Social Web Mashups Full Completion via Frequent Sequence Mining

Abderrahmane Maaradji, Hakim Hacid, and Ryan Skraba
Alcatel Lucent Bell Labs
91620 Nozay, France
Email: fname.lname@alcatel-lucent.com

Athena Vakali
Aristotle University of Thessaloniki
54124 Thessaloniki - Greece
Email: avakali@auth.gr

***Abstract*—In this paper we address the problem of Web Mashups full completion which consists of predicting the most suitable set of (combined) services that successfully meet the goals of an end-user Mashup, given the current service (or composition of services) initially supplied. We model full completion as a frequent sequence mining problem and we show how existing algorithms can be applied in this context. To overcome some limitations of the frequent sequence mining algorithms, e.g., efficiency and recommendation granularity, we propose *FESMA*, a new and efficient algorithm for computing frequent sequences of services and recommending completions. *FESMA* also integrates a social dimension, extracted from the transformation of $user \rightarrow service$ interactions into $user \rightarrow user$ interactions, building an implicit graph that helps to better predict completions of services in a fashion tailored to individual users. Evaluations show that *FESMA* is more efficient outperforming the existing algorithms even with the consideration of the social dimension. Our proposal has been implemented in a prototype, *SoCo*, developed at Bell Labs.**

***Keywords*-Mashups; Web services; Sequence mining; Social networks;**

I. Introduction

Web services composition has definitely contributed to developing new and existing applications for both academic and industrial communities [1]. Originally a task carried out by programming experts and developers, the emergence of Web 2.0 and the User Generated Services (UGS) paradigm enables end-users to create their own compositions of services. Nowadays, Web services have taken the form of Mashups, i.e. Web applications that combine existing services (API, data sources, etc.) into a single integrated service [2]. A popular Mashup example is the case of using cartographic data and interface (e.g. from Google Maps) with location information about real estate data.

Since creating Mashups is now an emerging trend, semi-automatic Mashup composition will assist in developing Mashups in a faster and user-tailored manner. Typically two categories can be distinguished: (i) *step-by-step completion*, in which a list of potential single services is suggested to the user on the basis of the currently selected service [3], and (ii) *full completion*, in which the whole composition (or a part of it) is recommended to the user [4]. Our work focuses on the full Mashup completion and aims at providing an answer to this problem through sequence mining for capturing, modeling, and suggesting the most interesting combination of services that should follow a current service creation flow. The goal of the proposed approach is to improve Mashups creation time and quality while dealing with the following challenges:

- *scalability*: the number of potential candidates to a full completion is combinatorially larger than the number of candidate services to step-by-step completion. This directly impacts the system's scalability.
- the terminating condition in suggesting a full completion has no specific boundary since the number of required services to complete the Mashup is not known a-priori. Therefore, recommendation of services involves an unknown parameter that increases the complexity.
- recommendation detail level given the wide variety of different services and resources that are available for the system for suggestion.

The problem we are tackling could be summarized as follows: *Given a user creating a Mashup within a Mashup creation platform, how can the platform suggest the finished Mashup that best meets her intentions, within a reasonable amount of time?* Our work addresses this problem by identifying frequent combinations of services and capturing users' social interactions over them. This approach is used for predicting and suggesting the next services that will complete an initiated Mashup by exploiting both co-occurrence frequencies and social interactions on earlier composed services. More specifically, such Mashups full completion strategy involves :

- modeling the problem of full completion as a frequent sequence mining problem. Thus, we show that existing frequent sequence mining techniques could be leveraged to provide a solution to this problem.
- dealing with scalability issues related to the existing frequent sequence mining algorithms. This is performed via the introduction of a new frequent sequence mining algorithm, called *FESMA*. *FESMA* offers a high performances in terms of computation time outperforming the existing algorithms in our context.
- personalized completions achieved through the introduction of a social dimension in the process. The social dimension is essential to this work since in Web 2.0, people can create, use, and share services. We assume that Mashups environments reflect the social behaviors of users and thus, social structures can be extracted from the interactions between users and services (and between users). These interactions can be analyzed and injected as a social information into the process of full completion for services discovery and composition.

The proposed approach is integrated and implemented in a social-oriented composition framework named *Social Composer (SoCo)*, developed at Bell Labs [3].

The rest of this paper is organized as follows: Section II introduces a very simple illustrative example and discusses the principles of Mashups full completion. Section III discusses our proposal to frequent sequences mining for

Mashups full completion. Section IV discusses two strategies for suggesting completions: community-based and social-based. Section V presents the evaluation results on different datasets as well as a comparative study with existing frequent sequence mining techniques. Section VI reviews some related work. Finally, we conclude and provide some future directions in Section VII

II. Context & motivating example

The recent acquisition by Alcatel-Lucent of ProgrammableWeb[1], one of the largest API (Web services) and Mashups Web repositories, is a clear intention of the company to implement its *"Application Enablement"* strategy and have a successful transition in the Telecommunications and the Web convergence. ProgrammableWeb may eventually offer a user friendly (Yahoo! Pipes-like) tool to leverage its leadership and stimulate larger number of users to easily create Mashups.

To illustrate the proposal of this paper, consider three users Alice, Bob, and Carol who compose services using their favorite Mashup creation environment. Alice and Bob, who are more comfortable composing services, create some Mashups.

In order to find the definition of a word in English, translate into French and then email it, Alice creates a Mashup composed as follows: $Dictionary \rightarrow Translator \rightarrow Email$. Moreover, in order to find the weather forecast, translate it and receive it by SMS, Alice creates a new Mashup as follows: $Weather \rightarrow Translator \rightarrow SMS$. On his side, Bob would like to create a Mashup that finds the weather description from his location, send it on his blog, and then post a tiny URL of his blog on his Twitter profile. Thus, he creates the following Mashups: $Mylocation \rightarrow Weather \rightarrow BlogPost \rightarrow tinyURL \rightarrow PostTwitter$.

Carol would like now to create a new Mashup based on weather forecast service. Once she selects the weather service, she gets suggestions (i.e., completions) based on other users' usage: (i) $\rightarrow Translator$, (ii) $\rightarrow Translator \rightarrow Email$, (iii) $\rightarrow BlogPost$, (iv) $\rightarrow BlogPost \rightarrow tinyURL$, and (v) $\rightarrow BlogPost \rightarrow tinyURL \rightarrow PostTwitter$. However, since she doesn't really have experience with these services, Carol wishes that the completion list would preferentially rank completions related to her. If we suppose that Carol is somehow close to Alice (e.g., share common interests), it is more likely that the system should prefer a recommendation originating from Alice rather then from the whole community. Suppose that Carol selected the completion (i) $\rightarrow Translator$. After selection, the completion list is updated dynamically. It will offer as following to $Weather \rightarrow Translator$ two completions: $\rightarrow sendSMS$ or $\rightarrow Email$, and so on until Carol chooses to terminate the Mashup.

Basically, the idea of composition completion is to predict the remaining part of a Mashup during its creation by a user, given its current state. Figure 1 illustrates three Mashups created by different users in which different services are combined together to achieve specific goals. It can be easily observed that there is a recurring configuration of services that appear together in the different compositions. We illustrate one of them represented by the chain: $w_3 \rightarrow w_4 \rightarrow w_5$. Since this configuration is repeated, it would be interesting to suggest it whenever similar composition schemes are started. Figure 2 illustrates this principle by considering the initial status, called the *query sequence*, of the current Mashup as a composition of $w_1 \rightarrow w_2 \rightarrow w_3$ respectively. The following chains of services are those predicted by the system which are expected to be the most suitable ones to complete that initial configuration.

[1] http://programmableweb.com/

III. From services to sequences of services

A. Data model

Intuitively, since services composition is based on the combination of different services together where the output of a service w_i is the input of service w_{i+1} (or a part of it) immediately following w_i, this builds a chain of services following different patterns. Thus, Mashups (and services compositions in general) can be considered as sequences of services[2]. Let $W = \{w_1, ..., w_n\}$ be a set of n items ($|W| = n$) that we explicitly call Web services from now on. Let $S = \{s_1, ..., s_m\}$ be a set of m sequences ($|S| = m$). A sequence, representing a Mashup in our case, is defined as an ordered set of services denoted by $s_i(w_1 \rightarrow w_2 \rightarrow ... \rightarrow w_k)$ where w_i $(i = 1, ..., k)$ represents the i^{th} Web service.

For each sequence we associate: (i) the length k, denoted $len(s_i)$ defining the number of successions of services included in the sequence (i.e., with repetition) (ii) a set $pref(s_i)$ representing the set of prefixes of a sequence s_i. As commonly used for strings, a prefix represents a subsequence having as a first service the first service of s_i with a length less than $len(s_i)$. As an example, let's consider the sequence $s_i(w_1 \rightarrow w_2 \rightarrow w_3)$ then $pref(s_i) = \{(w_1), (w_1 \rightarrow w_2)\}$. Finally, we associate a set $suf(s_i)$ representing the set of suffixes of a sequence s_i which represents all the subsequences having as a last service that of s_i. As an example, let's consider the sequence $s_i(w_1 \rightarrow w_2 \rightarrow w_3)$ then $suf(s_i) = \{(w_3), (w_2 \rightarrow w_3), (w_1 \rightarrow w_2 \rightarrow w_3)\}$.

The basic idea is to identify and count recurrent subsequences in compositions that have been previously created by users within the system. Those frequent sequences represent actually, on one hand, the composition behaviors of each individual, and, on the other hand, the common habits and behaviors shared implicitly between groups of users. The problem becomes to find frequent subsequences having the same prefix as the query sequence of a certain length $k \in \{1, ..., max(k)\}$.

The investigation of existing algorithms for sequence mining (cf. Section VI) has shown that (i) no algorithm had significantly better performances than others [5], and (ii) algorithms performances are heavily tied to the nature of the dataset that has been used to measure it. In our context, the first priority was given for the scalability since we need to process completion queries in real time and better personalized completion suggestions. This is not offered by the existing algorithm. We propose then a new algorithm for frequent sequence mining to tackle at a first stage, the scalability problem.

[2] This should not be confused with the composition patterns that include sequence operations, parallel operations, etc. but rather the way services are modeled at a logical level.

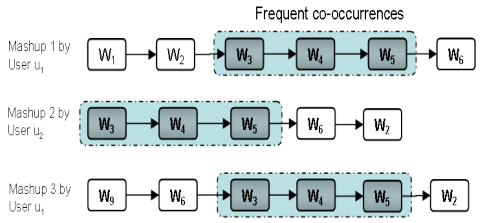

Figure 1. Example of services co-occurrences in different compositions

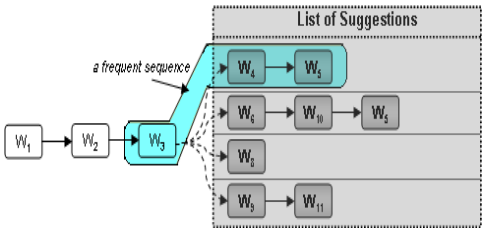

Figure 2. Illustration of full completion applied to the example of Figure 1

B. A New Algorithm for Fast and Efficient Frequent Sequence Mining

The algorithm we propose is called *FESMA* for *Fast and Efficient Sequence Mining Algorithm*. Just like the *FP-growth* algorithm [6], *FESMA* doesn't generate any candidates and uses a compact prefix tree representation to store all sequences (i.e., sub-Mashups) that exist within the transactions database (i.e., all created Mashups). By contrast to other sequence mining algorithms, *FESMA* scans the database only once. During this scan, and for each transaction, all sequences are added to the tree representation by updating the support associated with each sequence and the user's specific supports. We named that tree *FSTree* for *Frequent Sequences Tree*. Algorithm 1 summarizes the general process[3].

Algorithm 1 FESMA

Require: $S = \{s_j, j = 1, ..., m\}$ {list of sequences}
1: **for** $j = 1$ *to* m **do**
2: {scan all Mashups}
$s_j := GetSequence(j)$
3: **for** $k = 1$ to $len(s_j)$ **do**
4: {parse all subsequences of s_j}
$SSeq := subsequence(s_j, k)$
5: **if** $SSeq \in FStree$ **then**
6: Update the corresponding branch by incrementing nodes support in $FSTree$
7: **else**
8: Create a branch and set its node support to 1 {update the prefix tree}
9: Update users supports
Ensure: Return $FStree$

Table I
ILLUSTRATION OF MASHUPS DATABASE

ID, user	Transactions
$Mashup_1$, user 'a'	$w_1 \rightarrow w_2 \rightarrow w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6$
$Mashup_2$, user 'b'	$w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6 \rightarrow w_2$
$Mashup_3$, user 'a'	$w_9 \rightarrow w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_1$

In order to illustrate the algorithm, let's consider the simple example of Table I which shows three Mashups and their associated users as input. As the algorithm visits every Mashup in the database, the *FSTree* is updated to keep a current count of all the subsequences encountered, as follows: for every possible suffix of the current Mashup, a path corresponding to that suffix is followed through the $FSTree$ incrementing the value of existing nodes that are visited, and creating new nodes if necessary (with a value of 1) to finish the path. For instance, let's consider $Mashup_1$ of user 'a'. In order to represent in the $FSTree$ with all subsequencess generated from $Mashup_1$, we actually need just to update it with the following subsequences: $(w_1 \rightarrow w_2 \rightarrow w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6), (w_2 \rightarrow w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6), (w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6), (w_4 \rightarrow w_5 \rightarrow w_6), (w_5 \rightarrow w_6), (w_6)$. For illustration, e.g., when updating the $FSTree$ with $(w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6)$, this will actually represent in the tree with $(w_3), (w_3 \rightarrow w_4), (w_3 \rightarrow w_4 \rightarrow w_5), (w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6)$. This process is repeated on the whole sequences (all Mashups). The tree illustrated in Figure 3 is provided as an output.

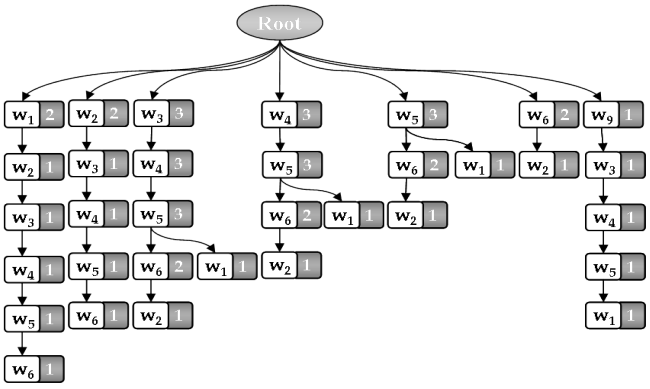

Figure 3. Illustration of the output tree after the execution of FESMA on the example of Table I

From the *FESMA* algorithm definition, we can see that one needs exactly one scan of the database to parse existing Mashups (i.e., transactions). The cost of parsing the database is $O(m)$, where m is the size of the database. In order to update the sequence tree $FSTree$ by subsequences, each transaction is parsed once. The cost of inserting a sequence in the tree depends on the sequence length (depth of the tree). In the worst case, this operation costs $O(K^2)$ with K corresponding to the size of the longest sequence. In summary, the overall complexity of the algorithm in the worst case is $O(m \times K^2)$.

IV. A NEW APPROACH FOR FINE GRAINED FULL COMPLETION PREDICTION

At this stage, we have succeeded in adapting the frequent sequence computation and making it more efficient via a faster computation and limited database scans. In this

[3]Note that Line 9 of the algorithm is responsible of integration the social dimension in this process. This feature is described later but we prefered to integrate it at this stage for completeness matters.

section, we focus on the use of the generated representation and the computed sequences. Intuitively, when processing the set of sequences using *FESMA* or otherwise, the only information we have is the frequencies of subsequences, providing a strictly global perspective for possible completion strategies. In other words, since the co-occurrences are computed according to their appearances over all existing sequences, this process considers only the aggregation of the behavior of all existing users regarding the most popular sequences. Thus, any assistance can only operate at a high level of granularity, i.e., community or global, equally valid for one user as for another, yet customized for neither.

In the following, we describe an enhanced community-based strategy for ranking the completion lists which improves on this global perspective of the direct application of frequent sequence mining algorithms to the full completion problem. Afterwards, we introduce and motivate a fine grained strategy based on social networks implicitly extracted from the analysis of interactions between the entities of the system.

A. Community-based recommendation

This functionality can be achieved by using any frequent sequence mining algorithm. At this stage, it is necessary to keep in mind that we are aiming at offering support for end-users (e.g., under the form of recommendations) to easily build her Mashup. Applying the aforementioned algorithms produces a set of subsequences with their frequencies as defined in Formula 1:

$$S' = \{(s'_i, freq(s'_i))/s'_i = (w_1 \rightarrow ...w_l) \wedge l \leq Argmax(len(s_i))\} \quad (1)$$

Where $freq(s'_i)$ is the frequency of subsequence s'_i in the initial set of sequences S, and $Argmax(len(s_i))$ is the length of the longest sequence in the initial set playing the role of highest limit, i.e., it is not possible to find a subsequences longer than the longest sequence in the initial sequence set. Depending on the algorithm used, this output could be represented and indexed as a tree. A query sequence s_q is sent to the system in the form of a service or a sequence of services (i.e., built from an initial successive combination of services). The system selects candidate sequences from S', where the prefix of the candidate subsequence is a suffix of the query sequence. All selected subsequences represent potentially interesting answers for completing s_q. At this stage, according to a predefined strategy, the recovered sequences are ranked by their relevance and only the $top-k$ sequences are proposed to the user. Algorithm 2 provides an abstract description of the completion process.

Basically, the full completion algorithm cost depends on the length of the query sequence $|s_q|$. In fact, for each suffix of the sequence query, the algorithm retrieves completions from the frequent subsequences list. This makes the use of traditional frequent sequence mining algorithm unsuitable in this context[4]. Our alternative approach uses the $FSTree$ representation which can be traversed with more efficient computation and access times. Once the branch of the sequence query suffix is retrieved, one needs just to browse that branch to access the most frequent sequences (with additional "meta-data" if it exists).

[4]The execution times of existing algorithms are discussed in the evaluation section.

Algorithm 2 Completion abstract algorithm

Require: s_q: {*the query sequence*}
S: {*the set of existing Mashups*}
1: $S' = FSM(S)$ {*mine frequent sequences*}
2: **for all** each $sq'_i \in suf(s_q)$ {*all suffixes of the query*} **do**
3: $TempList$ = Select s'_i from S' where sq'_i is prefix of s'_i
4: $RecList \Leftarrow RecList \cup TempList$ {*building the recommendation list*}
5: Rank $RecList$
Ensure: Return $top-k$ elements of $RecList$

B. Social networks based recommendation

As we are clearly in a Web 2.0 environment, we believe that the user needs to be introduced in the process not only as a separate entity or a group of people but as an interlinked entity with other entities following a relation translating, e.g., common interests and friendship. We believe that this will lead fine grained, more precise and personalized support for users.

We consider interactions that involve end-users as social interactions, and part of the social dimension. The remaining type of interactions, i.e., those between services, is considered as a structural support for the approach since such information is necessary to, e.g., ensure that the input of service w_{i+1} is compatible with the output of service w_i.

Frequent sequence mining algorithms, even *FESMA*, don't consider a fine grained level of granularity since (i) they mainly operate at a global level and (ii) they reason about one type of entity, i.e., services. Thus, they need to be adapted not only to keep track of social information but also to support the high number of possible combinations due to the introduction of the user in the process. A social network in this context is then defined as an abstraction of interactions that occur between people and services in Web services environments, capturing the behavior of social entities in the form of a social graph. This structure may be inferred or extracted directly from common interests between the users of the composition platform. The principle is based on the transformation of $user \rightarrow services$ interactions to a $user \rightarrow users$ social network on top of which statistical processes are applied to, e.g., fire recommendations for assisting the user in constructing the Mashup. Impacts and interests of the social dimension have been introduced in [3] and were heavily discussed. Since the objective here is to show how this dimension is leveraged for building sophisticated full completion strategies, we don't detail this aspect further in this paper.

With this new constraint, the method has to enumerate and count not only the frequency for each subsequences, also named here support (i.e., number of occurrences), but the specific sequence support for each user. In other words, each node is related to the users who have used the subsequence it represents. This information is associated in the form of an array capturing: user u_i has used subsequences s_j, l times. In order to reduce the construction cost, this

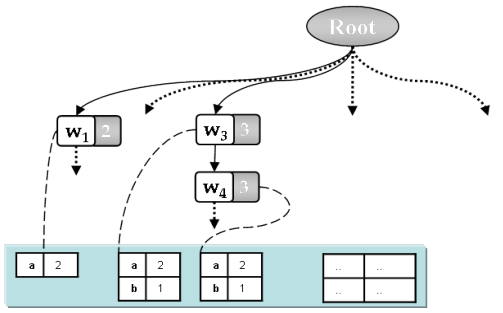

Figure 4. $FSTree$ with social information

information is updated while building the tree. The result is illustrated in Figure 4. Thus, Formula 1 needs to be revisited to incorporate this level of granularity, as in Formula 2:

$$S' = \{(s'_i, u_j, freq(s'_i, u_j))/s'_k \in S' \wedge u_j \in U\} \quad (2)$$

In terms of algorithm complexity, adding the users' specific sequence occurrence within the FESMA sequence mining algorithm, i.e., Algorithm 1, impacts not impacts not only the algorithm computation resources but also the memory space occupation.

In order to consider that social dimension, we propose in the following an efficient strategy for a social-based full completion approach based on the construction of an implicit *social graph* between users. We consider the resulting graph as social since it captures the behavior of users regarding services composition and their potential common interests.

Besides the $users \rightarrow (single\ services)$ relationship, we consider $users \rightarrow sequences$ interactions as a bipartite graph [7] that represents how frequently users include sequences in composition schemes. The links represent the usage frequency which a user u_i has of a sequences s'_i in all the compositions she created. To transform the bipartite graph into a social graph to help rank recommendations we rely on three main steps: (i) local information extraction, (ii) semi-global information extraction, and (iii) global information extraction.

Local information extraction: the *local information* considers only the interaction between a specific user and a specific sequence. This information tells us whether a specific user is confident (i.e., expertise indicator) using this sequence among other sequences. To materialize this idea, we define this information in a quantity called *Activity* defined in Formula 3 where M is the total number of sequences a user u_i exploited in her different compositions.

$$Act(u_i, s'_i) = \frac{f(u_i, s'_i)}{\sum_{k=1}^{M} f(u_i, s'_k)} \quad (3)$$

Semi-global information extraction: At the level of semi-global information, we consider the interest a user may have in other users regarding a given sequence. Thus, for a given user u_i we calculate how much the sequence s'_i recommended by the user u_l matters to her. This is called *Special Interest* (SI) and is calculated using Formula 4.

$$SI(u_i, u_l, s'_i) = \frac{f(u_l, s'_i)}{f(u_i, s'_i)} \quad (4)$$

Global Information extraction: In order to have as precise transformation as possible with less data loss, we add another level of information in the transformation process. The global information captures whether a couple of users have common general interest or not. At this stage of our study, and for simplification reasons, we consider that the general interest of a couple of users is equal to the sum of their specific interests, thus building the implicit graph as illustrated in Formula 5. The output of this step is a users' social graph aggregating all the specific interests graph obtained previously.

$$IG(u_i, u_l) = \sum_{k=1}^{M} SI(u_i, u_l, s_k) \quad (5)$$

Sequences recommendation strategy: Once the bipartite graph is transformed to a social graph thanks to the three previously described steps, we proceed to recommendation calculation to suggest a coming sequence according to a selected query sequence. Thus, considering the intrinsic user's usages frequency (local information), the specific interest between two users (semi-global information), and the implicit graph (global interest between users), we define the *Recommendation Confidence* RC of a given sequences s'_i according to the introduced sequence s_q (considered as prefix of s'_i) for the user u_i as follows (see Formula 6):

$$RC(u_i, s_q, s'_i) = \sum_{l=1}^{N} SI(u_i, u_l, s'_i) \times Act(u_l, s'_i) \times IG(u_i, u_l) \quad (6)$$

The recommendation confidence RC is the metric that indicates how a completion is important to the user. Concretely, when the user u_i is creating a new composition of services, and has entered s_q as prefix for Mashup full completion, a ranked list of recommended Mashup completions is proposed in decreasing order of recommendation confidence RC.

V. Experimental study

We have performed mainly two kinds of evaluation: (i) a comparison evaluation in which we compare the performances of our approach to existing frequent sequence mining algorithms, and (ii) an evaluation of particular properties of *FESMA* to measure the overhead generated by the consideration of the social dimension.

Being a succession of services, Mashups have their own statistic properties, e.g., their distribution and their length (according to analysis of available data on ProgrammableWeb [8]). Thus, the dataset which can be used need to respect the behavior of real world observations. On the other hand, another important aspect, especially when comparing to other methods, is to select datasets which are supported by existing approaches. We have decided to use the synthetic data generator from "*IBM quest data generator*". For instance, IBM-Artificial dataset $T10k - L5$ contains 10^5 transactions (defining Mashups in our case) and the average sequence length is equal to 5.

Generally speaking, the main performance criteria used to evaluate this kind of methods are: (i) the execution time and (ii) the memory space required by each algorithm to

find frequent sequences in a dataset. It should be noted that in the case of *FESMA*, this time includes reading the dataset from an input file and writing results in an output file (costly operations in terms of time). *FESMI* is implemented using standard C++ library. Finally, all the test that we have performed are done on an Intel Core 2 Duo T9600 With 2.8GHz of processor and 3GB of RAM.

Table II
LIST OF DATASETS

Dataset name	Number of transaction	Transactions average length
T100k-L2.5	10^5	2.5
T200k-L2.5	2×10^5	2.5
T500k-L2.5	5×10^5	2.5
T1000k-L2.5	10^6	2.5
T100k-L5	10^5	5
T200k-L5	2×10^5	5
T500k-L5	5×10^5	5
T1000k-L5	10^6	5
T100k-L10	10^5	10
T200k-L10	2×10^5	10
T500k-L10	5×10^5	10

A. FESMA Vs state of the art algorithms

In order to compare FESMA to state-of-the-art frequent sequence mining algorithms, we run it over a bunch of datasets. Table II shows a list of datasets used to evaluate the proposal with a comparison to existing algorithms. We illustrate the comparison results between FESMA and Apriori-seq [9] on 3 different datasets represented in Figures 5, 6, and 7 respectively[5]

Regarding the obtained results, it appears that there is a clear gap between the results obtained by *AprioriSeq* and *FESMA* with a better performances for *FESMA* on all the configurations of the support (minimum frequency) represented on x-axis. We believe that with these results, even other algorithms will be outperformed. Another interesting observation regarding *FESMA* is its ability to manage very short frequent sequences which generally constitutes a problem to existing techniques because of their large number. Finally, it can be easily observed that *FESMA* is stable after considering a minimum support of 2 services. This means that the support doesn't influence the performances of the method too much contrary to the existing methods.

Once this information checked, and since we could not use larger datasets with the implementation of the *AprioriSeq* that we have, we wanted to check the scalability of the proposed approach. We have considered the same datasets described in Table II. The results are illustrated in Figure 8. Considering the size of the datasets and the minimum support, the results are satisfactory since the maximum time needed to build the tree with 10^6 rows is about 90 seconds. Note also the behavior of the algorithm which reproduces exactly the same stability for all the situations like the one observed before.

[5]We could not reproduce the experiments using other algorithms due to some technical issues related to the code available on different Web sites. We could not even report the obtained results on the literature, even with the use of the same datasets, since the hardware configuration is not the same since it's useless for comparison.

B. Social-dimension integration cost

As a second experiment, we wanted to measure the overload generated by the integration of the social dimension within *FESMA*. To evaluate this, we have modified an initial dataset, i.e., $T1014D100K$, by associating to each sequence a user identifier who is supposed to be the creator of such sequence (i.e., Mashup). We have generated $User \longleftrightarrow Mashups$ association satisfying the most important property of social networks, i.e. the long tail of the activity distribution. This property argues that some users (Web-users) are much more active than others in terms of generated content (Mashups). Figure 9 illustrates the obtained results.

The result clearly show that the algorithm's runtime responses keep the same behavior with an average of 25% of overload for social dimension which is reasanable regarding the personalizarion and social added-value features provided to users. In the same time, even with the overhead generated by the social dimension, the results are more interesting than all the existing algorithms without the consideration of the social dimension.

C. Completion response time evaluation

At this stage we wanted to measure the response time of the completion strategy. Indeed, the completion strategy needs to satisfy interactive application requirements since it's supposed to provide real-time and dynamic recommendations and suggestions to users who are creating (editing) Mashups. As mentioned before, the completion algorithm uses the frequent subsequences tree *FSTree* generated by *FESMA* in order to retrieve the remaining piece of a sequence introduced by the user.

To perform this evaluation, we construct on the previous dataset and $FSTree$ and associate for each frequent sequence its users. Then, we consider different initial queries by different users (randomly selected from the database) while varying the size of each query sequence from 1 to 10, i.e. $len(s_q) \in \{1, ..., 10\}$. We recover then the maximum time for each value of the size. The results are illustrated in Figure 10 with time unit expressed in milliseconds. The results shown in Figure 10 illustrates the efficiency of the proposed completion strategy and its ability to support real-time query (less than 0.1 second). Note that in a realistic context, network latency need to be considered as well. Moreover, as it should be expected, this figure shows that more the length of a query sequence is high, more response time decreases (because there is less choices of completion).

VI. RELATED WORK

In the following, we discuss the two main related topics: (i) end-users service composition and full completion, and (ii) frequent sequence mining.

A. Mashups full completion

In Greenshpan et al. [4], the authors rely on services categories to compute completions of Mashups using a top-k strategy. This is, to the best of our knowledge, the only work in the literature which is directly related to Mashups completion. Our work has the same objective as that in [4] but from a different perspective. We consider not only the community level but especially the individual level (how the end-users are related in social network) to compute

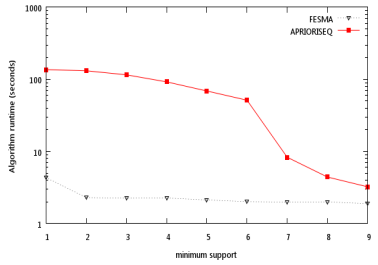

Figure 5. FESMA vs Apriori runtime over support (log-lin scale) on $T100kL2.5$ dataset.

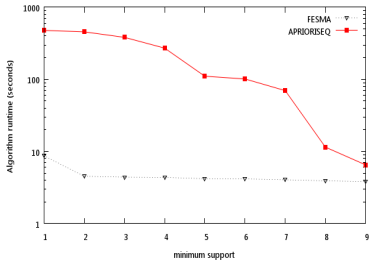

Figure 6. FESMA vs Apriori runtime over support (log-lin scale) on $T200kL2.5$ dataset.

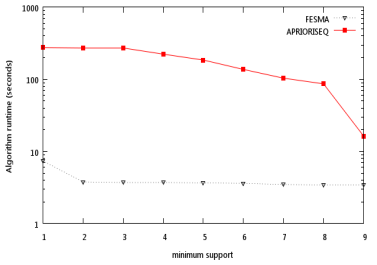

Figure 7. FESMA vs Apriori runtime over support (log-lin scale) on $T100kL5$ dataset.

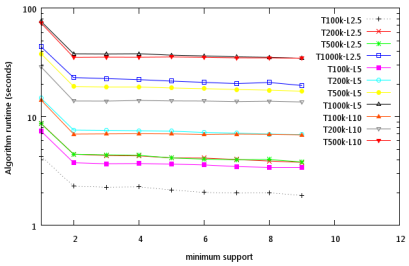

Figure 8. *FESMA* runtime on all the datasets

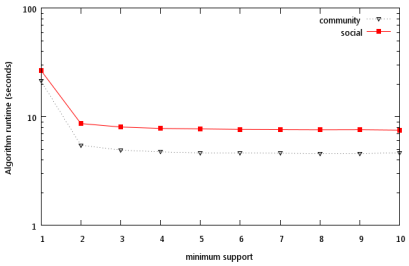

Figure 9. Overhead generated by the social dimension

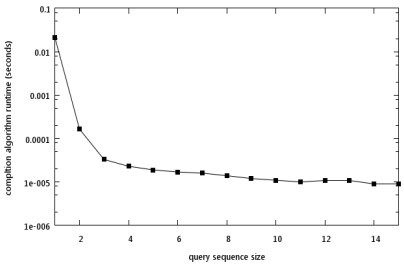

Figure 10. Obtained results on completion times

completions. Another difference is that our calculations are performed using specific services within the compositions as opposed to broader service categories. Instead of requiring more resources, the computational efficiency of our approach adequately copes with the increase in precision. The scalability of our approach is advantageous, using datasets five times larger and with more precision than in [4].

There have been other efforts to understand social phenomena in Mashups platforms, illustrating of the increasing integration of the social dimension. Through a use-case approach, Floyd et al. [10] highlight the APIs proliferation on the Web in parallel with the number of creative Web users. The study shows the benefits of collaboration between end users and developers that combines the creativity of end users with the expertise of developers. Automating this process is the important challenge we're addressing. In that regard, an interesting study [11] describes the interactions of Yahoo! Pipes' users. This can be used to extract social structures based on an analysis of user interactions.

Soriano et al. [12] emphasize the growing importance of the user-service relationship in a Service Oriented Architecture for composing services. The authors introduce EZWeb, an environment for sharing Mashups between colleagues, as a basis for co-production in an enterprise context. In addition, [13] emphasizes the phenomenon of what they call "social interaction" between services. The aspects of trust and reliability between services may impact the service selection for composition. Yu et Woodard [8] propose a very interesting view of the ecosystem of Mashups. This study on an API repository[6] shows that services usage follows a long-tail effect (power-law distribution), one of the major and interesting properties in social networks [14]. From the services recommendation perspective, some systems are based on user preferences (user profile) to suggest services [15]. Others rely instead on the concept of domain-specific knowledge expressed in a specific area (science, business, etc.) processed to extract rules that are used for building recommendations [16].

Our approach to service full completion is innovative since this is the first approach that offers fine grained recommendations. Moreover, our approach leverages both the community-based principles and a social dimension with a well balanced importance thus providing the user with well targeted and more personalized recommendations.

B. *Frequent sequence mining applications*

Mashup full completion is an emerging field that seeks to complete a composition of services supplied by the user. To the best of our knowledge, there is no study that addresses directly this issue (except [4]). Therefore, the problem has been compared to similar work in other areas leading to studies in many fields: words and phrases full completion, DNA sequence prediction, travel itinerary recommendation, and others dealing with sequence mining [17].

The most frequent case in full completion occurs in the context of search engines where full completion of words displays strings that are the most relevant to complete the introduced prefix (typed words or letters). Classical approaches, such as suffix trees [18], could not be used directly in our case. In fact, in words full completion, only whole words are considered in the training phase without taking into account sub-strings within those words. However, ideas coming from this field were a source of inspiration for different proposals. Typically, [19] has shown fault-tolerant full completion considering variants of the introduced prefix, a feature targeted as a future direction to our work. Another similar topic is predicting user actions based on user logs and preferences. For instance, [20] predicts *UNIX* commands that a user may enter based on previously entered sequences. As mentioned before, full completion is based on frequent pattern mining.

[6]http://programmableWeb.com

In the area of frequent sequence mining, many algorithms , issued from the area of frequent item sets mining, have been proposed like Apriori [21] and Eclat [22]. However, the most known algorithm for frequent sequence mining is SPADE [23]. SPADE has been defined for the particular case of frequent sequent mining. Similar to Eclat, SPADE uses a vertical representation of a sequence database with simple joins (intersection). Furthermore, it uses a lattice-theoretic approach to decompose the original search space in order to be processed separately in the main memory. SPADE algorithm scans the database only 3 times, leading it to outperform existing sequence mining algorithms as AprioriAll [24] and GSP [25]. *FESMA* as shown in the evaluations, has outperformed the existing algorithms for predicting the composition completion chains. Moreover, *FESMA* is unique in that it introduces the user dimension in the context of frequent sequence mining

VII. Conclusion and open issues

In this paper, we have proposed a new approach for Mashups full completion relying on frequent sequence mining and social networks analysis. To support the generally large amount of interactions that can occur in a Mashup environment, we have proposed a new fast and efficient frequent sequence mining algorithm, called *FESMA*. This algorithm integrates a social dimension that enables fine grained recommendations. *FESMA* can be used without the social dimension aspects, much like existing sequence mining algorithms. This has been demonstrated in the evaluation section that showed that *FESMA* outperforms the considered existing algorithms. However, as *Eclat* and *FP-growth*, the main constraint of *FESMA* is that intermediate data structures need to fit in the main memory. This currently doesn't cause any problem especially in the context of Mashups where on-line repositories are of a reasonable size (about $5k$ Mashups in ProgrammableWeb). Moreover, the completion algorithm (using FESMA) far fits interactive application requirement with few millisecond as response time. Even with the benefits of the proposed approach, there are still some issues that need to be considered. For future work, we mainly consider two issues: (i) the cold-start problem and (ii) the behavior of small sequences.

References

[1] M. ter Beek, A. Bucchiarone, and S. Gnesi, "Web service composition approaches: From industrial standards to formal methods," in *ICIW*, 2007, pp. 15–20.

[2] R. Ennals and M. Garofalakis, "Mashmaker: mashups for the masses," in *SIGMOD Conference*, 2007, pp. 1116–1118.

[3] A. Maaradji, H. Hacid, J. Daigremont, and N. Crespi, "Towards a social network based approach for services composition," in *IEEE ICC'10.*, may 2010.

[4] O. Greenshpan, T. Milo, and N. Polyzotis, "Autocompletion for mashups," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 538–549, 2009.

[5] B. Goethals and M. Zaki, "FIMI03: Workshop on frequent itemset mining implementations," in *Third IEEE ICDM FIMI Workshop*. Citeseer, 2003, pp. 1–13.

[6] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, 2000.

[7] J. Guillaume and M. Latapy, "Bipartite structure of all complex networks," *Information processing letters*, vol. 90, no. 5, pp. 215–221, 2004.

[8] S. Yu and C. J. Woodard, "Innovation in the programmable web: Characterizing the mashup ecosystem," pp. 136–147, 2009.

[9] F. B., "Trie-based apriori implementation for mining frequent itemsequences," in *ACM SIGKDD IW on OSDM'05*, B. Goethals, S. Nijssen, and M. J. Zaki, Eds., Chicago, IL, USA, August 2005, pp. 56–65.

[10] I. Floyd, M. C. Jones, D. Rathi, and M. B. Twidale, "Web mash-ups and patchwork prototyping: User-driven technological innovation with web 2.0 and open source software," in *HICSS '07*. Washington, DC, USA: IEEE Computer Society, 2007, p. 86.

[11] M. C. Jones and E. F. Churchill, "Conversations in developer communities: a preliminary analysis of the yahoo! pipes community," in *CCT '09*. New York, NY, USA: ACM, 2009, pp. 195–204.

[12] J. Soriano and al, "Enhancing user-service interaction through a global user-centric approach to soa," in *ICNS '08*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 194–203.

[13] Z. Maamar, L. Wives, Y. Badr, and S. Elnaffar, "Even Web Services Can Socialize: A New Service-Oriented Social Networking Model," in *INCOS'09*, 2009, pp. 24–30.

[14] S. Wasserman and K. Faust, *Social network analysis: Methods and applications*. Cambridge Univ Pr, 1994.

[15] T. Law, "Social Scripting for the Web," *Computer*, vol. 40, no. 6, pp. 96–98, 2007.

[16] L. Chen and al., "Towards a knowledge-based approach to semantic service composition," *LNCS*, pp. 319–334, 2003.

[17] G. Dong and J. Pei, *Sequence data mining*. Springer-Verlag New York Inc, 2007.

[18] E. M. McCreight, "A space-economical suffix tree construction algorithm," *J. ACM*, vol. 23, no. 2, pp. 262–272, 1976.

[19] S. Chaudhuri and R. Kaushik, "Extending autocompletion to tolerate errors," in *SIGMOD '09*. New York, NY, USA: ACM, 2009, pp. 707–718.

[20] B. Davison and H. Hirsh, "Predicting sequences of user actions," in *AAAI/ICML 1998 Workshop on PF AI ATSA*, 1998.

[21] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *ACM SIGMOD Record*, vol. 22, no. 2, pp. 207–216, 1993.

[22] M. Zaki *et al.*, "Scalable algorithms for association mining," *IEEE TKDE*, vol. 12, no. 3, pp. 372–390, 2000.

[23] M. Zaki, "SPADE: An efficient algorithm for mining frequent sequences," *Machine Learning*, vol. 42, no. 1, pp. 31–60, 2001.

[24] R. Agrawal and R. Srikant, "Mining sequential patterns," mar. 1995, pp. 3 –14.

[25] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," *EDBT'96*, pp. 1–17, 1996.