgRSV_{M_n}$ corresponds to average RSV for the first N mashups displayed to the end-user. $AvgRSV_{M_n}$ is the highest possible value for average RSV because the first N mashups presented to the end-user have the highest RSV values. Therefore, we can define quantity RE in Eq. (4) which indicates the effectiveness of our ranking technique. Higher RE value indicates a more effective ranking technique:

$$RE = \frac{AvgRSV_{M_{user}}}{AvgRSV_{M_n}} \tag{4}$$

7.2.2. R-Precision

R-Precision is another metric that helps in measuring how good a ranking technique is. A ranking technique is more effective if a higher percentage of mashups the end-user selected appear within the first D mashups displayed to the end-user. The best scenario happens when the mashups the end-user selected are all within the first D mashups presented to the end-user. Let N be the number of mashups the end-user selected in a given session. Let Inc_N reflect the number of mashups the end-user selected that fall within the first D mashups displayed to the end-user. Therefore, in Eq. (5), we can define $RPrec$ which represents the R-Precision value for a ranking technique in a given session. A high value of $RPrec$ is a sign for an effective ranking technique:

$$RPrec = \frac{Inc_N}{N} \tag{5}$$

One point to mention is that average RSV and R-Precision values are not defined after the end of the first end-user session. Therefore, these values are only recorded for the 10 sessions that follow the first initial session.

7.3. Experiments

First, we measure the RE value for each end-user in the last session of 'Random Ranking', 'Binary Ranking', 'Mashup Ranking', and 'Enhanced Mashup Ranking' categories. This is shown in Fig. 11 where we notice that RE values in 'Enhanced Mashup Ranking' is in average 9% higher than 'Mashup Ranking' case, this is due to the effect of keyword synonyms and semantics in considering mashups that truly represent the end-user interest. Also, RE values in 'Mashup Ranking' case are in average 25% and 90% higher than 'Binary Ranking' and 'Random Ranking' cases, respectively. This implies that mashups displayed for the end-user using our mashup ranking technique are more interesting to him/her than mashups chosen randomly. Also, this implies that 'Mashup Ranking' technique is more effective in catching the

Fig. 8. Part of the webpage of one feed. It shows the content of the feed as a list of RSS items.

```
<item>
  <title>
    KCMO Grade a Food Excellence Award Recipients
  </title>
  <link>http://www.infozine.com...../sid/54739/</link>
  <description>  This year, approximately 10 percent of
                 permitted food establishments in Kansas
                 City, MO met the higher standards of
                 this award.
  </description>
</item>
<item>
  <title>
    Updated: Whole Foods Market Recalls Alaskan Sockeye Salmon
  </title>
  <link>http://www.infozine.com...../sid/54682/</link>
  <description>  The recalled items were sold in stores in
                 Colorado, Connecticut, Florida, Idaho, Kansas,
                 Maine, Massachusetts, New Jersey,New Mexico,
                 New York, Rhode Island, and Utah.
  </description>
</item>
```

Fig. 9. Part of the RSS code of the feed as shown in Fig. 8.

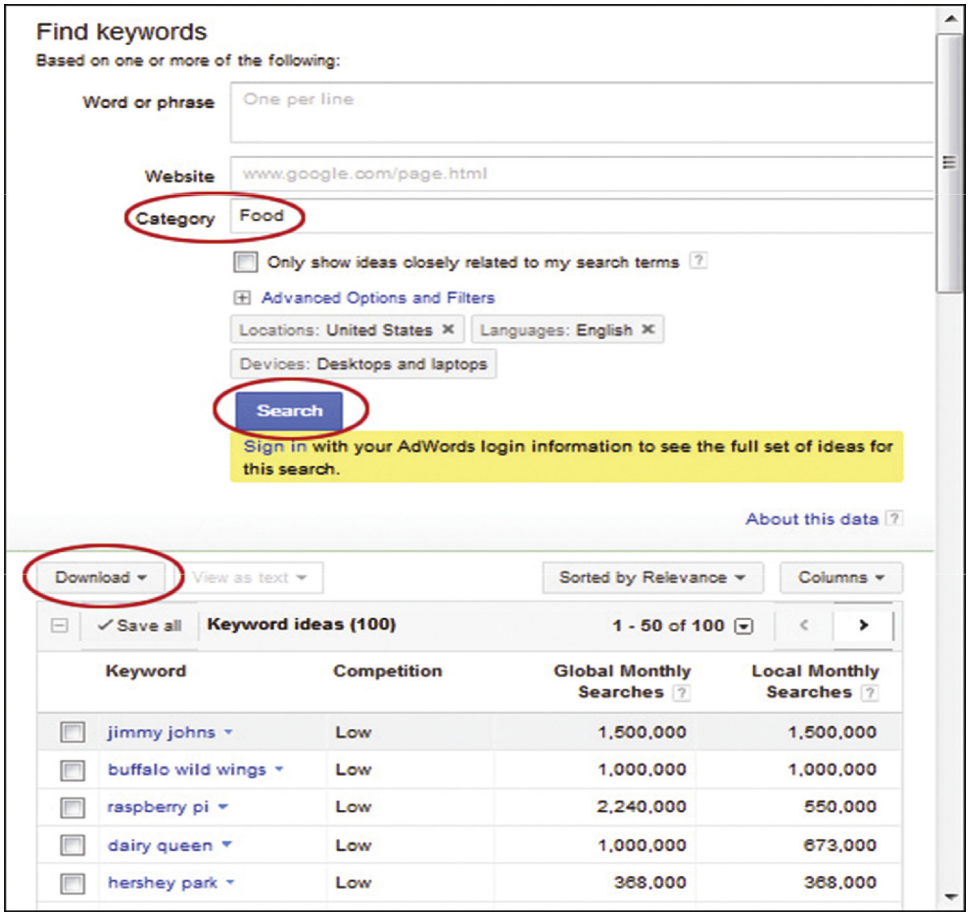


Fig. 10. Using Google SK Tool to find and extract keywords.

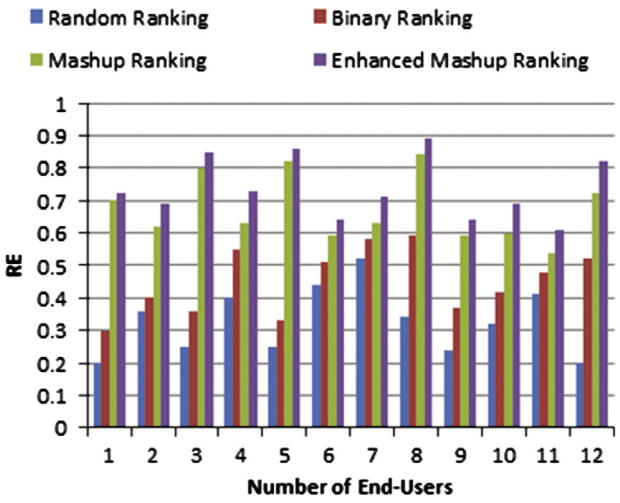


Fig. 11. RE values for end-users in their last session.

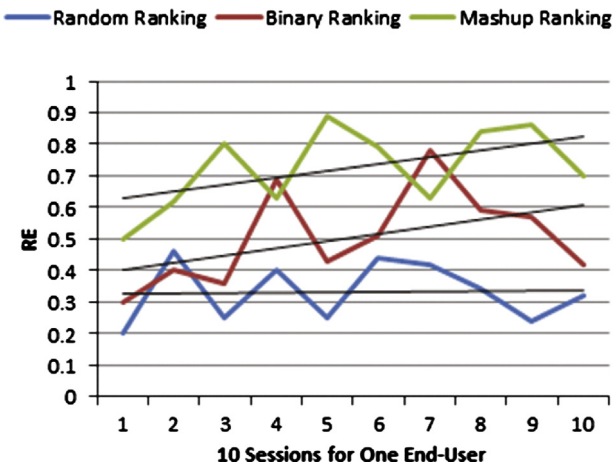


Fig. 12. RE values for one end-user in all his/her sessions.

end-user interest than 'Binary Ranking' technique. In the following experiment (Fig. 12), we pick an end-user randomly. Then, we measure RE values in each of his/her 10 sessions which he/she goes through in each ranking technique. For each ranking technique, we plot its trend line in order to grasp patterns in the graph. Trend lines for 'Mashup Ranking' and 'Binary Ranking' shows that RE values increase in these two techniques, this shows that as sessions pass, ranking becomes closer and closer to the end-user interest. One thing to mention here is that in many experiments, we noticed that there is no specific pattern for the trend line of 'Random Ranking' technique. If 'Random Ranking' happened to pick mashups which are most interesting to the end-user, the trend line would be inclining. Otherwise, it would be descending. In Fig. 12, we notice that RE values are increasing in most points, but there are few points where RE values decrease. Those points represent the time when the end-user selects a mashup that he/she has not selected before. Consequently, this new mashup is not present in end-user profile. This shift of interest for the end-user would decrease RE value. Nevertheless, since the new mashup would be incorporated with end-user profile, RE values start to increase again. Next, we repeat the previous two experiments while

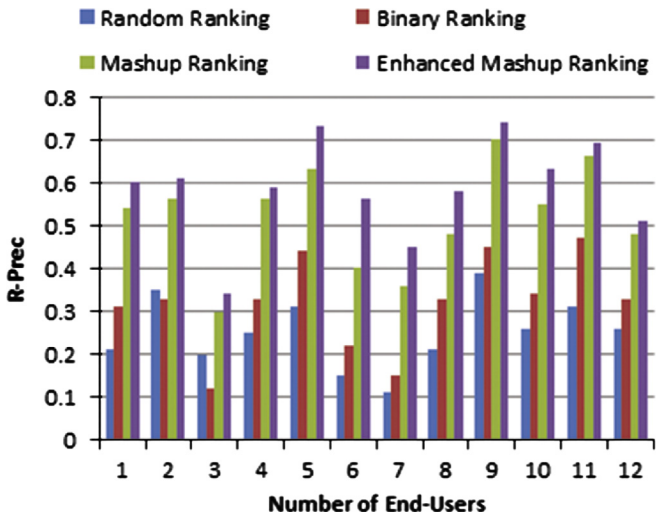


Fig. 13. R-Precision values for end-users in their last session.

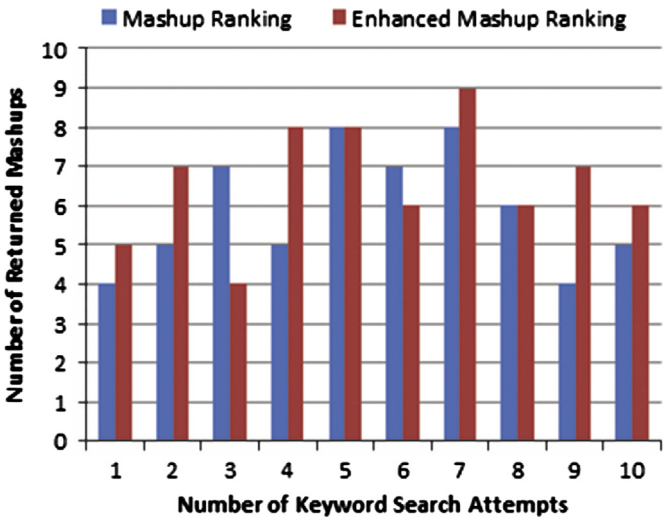


Fig. 15. Number of mashups returned in different search attempts.

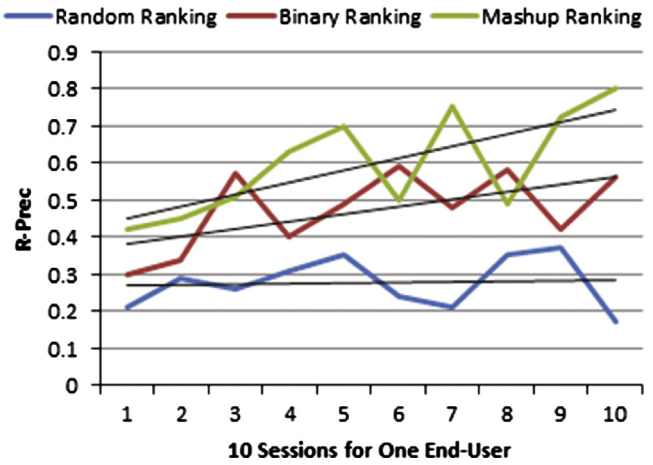


Fig. 14. R-Precision values for one end-user in all his/her sessions.

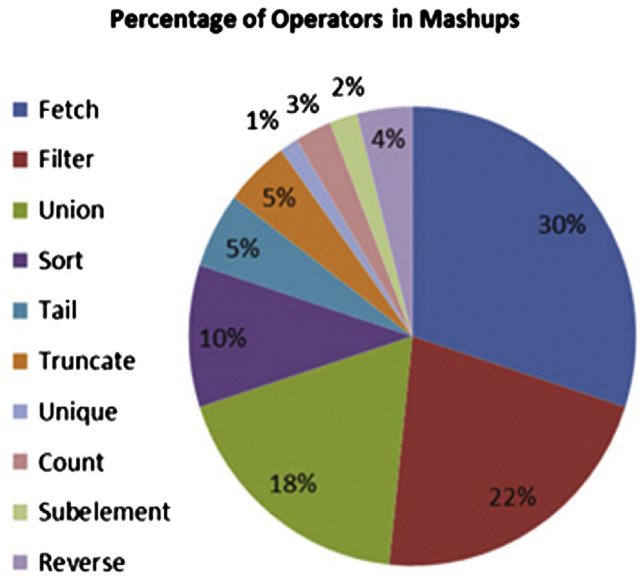


Fig. 16. Percentage of operators appearing in mashups.

measuring R-Precision values instead of RE values. The same patterns resulting in Figs. 11 and 12 also appear in Figs. 13 and 14. In Fig. 13, R-Precision values in 'Enhanced Mashup Ranking' are in average 12% higher than 'Mashup Ranking' technique. R-Precision values in 'Mashup Ranking' technique are in average 32% higher than 'Binary Ranking' technique. The three techniques show higher R-Precision values than 'Random Ranking' technique. This shows that the end-user finds more interesting mashups among those presented to him/her by 'Mashup Ranking' technique and its enhanced version. Figure 14 demonstrates inclining trend lines for'Mashup Ranking' and 'Binary Ranking' techniques. Throughout our experiments, we notice that the trend line for 'Random Ranking' techniques is fluctuating as it is not stable in representing end-user interest.

In the following experiment, we show the effect of incorporating keyword semantics and synonyms into our 'Mashup Ranking' scheme. Here, we asked one end-user to attempt 10 keyword based search requests. In each request, the search process is performed twice with and without taking keyword semantics and synonyms into account. In each search process, we compute the number of mashups returned to the end-user. In Fig. 15, when taking keyword synonyms and semantics into consideration, we notice that the number of returned mashups fluctuates between exceeding and falling behind the same number when keyword synonyms and semantics are not taken into account. The former case indicates that keyword synonyms have been used which in turn caused additional mashups to be considered relevant to the end-user. The latter case indicates a scenario where keyword semantics caused other mashups not to appear for the end-user because they do not match the semantics of his/her search request. The same figure has points where the average number of selected mashups in both schemes is the same. This happens when mashups containing synonyms of keywords the end-user is interested in do not exist. It also happens when the keyword used for the end-user search query does not have semantics other than its original meaning. In Fig. 16, we plot a pie chart showing the percentage of operators appearing in our mashups dataset. We believe that this is important because if a certain type of operator tends to appear very frequently in mashups, this might indicate that it is popular among end-users. We can see from Fig. 16 that 'Fetch', 'Filter', 'Union', and 'Sort' operators are the most frequent in mashups and they represent more than 45% of used operators.

The previous set of experiments showed that 'Mashup Ranking' and its enhanced flavor are better in depicting end-user interests than 'Binary Ranking' and 'Random Ranking' techniques. This is due to the nature of VSM that deals with the existence and absence of mashup operators in a more sensitive way than 'Binary Ranking'. In addition, 'Mashup Ranking' takes operators

and subtrees of operators into consideration while 'Binary Ranking' only considers mashup operators. Moreover, involving keyword synonyms and semantics in 'Enhanced Mashup Ranking' added a positive effect to 'Mashup Ranking' technique.

8. Conclusion and future work

Mashup platforms host a large number of mashups. When the end-user is searching for a mashup that matches his/her needs, the end-user would be overwhelmed by the large number of mashups that would satisfy his/her needs. Therefore, we designed a ranking technique tailored for the technical details of mashups that use RSS feeds as data sources. This ranking technique depends on Vector Space Model to find similarity between end-user request and existing mashups. We make sure our ranking technique is efficient by utilizing indexing data structure that facilitates efficient access of large number of mashups. Our technique is carefully evaluated using a set of experiments with real end-users. Results showed that the enhanced version of our ranking technique 'Enhanced Mashup Ranking' is preferred over the other techniques because it outperforms them in finding mashups that closely resemble the end-user interest. As part of our future work, we can improve our ranking technique by applying higher weights for operators that appear the most in mashups. We expect this to enhance the accuracy of our ranking technique in representing the end-user interest. The efficiency part of our work focuses on efficient access of mashup operators. In future, we plan to extend the efficiency of our system by also incorporating the efficiency of executing mashup operators into our system.

References

Abiteboul S, Greenshpan O, Milo T, Polyzotis N. MatchUp: autocompletion for mashups. In: International conference on data engineering; 2009. p. 1479–82.

Barr J, Kearney B. Syndic8 feeds repository, ⟨http://www.syndic8.com⟩ , 2001 [accessed 10 February 2013].

Beemer BA, Gregg DG. Mashups: a literature review and classification framework. Journal of Future Internet 2009;1:59–87.

Caverlee J, Liu L, Rocco D. Discovering and ranking web services with basil: a personalized approach with biased focus. In: Proceedings of the 2nd international conference on service oriented computing, ICSOC '04. ACM, New York, NY, USA; 2004. p. 153–62.

Chen T, Han W-L, Wang H-D, Zhou Y-X, Xu B, Zang B-Y. Content recommendation system based on private dynamic user profile. In: International conference on machine learning and cybernetics. IEEE; 2007. p. 2112–8.

Chowdhury S. Assisting end-user development in browser-based mashup tools. In: International conference on software engineering (ICSE); 2012. p. 1625–27.

Cingil I, Dogac A, Azgin A. A broader approach to personalization. Communications of the ACM 2000;43(8):136–41.

Datta A, Dutta K, VanderMeer D, Ramamritham K, Navathe SB. An architecture to support scalable online personalization on the web. VLDB Journal 2001;10(1):104–17.

Di Lorenzo G, Hacid H, Paik H-y, Benatallah B. Data integration in mashups. SIGMOD REC 2009;38(1):59–66.

Dong X, Halevy A, Madhavan J, Nemes E, Zhang J. Similarity search for web services. In: Proceedings of the thirtieth international conference on very large data bases, VLDB '04. VLDB Endowment. 2004. p. 372–83.

Ennals RJ, Garofalakis MN. Mashmaker: mashups for the masses. In: ACM SIGMOD international conference on management of data; 2007. p. 1116–8.

Google Inc., Google SK Tool. ⟨https://adwords.google.com/o/Targeting/Explorer?__c=1000000000&__u=1000000000&ideaRequestType=KEYWORD_IDEAS⟩, 2008. [accessed 10 February 2013].

Greenshpan O, Milo T, Polyzotis N. Autocompletion for mashups. Proceedings of the VLDB Endowment 2009;2(1):538–49.

IBM Corp, Damia. ⟨http://services.alphaworks.ibm.com/damia/⟩, 2007 [accessed 10 February 2013].

Intel Corp., Mash maker. ⟨http://software.intel.com/en-us/articles/intel-mash-maker-mashups-for-the-masses⟩, 2007 [accessed 10 February 2013].

JackBe Corp., Presto enterprise mashups. ⟨http://www.jackbe.com/products/presto⟩, 2011 [accessed 10 February 2013].

Jung JJ. Collaborative browsing system based on semantic mashup with open apis. Expert Systems with Applications 2012;39(8):6897–902.

Kulathuramaiyer N. Mashups: emerging application development paradigm for a digital journal. Journal of Universal Computer Science 2007;13(4):531–42.

Lin H, Zhang C, Zhang P. An optimization strategy for mashups performance based on relational algebra. In: Proceedings of the 14th Asia-Pacific international conference on web technologies and applications, APWeb'12, Springer-Verlag, Berlin, Heidelberg; 2012. p. 366–75.

Liu F, Yu C, Meng W. Personalized web search by mapping user queries to categories. In: Conference on information and knowledge management archive. ACM, New York, NY, USA; 2002. p. 558–65.

Liu X, Hui Y, Sun W, Liang H. Towards service composition based on mashup. In: IEEE congress on services; 2007. p. 332–9.

Liu Y, Liang X, Xu L, Staples M, Zhu L. Composing enterprise mashup components and services using architecture integration patterns. Journal of Systems and Software 2011;84(9):1436–46.

Mobasher B, Cooley R, Srivastava J. Creating adaptive web sites through usage-based clustering of URLs. In: Workshop on knowledge and data engineering exchange. Washington, DC, USA: IEEE Computer Society; 1999. p. 19.

Mobasher B, Cooley R, Srivastava J. Automatic personalization based on web usage mining. Communications of the ACM 2000;43(8):142–51.

Mobasher B, Dai H, Luo T, Nakagawa M. Effective personalization based on association rule discovery from web usage data. In: Workshop on web information and data management archive. ACM; 2001. p. 9–15.

Murugesan S. Understanding Web 2.0. IT Professional 2007;9(4):34–41.

Ngu AHH, Carlson MP, Sheng QZ, Paik H-y. Semantic-based mashup of composite applications. IEEE Transactions on Service Computing 2010;3(1):2–15.

Omelette Project, Omelette Project. ⟨http://www.ict-omelette.eu⟩, 2009 [accessed 10 February 2013].

Paolucci M, Kawamura T, Payne TR, Sycara KP. Semantic matching of web services capabilities. In: Proceedings of the first international semantic web conference on the semantic web, ISWC '02, London, UK: Springer-Verlag; 2002. p. 333–47.

Picozzi M, Rodolfi M, Cappiello C, Matera M. Quality-based recommendations for mashup composition. In: Proceedings of the 10th international conference on current trends in web engineering, ICWE'10. Berlin, Heidelberg: Springer-Verlag; 2010, p. 360–71.

Pietschmann S, Radeck C, Meissner K. Semantics-based discovery, selection and mediation for presentation-oriented mashups. In: Proceedings of the 5th international workshop on web APIs and service Mashups. New York, NY, USA: ACM; 2011. p. 71–8.

Radeck C, Lorz A, Blichmann G, Meibner K. Hybrid recommendation of composition knowledge for end user development of mashups. In: Proceedings of the 7th international conference on internet and web applications and services (ICIW2012). Stuttgart, Germany; 2012.

Salton G. The SMART retrieval system: experiments in automatic document processing. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.; 1971.

Salton G. Automatic text processing: the transformation, analysis, and retrieval of information by computer. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.; 1989.

Samper JJ, Castillo PA, Araujo L, Merelo JJ, Cordón ı, Tricas F. Nectarss, an intelligent RSS feed reader. Journal of Network and Computer Applications 2008;31:793–806.

Shahabi C, Chen Y-S. Web information personalization: challenges and approaches. In: Bianchi-Berthouze N, editor. Databases in Networked Information Systems, Lecture Notes in Computer Science. Berlin Heidelberg: Springer; 2003. p. 5–15.

Skoutas D, Simitsis A, Sellis T. A ranking mechanism for semanticweb service discovery. In: IEEE congress on services; 2007. p. 41–8.

Sobel W, Subramanyam S, Sucharitakul A, Nguyen J, Wong H, Klepchukov A, et al. Cloudstone: multi-platform, multi-language benchmark and measurement tools for web 2.0. In: Cloud computing and its applications (CCA), CCA '08; 2008. p. 333–47.

Tuchinda R, Szekely P, Knoblock C. Building mashups by example. In: International conference on intelligent user interfaces; 2008. p. 139–48.

Wong J, Hong J. Making mashups with marmite: towards end-user programming for the web. In: SIGCHI conference on human factors in computing systems; 2007. p. 1435–44.

Yahoo Inc., Yahoo pipes. ⟨http://pipes.yahoo.com/⟩, 2007 [accessed 10 February 2013].

Zemke T, Fernández-Villamor JI, Iglesias CA. Ranking web services using centralities and social indicators. In: Proceedings of the 7th international conference on evaluation of novel approaches to software engineering (ENASE 2012). Wroclaw, PL: SciTePress; 2012. p. 156–60.

Zhang D, Ooi BC, Tung AKH. Locating mapped resources in web 2.0. In: ICDE; 2010. p. 521–32.