A Framework for Web Service Usage Profiles Discovery

Bruno Vollino, Karin Becker
Instituto de Informatica
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brazil
{*bruno.vollino, karin.becker*}*@inf.ufrgs.br*

***Abstract*—As part of web services life-cycle, providers frequently face decision about changes without a clear understanding of the impact on their clients. The identification of clients' consumption patterns constitute invaluable information to support more effective decisions. In this paper, we present a framework that supports the discovery of service usage profiles, to bring awareness on the distinct groups of consumers, and their usage characterization in terms of detailed service functionality. The framework encompasses monitoring of clients requests, constituting a general purpose Usage Database, and a process to cluster client applications and derive usage profiles. The paper details the framework and presents experiments.**

***Keywords*-web service; data mining; service usage;**

I. INTRODUCTION

Web services became vital for the business of many companies in the software industry, specially with the advent of the SaaS (Software as a Service) paradigm. As in any business, providers have interest in understand the needs of their clients to avoid customer attrition, and to attract new clients. Many providers focus on large scale service provision, and have very little knowledge about their clients. At the same time, they face hard decisions related to the maintenance of deployed services, service versioning and service redesign, without a clear understanding of the possible outcomes. Understanding the usage patters of clients is thus invaluable to support web service life-cycle [1].

Data mining has been applied in many business segments to discover knowledge about clients, which is hidden in large volumes of data [2]. Web service mining [3] aims at discovering patterns of service usage, i.e. specific ways in which web services are used repeatedly by a group of users with similar properties. Usage analysis have been used to support the recommendation of services [4] [5] [6] and compositions [3], the discovery of service composition communities [7], as well as processes documentation and conformance checking [8] [9]. Even when predefined interaction models are available, very often the reality differs of the expected behavior, justifying the deployment of sophisticated techniques to capture the actual usage patterns [9].

Our work [10] [11] is focused on the usage analysis as a support for the service evolution life-cycle. Our approach is to empower providers with an understanding of the overall impact of changes in the whole set of client applications, enabling sound decisions in terms of evolution strategies. Providers can leverage usage impact information to make decisions about the creation, maintenance and decommissioning of versions. For that purpose, they must have a clear understanding of the patterns involved in the overall requests clients make (the operations they request, the structure of the messages exchanged, co-occurrence of operations, among others), and leverage these patterns to group clients with a similar service usage behavior, which we refer to as *usage profiles*. We have explored usage profiles for the quantification of change impact in terms of affected clients [10] or financial metrics [11]. With the proposed mechanisms, the provider could identify which applications would be affected by an incompatible change (if any), and quantify this information in terms of broken applications, or financial losses due to client attrition or penalties. Other applications are service and process redesign, deployment provision maintenance, service recommendation, among others.

In this paper, we detail a framework that supports the development of a knowledge discovery process (KDD) [2] over monitored clients requests to derive usage profiles. The framework encompasses components for a) monitoring and logging of clients requests, b) inputting this data in a general purpose Usage Database, and c) applying a knowledge discovery process to derive usage profiles. The framework predefines tasks that requires minimum user intervention for the selection and transformation of relevant data, data mining using clustering techniques, and summarization of clusters as profiles. We present experiments based on synthetic data, simulating requests to a real service.

The paper contributes with techniques for identifying usage patterns that existing works on service mining (e.g. [3] [4] [5] [6] [7] [8] [9]) have not addressed yet, namely groups of clients based on detailed service functional properties. It details the ideas initially sketched in [10] [11].

The remaining of this paper is structured as follows. Section II presents the fundamental concepts underlying KDD and clustering. Section III presents an overview of the service evolution framework, and Section IV, the components of the Profile Manager. The KDD process to generate usage profiles is detailed in Section V. Experimental results are presented in Section VI and Section VII discusses related work. Section VIII presents conclusions and future work.

II. KDD and Clustering

KDD is a process targeted at discovering new, valid and useful information from large datasets [2]. This iterative and interactive process involves the steps of data selection and preprocessing, data mining, and evaluation of results. In the mining step, algorithms are applied to find patterns in data.

Clustering is a mining technique that groups data objects according to some similarity measure. Objects inside a cluster should have high intra-cluster similarity, and low inter-cluster similarity. The criteria for defining clusters depend on the nature of the data and the desired results, since distinct algorithms may output different sets of clusters. Algorithms that adopt distinct definitions of clusters may present conflicting results, and it is not possible to state that there is a superior technique.

The definitions of cluster (a group of similar objects) and clustering (the set of clusters derived from a dataset) can be used to classify the techniques over orthogonal dimensions [2]. A clustering may be classified as: *partitional*, where clusters are non overlapping subsets of the whole dataset; or *hierarchical*, where clusters may be nested, and organized in a tree structure. The clustering is *exclusive* if each object belongs to a single cluster.

Clusters may be classified as: a) *well separated*, where the objects inside a cluster are more similar to every other objects in the cluster than to any object outside it; b) *prototype-based*, where objects inside a cluster are more similar to its cluster prototype (e.g. centroid) than to any other clusters' prototypes; c) *density-based*, formed by contiguous objects in high density areas; and d) *distribution-based*, in which objects probably belong to a same statistical distribution.

Given a clustering, it is necessary to assess that its tendency is not a mere random structure, the number of clusters, and how well data objects fit together [2]. Assessment can be perfomed using supervised and unsupervised techniques. Supervised evaluation compares the discovered model to externally available information, such as a golden standard. Metrics such as the pair-counting F-Measure [12] can be applied to support such a comparison. However, in practice such a reference hardly exists, and the evaluation is made by an expert, with the help of unsupervised, internal indices. An internal index assumes a particular cluster definition, and it enables the comparison of clusterings and algorithms of the same type (e.g. to find the best parameterization). Silhouette, Cophenetic Correlation, and Dunn are examples of external indices [2].

III. Service Evolution Framework Overview

The service evolution framework [10] was proposed to support actions and decisions underlying service evolution, by considering the actual use clients make of services. It is composed of three modules, as depicted in Fig. 1.

The *Version Manager* is responsible for maintaining, in the Version Repository, a set of versioned service interface descriptions, and for assessing their compatibility. It adopts a fine-grained, feature-based versioning model [10], which allows versioning specific portions of a service interface description, relating the unaltered parts with previously created versions. A feature is a portion of an interface description, such as an operation, data type, or information related to the overall service. A service version is then represented by a graph of interrelated feature versions.

The *Profile Manager* aims at discovering usage patterns in the requests that clients issue for a service, and representing them as usage profiles, as detailed in sections IV and V.

The *Usage Manager* encompasses components that explore the profiles to assess change impact based on the actual use clients make of a service. In [10], we proposed profile-based metrics to quantify the impact of incompatible changes, and in [11], we explored usage profiles to measure the financial impact of changes.

IV. Usage Profiles and the Profile Manager

The Profile Manager has two main purposes: a) to automatically monitor service requests from clients to extract fine-grained data, and load it into a general-purpose Usage Database that suits many types of analysis; and b) to support the development of a KDD process to generate usage profiles, with the least user intervention possible. The latter is achieved by predefining the necessary tasks, which can be configured using simple parameters.

Usage profiles are representations of groups of client applications with similar usage patterns with regard to functionality described in the service interface. Such patterns describe the operations clients make use of, as well as the types of data they exchange.

The analysis of profiles in such a detailed level can reveal knowledge that suits many applications. For instance, awareness of which operations and types are actually in use may motivate providers to perform incompatible changes to improve service quality, which normally they would not consider due to the worst-case possibility of breaking clients. The knowledge of which operations are used together by relevant groups of applications may serve as a guide to redesign large service descriptions. Thus we include in the profiles as much information as possible, and let the provider explore it according to his/her analysis needs.

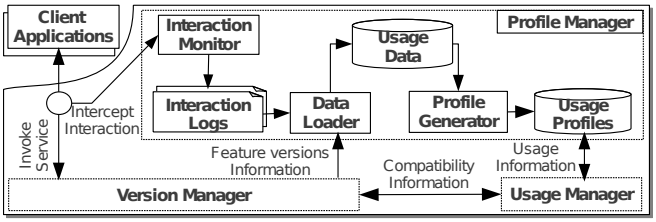

Figure 1. Service evolution framework.

Each profile (Fig. 2) is related to the applications from which the patterns were extracted, and to the feature versions they use. Metrics can be associated to applications (e.g. total number of requests) or feature versions (e.g. number of requests to an operation. Although we assume features to identify and describe profiles, the approach is relatively independent from any specific representation, and can be applied as long as smaller grained elements can be recognized from service descriptions.

A. Web Service Monitoring

The Interaction Monitor is responsible for intercepting and logging the messages exchanged between client applications and service versions they are bound to. The interception of service interactions is a challenging task, given the distributed nature of web services. Each alternative imposes distinct trade-offs in terms of scope of extractable data and performance of the monitoring capabilities, which must be carefully considered when determining where the logging infrastructure will reside in the web service architecture.

The service interactions may be intercepted [13]: in the HTTP layer, where the web server records the HTTP requests in logs; in the service application server, by implementing the adapter or interceptor patterns to handle messages; by adapters in the web services framework; in a proxy server or application, located either in the client [5] or provider side [8]; or hard coded in the web service itself.

Given our purpose, the Interaction Monitor has to be capable of intercepting and logging all operations requested, with the corresponding messages. These messages are usually documents exchanged by HTTP POST requests, which are not logged by web servers. Proxy servers or applications result in an overhead in the transport of messages and in the consolidation of logs. Hard coded solutions increase the costs of developing and maintaining the service.

Thus, the best option is to deploy interceptors in the application server or in the web service framework. With the latter, one can take advantage of the service framework to interpret the messages. Message handlers depends on the technology used, but are not affected by service evolution.

We assume that messages are exchanged in the SOAP format, and each service version has its own message handler. The handler registers the clients' requests in log files. We also assume that each web service version has a custom authentication mechanism, which associates a unique identifier to each application. It is a common practice of providers to request this unique identifier or some kind of access token as a parameter in its clients' requests.

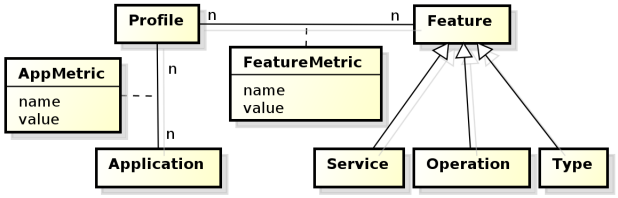

Figure 2. Usage profiles structure.

B. Data Loader and Usage Database

The Usage Database is a general-purpose, centralized repository that contains detailed data about service usage, and which suits different types of analysis. In this way, different criteria for defining the profiles can be experimented, as it will be described in Section V. The Data Loader is responsible for cleaning, interpreting and transforming raw data collected by the monitor and distributed in several logs, into the set of interrelated features involved in these interactions, as represented by the Usage Database.

The Loader needs to extract from logged raw data all features used by each client application, i.e. the service version, the operations requested and the parameters exchanged. This extraction is dependent on the message format logged. In the following we assume that a) the log registers the entire SOAP messages of requests and responses; and b) messages use literal encoding, which means that only the hierarchy of parameters and their values are provided, omitting the names of the operation and types (e.g. Fig. 3). By accessing the respective service description in the Version Repository, the loader identifies the operation requested, based on the parameters' names, and the used types, by recursing into the message hierarchy. Note that only the requests' structure (operations and types used) is required, not the actual data transmitted by the involved parties.

As illustrated in Fig. 3, the Loader (i) parses a request, (ii) retrieves the operation from the version repository, (iii) makes a recursive scan over the interaction parameters, identifying the used types, and (iiii) stores the processed data in the Usage Database. It also stores identifiers that

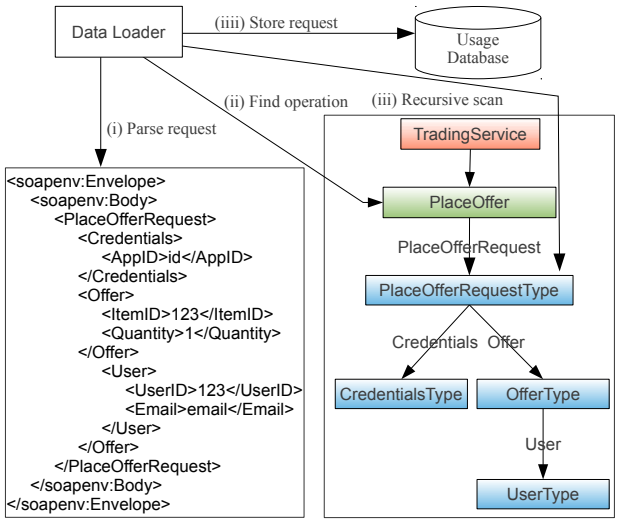

Figure 3. Process of extracting usage data from raw interaction data.

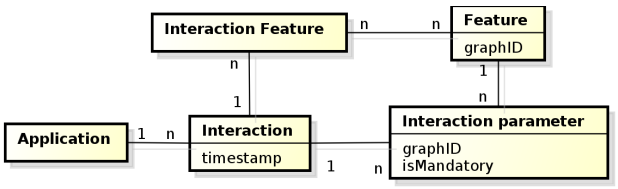

Figure 4. Usage Database schema.

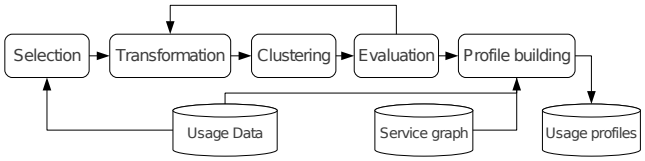

Figure 5. The tasks of the profile discovery workflow.

enables to relate, in both ways, the features in the Usage Database and the respective ones in the Version Repository. In this process the Loader discards all the invalid requests.

The Usage Database schema is depicted in Fig. 4. Every interaction (request or response) is performed by or targeted at an application. Services and operations are directly referenced by the interaction, which is represented by the 'Interaction Feature' relationship. The operation parameters and type parameters used in the interaction are represented by the 'Interaction Parameter' relationship. An identifier enables to associate each feature/parameter with its respective version in the Version Repository. Information of parameter optionality is also recorded.

C. Profile Generator

To hide the natural complexity of a KDD process to the users of the framework, the discovery of profiles is developed by parameterizing a set of predefined tasks, as depicted in the Fig. 5. The user: a) provides parameters to select data from the Usage Database that meets the analysis goals, b) selects among predefined data transformation alternatives, c) parameterize cluster algorithms and compare the results using metrics, and d) triggers the automatic generation of profiles for validated clusters.

V. Profile Generation Workflow

A. Data Preparation

Data preparation is crucial in profile discovery, because it influences how mining algorithms will cluster service clients. So, data must be carefully selected with regard to the business goals for defining usage patterns. Transformations should adjust selected data to the mining goals and the characteristics of the applied algorithms. Considering possible analysis goals, we have predefined tasks for data selection and transformation.

1) Data Selection: Data selection is driven by two parameters: time interval and data granularity. The service provider may be interested in the usage patterns with a temporal validity, such as last month, or since the last version released. So the selection component must be parameterized with initial and final timestamps, such that only the interactions within the specified time interval are selected.

The granularity refers to the level of detail used to cluster applications. The user can analyze usage either on the level of operations, or into more details, according to operations and data exchanged. In the first case, clients using the same operations are similar, whereas in the latter, they are considered similar according to the message structures exchanged. If operation level is chosen, the query to the Usage Database returns all the features in the relationship 'Interaction Feature' (Fig 4) that are used in at least one interaction in the defined time window. If the usage of types is additionally required, all types referred in the 'Interaction Parameter' (Fig. 4) relationship must be retrieved as well. Notice that only the variable part of requests involving a given operation must be retrieved, i.e. the optional parameters. If parameters are mandatory, at any level of recursion, their presence is implied by the mere usage of the operation, and therefore they can be disregarded.

To illustrate how relevant data types are retrieved, Fig. 6 depicts a service with operations Op1 and Op2, and their respective complex message structures defined in terms of 4 types. Dotted boxes denote optional parameters (i.e. P2, P6), and solid ones, mandatory. The type T1 is not selected, because the parameters P1 and P3 are mandatory, so they are always used in requests to Op1 and Op2. The type T2 is selected, because it is referenced only by the optional parameter P2. Thus, applications that request Op1 and Op2 with messages that include T2, are considered different from the ones that do not. Note that T4 is not selected, because it is also referenced by the mandatory parameter P5.

2) Transformation: Retrieved data must be transformed into a tabular format that summarizes how each application uses each selected feature. The rows represent the applications, and the columns, the features. Each row is thus an aggregation of all interactions of a same client with regard to the features. The user can select between two usage representations to fill the cells, namely *binary* or *weighted*. The former represents whether an application uses a feature

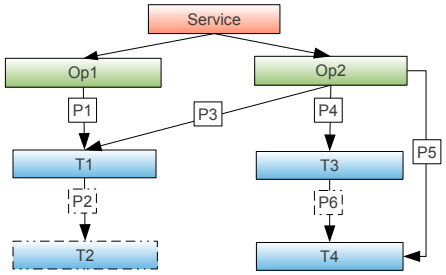

Figure 6. Mandatory and optional parameters.

(1), or not (0). The weighted representation adopts a measure for weighting how often a feature was used. The user can select in addition other types of transformations that may improve the results, such as normalization or dimension reduction [2] (e.g. eliminate features never used).

The profiles generated by each type of summarization answer very distinct analysis questions, and therefore the appropriate transformation should be selected. In the context of service evolution, profiles generated using the binary preparation are most valuable to *identify* which applications are not compatible with certain changes, because the clustering algorithms do not tend to split applications that use the same set of features over distinct clusters. In the weighted representation, applications are considered similar when they use similar sets of features with similar frequencies. With this representation, the clustering algorithm is able to distinguish between applications that use exactly the same set of features, but not in the same manner (e.g. heavy users of OP1 are not considered similar to eventual users). This type of profile is more interesting to *measure* the impact of changes over distinct groups of clients.

B. Clustering

As mentioned in Section II, finding which type of clustering algorithm better fits the data at hand is a challenge. Service usage data does not have *a priori* any particular property enabling the identification of the most appropriate clustering technique. Our aproach is to integrate in our environment several clustering algorithms, and to compare the resulting clusters through assessment metrics. In our current implementation, we adopted four algorithms of distinct classes: K-Means (partitional, centroid-based), Hierarchical agglomerative (hierarchical, well-separated), DBSCAN (partitional, density-based) and Expectation-Maximization (EM) (partitional, distribution-based). We are developing experiments to provide in the future parameterization guidelines.

Cluster assessment is also a challenge, because there is no information on the expected partitions. To develop experiments with synthetic data, we integrated in the framework the pair-counting F-Measure [12], which enables to compare clustering results against a golden standard. However, in real situations one would have to rely on an expert's knowledge of which clusters are valid and useful, with the help of internal indices. We are currently implementing several internal indexes, namely Silhouette, Cophenetic Correlation and Dunn [2], to develop experiments and compare their contribution to clustering assessment.

C. Profile Building

Clusters and profiles are distinct in nature. Clusters contain only the features that may be used to distinguish groups of applications, as result of the preparation step (Section V-A). Profiles, on the other hand, are an enriched representation of these groups of applications (Fig. 2). Thus, a profile includes all features used, together with metrics that indicate the importance of the group of applications, and of the features the group uses. If we consider the example of Fig.6, the features OP1, OP2 and T2 are submitted as input to clustering. A resulting cluster may indicate that only OP1 is used, without the optional parameter P2. Therefore, the profile would contain Service, OP1 and T1 (mandatory parameter P1), together with the respective metrics. Two metrics are considered (number of interactions per application and per feature), but others could be adopted as well.

```
1:  procedure BUILDPROFILE(cluster, initDate, finalDate)
2:    p ← Profile()
3:    instances ← cluster.instances
4:    p.apps ← processAppsAndMetrics(initDate,
5:                                   finalDate, instances)
6:    for all app from p.apps do
7:      for all attr from cluster.attributes do
8:        ft ← versionRepository.feature(attr.featId)
9:        if ft is Operation and
10:             instances(app).value(attr) > 0 then
11:         opUsage ← usageDB.numInteractions(
12:                       initDate, finalDate, app, ft)
13:         app.usageMap(ft) ← opUsage
14:         app.usageMap ← recurseParameters(
15:                        initDate, finalDate, app,
16:                        ft, app.usageMap, opUsage)
17:     for all (f, val) from app.usageMap do
18:       p.usageMap(f) ← p.usageMap(f) + val
19:   return p
```

Figure 7. The algorithm for building profiles.

To automatically construct a profile, metrics are calculated for all used features, by querying the Usage Database. These features are operations pointed as used in prepared data (non-zero values) and types of parameters used in requests for these operations, according to the service description (Version Repository of the Version Manager - Fig. 1).

The pseudo algorithm of Fig. 7 describes the procedure to be repeated for each valid cluster. From the instances of the cluster received as parameter, it computes, for each application, the total number of interactions performed (line 4), and the number of interactions related to the operation (lines 9-13). Then, it recurses over the operation parameters trees and types subtrees, computing the number of interactions in which each type appears (line 14). Note that mandatory parameters of operations are always used in every request, and their counting is derived from the respective operations. We need to retrieve the number of interactions for types of optional parameters, and for the types of parameters under them, in the service structure. Finally, the usage of features of each application is summarized in the profile (line 18).

VI. Experiments

The objective of our experiments is to demonstrate that the proposed framework can deliver useful service usage

profiles from an interaction log, with minor parameterization and evaluation of an expert.

In the absence of real interaction logs, we generated a synthetic log by simulating clients' requests over a real service, namely *eBay Trading*. This is a very popular service that supports a wide range of applications. Its interface is described by more than 150 operations and a thousand of data types. The service documentation organizes these operations in common workflows that can be used independently, or in combination to generate applications. We assumed that these workflows could be combined differently to characterize sets of applications with similar behavior.

Requests were created to represent predefined groups of clients, with some level of noise. The log was loaded in the Usage Database, from which we extracted datasets that varied in the level of detail (operation vs. types) and usage representation (binary vs. weighted). We developed the experiments using four clustering algorithms from Weka [14]: KMeans, EM, DBSCAN and hierarchical agglomerative with mean linkage. We have experimented with different parameterizations. Only the best results are reported here due to space limitations.

As a result, we expect to generate clusters that match the injected usage patterns, and to identify the best clustering algorithm(s) and parameterization for each type of dataset. The criteria used to evaluate clustering results are based on three aspects: the number of generated clusters; the number of distinct profiles represented by clusters, considering that a cluster represents its predominant profile (by number of applications); and the pair-counting F-Measure [12]. This supervised assessment metric reflects the homogeneity of applications inside clusters and the heterogeneity of distinct clusters, regardless the number of clusters.

A. Dataset

We adopted version 767 of the eBay Trading service, and 7 workflows representing common usage cases documented in the API guide[1]. We used JMeter[2] to generate requests to operations belonging to these workflows, according to some probability. Fig. 8 depicts the simulated profile *Buyer*, which has 5% of probability of executing the workflow "Get token", and 95% chances of executing "Buy item".

As summarized in Table I, we have simulated 525 applications distributed in 6 profiles, which performed 448,703 requests for 42 distinct operations. The Venn diagram in Fig. 9 shows the relationship between profiles, highlighting the common operations. Three of the profiles are proper subsets of others (P1.2, P1, P2), and two profiles (P6 and P8) use the same set of operations with different frequencies.

The generated log consisted of SOAP messages using literal encoding, which were preprocessed and loaded into the Usage Database, as described in Section IV-B. We report here three experiments based on different sets of selected and transformed data. We have also systematically added noisy applications to these prepared datasets, which have random values for the usage of features. The noisy applications were added in proportions of 10% and 30% with regard to the original number of applications (no noise). Data objects were labeled with the respective profile/noise class, such that clusters could be assessed using a supervised metric.

B. Clustering binary data

The first dataset involved only *operations*, prepared using binary representation. Profile P8 was excluded from the dataset because it is identical to P6 with regard to the binary use of operations. Results are displayed in Table II.(A), which shows the number of clusters (column C), the number of distinct profiles they represent (column P), and the F-measure (column F-M). The number of clusters and profiles ideally should be the same, i.e. each cluster represent a predefined profile. A higher number of clusters means that members of a same profile were spread (i.e. profiles are redundant), whereas a smaller number of profiles means that some clusters mix applications of distinct profiles. When F-measure is 1, it indicates perfectly clustered data objects.

In general, the hierarchical algorithm, using the mean linkage, yielded the best results. The clustering matched exactly the simulated profiles in the presence of any level of noise. Considering the dataset with no noise, 3 instances of the profile P2.2 were grouped together with P2 objects.

K-means and the EM were more sensitive to noise, not being able to detect subgroups of applications. They have mixed P1/P1.2 and P2/P2.2 data objects, and created clusters for noisy data. DBSCAN has matched almost exactly the

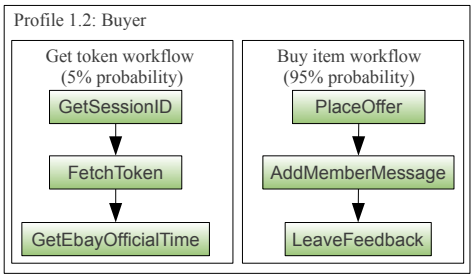

Figure 8. Example of a simulated application profile.

Table I
SIMULATION TOTALS.

Profile	Applications	Operations	Requests
P1	100	12	89,996
P1.2	50	6	44,017
P2	100	28	82,538
P2.2	25	31	20,841
P6	125	33	103,232
P8	125	33	108,079
Total (distinct)	525	42	448,703

[1]http://developer.ebay.com/DevZone/XML/docs/WebHelp
[2]http://jmeter.apache.org/

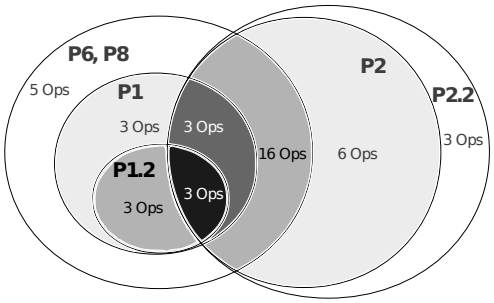

Figure 9. The simulated profiles and their intersections.

simulated profiles, but, as a density based algorithm, it tends to join in the wrong cluster some applications with small variations in relation to their profiles.

The second dataset varied by including *types* of optional parameters, in addition to the operations. Because the simulated log does not cover the use of types, we have inserted 2 new profiles in the prepared data: P6T, which includes the same operations as P6 and additionally 10 randomly selected types; and P2T, with the same behavior of P2, with additionally 10 randomly selected types.

As shown in Table II.(B), hierarchical clustering yielded the best results. K-means was not able to detect profiles with subset relations, mixing the profiles P2/P2.2/P2T, P1/P1.2 and P6/P6T. EM could distinguish P2 from P2.2 in the presence of noise, and the DBSCAN was able to detect all the distinct profiles, but the cluster representing P1 included applications belonging to neighbor clusters.

The experiments over this simulated data show evidences that the hierarchical agglomerative algorithm with the mean linkage yields good results for binary data. At a first glance, DBSCAN also seems to be a good choice, but it required extensive trial-error to find a good parameterization.

C. Clustering weighted data

The final dataset involved *operations* according to weighted representation for all profiles. Recall that P6 and P8 are distinguished only by usage frequency. We have normalized the usage values (number of interactions) using the z-score, a transformation that may reduce the effect of noise in some clustering algorithms.

The hierarchical algorithm with mean linkage yielded the best clustering. As displayed in Table II.(C), it has detected the six profiles in all cases, with the highest F-Measure values. However, it has always misplaced a few applications of P8 in the clusters representing profiles P1 and P1.2.

The K-means and EM algorithms could not distinguish profiles that differ only in the usage frequencies, clustering together P6 and P8 applications. They also could not detect subset relations: K-means merged profiles P1 and P1.2 in a single cluster, and EM merged P2 and P2.2.

The DBSCAN algorithm, using the two best parameterization we have found, failed to create one cluster per profile.

Table II
CLUSTERING RESULTS

A. Binary data in granularity of operations									
	Binary			10% noise			30% noise		
Algor.	C	P	F-M	C	P	F-M	C	P	F-M
K-Means	5	5	1.00	5	5	1.00	5	4	0.94
EM	5	5	1.00	5	4	0.94	5	4	0.94
DBSCAN	5	5	0.97	5	5	0.97	5	5	0.97
Hierarc.	5	5	0.99	5	5	1.00	5	5	1.00
B. Binary data in granularity of types									
	Binary			10% noise			30% noise		
Algor.	C	P	F-M	C	P	F-M	C	P	F-M
K-Means	7	6	0.94	7	5	0.86	7	6	0.95
EM	7	7	1.00	7	6	0.95	7	5	0.86
DBSCAN	7	7	0.94	7	7	0.94	7	7	0.94
Hierarc.	7	7	0.99	7	7	1.00	7	7	1.00
C. Weighted data in granularity of operations									
	Weighted			10% noise			30% noise		
Algor.	C	P	F-M	C	P	F-M	C	P	F-M
K-Means	6	4	0.78	6	4	0.72	6	4	0.52
EM	6	5	0.88	6	5	0.95	6	5	0.95
DBSCAN (0.5,3)	7	6	0.93	7	6	0.93	7	6	0.93
DBSCAN (0.5,7)	6	6	0.89	6	6	0.89	6	6	0.89
Hierarc.	6	6	0.97	6	6	0.98	6	6	0.98

With minimum points = 3, P2 was split in two clusters, one of them melded with applications of the profiles P2.2, P6 and P8. With minimum points = 7, it was able to find 6 distinct profiles, but it also created a cluster of P2 with many other nearly applications of other profiles, a problem of the density based approach.

This experiment also shows evidences that the hierarchical agglomerative algorithm with the mean linkage also produces consistent clustering results for weighted data. It should be noticed that distinguishing clusters based on weighted data is a more challenging problem.

VII. RELATED WORK

Three types of service usage patterns are highlighted in [3]: users access, service composition, and business process. Access patterns are explored in [4], where clients are clustered according to historical QoS and similarity over service invocation, to build a predictive model for future users. The similarity of functional and non-functional queries, of the resulting service invocations, and invocations recency, are explored in [5] to recommend services. To improve service recommendation, [6] find similar users according to service invocation patterns, and enrich users interest by discovering association rules that relate services they use in combination. Dynamic service compositions are discovered in [7] from access patterns, by clustering groups of services in terms of the strenght of their interactions. The discovery of business workflows are proposed in [3] [8] [9], for purposes such as documentation, process optimization and conformance checking.

All these works explore usage data either to define similarity of clients in terms of the services they use, or relationships between services in terms of compositions and

processes they take part on. Our work differs from the above in terms of the granularity of patterns we seek (how service functionality is used), and on the ability to relate and describe similar clients in terms of quantified usage patterns.

VIII. Conclusions and Future Work

We presented a framework that supports the application of a KDD process over interaction logs to discover groups of clients with similar usage characteristics. By collecting and storing fine-grained usage data, applications can be clustered according to different criteria, without having to recollect and reprocess raw interaction data. To reduce the complexity inherent to any KDD process, the user is supported through predefined tasks that can be parameterized. Data selection and transformation tasks enable the generation of distinct types of profiles, according to the analysis goal. Whereas binary preparation is better to identify the applications impacted by changes, the weighted preparation is better to measure the change impact.

The framework integrates different algorithms that can be experimented and the results assessed using known external assessment measures. Experiments on synthetic data have displayed encouraging results even in the presence of significant amount of noise. Synthetic data was generated based on the workflows described in the documentation of a real, complex service, describing thus potential client applications that fit distinct profiles. Further experimentation needs to be developed with real data.

The knowledge extracted by the proposed process cannot be derivedby investigating the expected service workflows. Even when the provider expect interactions to follow a model, clients may not conform to it [9], or it may include several variability points that enable one to derive at most a worst case scenario, rather than the actual one [8].

Providers can leverage usage impact information to make decisions about the creation, maintenance and decommissioning of versions, but the segmentation of clients according to temporal usage activities or preferences suits other applications (e.g. service recommendation, optimization, load balance, redesign, etc).

Currently we are implementing unsupervised clustering assessment measures, and experimenting with more data to recommend clustering parameters in the future. We are also integrating it with the applications of the Usage Manager [10] [11]. Future work includes an evaluation of the costs involved in the collection of detailed data, mechanisms for exploring and interpreting the profiles, developing usage profiles for combined use of services in portfolios and service versions, as well as new applications, such as usage-oriented compatibility and service recommendation.

Acknowledgments

We would like to thank Lucas Alves who has helped with the implementation. This research is financially supported by FAPERGS, CNPq and CAPES - Brazil.

References

[1] M. P. Papazoglou, V. Andrikopoulos, and S. Benbernou, "Managing evolving services," *IEEE Software*, vol. 28, no. 3, pp. 49–55, 2011.

[2] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Addison Wesley, 2006.

[3] Q. A. Liang, J.-Y. Chung, S. Miller, and Y. OUYang, "Service pattern discovery of web service mining in web service registry-repository," in *e-Business Engineering, 2006. ICEBE '06. IEEE International Conference on*, 2006, pp. 286–293.

[4] Q. Yu, "Decision tree learning from incomplete qos to bootstrap service recommendation," in *ICWS, 2012*, june 2012, pp. 194 –201.

[5] Q. Zhang, C. Ding, and C. Chi, "Collaborative Filtering Based Service Ranking Using Invocation Histories," in *ICWS, 2011*. IEEE, 2011, pp. 195–202.

[6] W. Rong, K. Liu, and L. Liang, "Personalized Web Service Ranking via User Group Combining Association Rule," *2009 IEEE International Conference on Web Services*, pp. 445–452, Jul. 2009.

[7] X. Zhang, Y. Yin, M. Zhang, and B. Zhang, "Web service community discovery based on spectrum clustering," *2012 Eighth International Conference on Computational Intelligence and Security*, vol. 2, pp. 187–191, 2009.

[8] R. Tang and Y. Zou, "An approach for mining web service composition patterns from execution logs," in *Web Systems Evolution (WSE), 2010 12th IEEE International Symposium on*. IEEE, 2010, pp. 53–62.

[9] W. van der Aalst, "Service mining: Using process mining to discover, check, and improve service behavior," *Services Computing, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2012.

[10] M. Yamashita, B. Vollino, K. Becker, and R. Galante, "Measuring change impact based on usage profiles," in *Web Services (ICWS), 2012 IEEE 19th International Conference on*. IEEE, june 2012, pp. 226 –233.

[11] E. Silva, B. Vollino, K. Becker, and R. Galante, "A business intelligence approach to support decision making in service evolution management," in *Services Computing (SCC), 2012 IEEE Ninth International Conference on*. IEEE, june 2012, pp. 41 –48.

[12] D. Pfitzner, R. Leibbrandt, and D. Powers, "Characterization and evaluation of similarity measures for pairs of clusterings," *Knowledge and Information Systems*, vol. 19, no. 3, pp. 361–394, Jul. 2009.

[13] A. Chuvakin and G. Peterson, "Logging in the age of web services," *Security & Privacy, IEEE*, vol. 7, no. 3, pp. 82–85, 2009.

[14] M. Hall, E. Frank, and G. Holmes, "The WEKA data mining software: an update," *ACM SIGKDD*, vol. 11, no. 1, pp. 10–18, 2009.