# Usage Profiles:
## A Process for Discovering Usage Patterns over Web Services and its Application to Service Evolution

*Bruno Vollino, Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Rio Grande do Sul, Brazil*

*Karin Becker, Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Rio Grande do Sul, Brazil*

## 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 constitutes invaluable information to support more effective decisions. In this paper, the authors 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 a process to cluster client applications and derive usage profiles. The paper also discusses how usage profiles can help to access the real impact on clients of incompatible changes performed over service descriptions, and presents a usage-oriented compatibility assessment algorithm. Experimental results are presented for both the profile discovery process and profile-based compatibility analysis.*

*Keywords:* *Compatibility, Data Mining, Usage Patterns, Usage Profiles, Web Service*

## INTRODUCTION

Web services became vital for the business of many companies in the software industry, especially with the advent of the software on demand paradigm, such as SaaS (Software as a Service). As in any business, providers have interest in understanding 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 to avoid breaking clients, and service redesign evolution to keep up with clients expectations. Typically, these decisions are made without a clear understanding of the possible outcomes, frequently based on worst-case scenarios. Understanding the usage clients make of services is thus invaluable to support web service life-cycle (Papazoglou, Andrikopoulos et al., 2011).

DOI: 10.4018/jwsr.2013010101

Data mining techniques have been applied in many business segments to discover knowledge about clients, which is hidden in large volumes of data (Tan, Steinbach et al., 2006). Web service mining (Liang, Chung, et al., 2006) aims at discovering patterns of service usage, i.e. specific ways in which web services (or their operations) are used repeatedly by a group of users with similar properties, as well as are correlated to each other. Usage analysis have been used to support the recommendation of services (Yu, 2012; Zhang, Ding et al., 2011; Kang, Liu et al., 2012; Rong, Liu et al., 2009), the discovery of service composition communities (Zhang, Yin et al., 2009; Wang, Wang et al., 2012), or process discovery for applications such as process documentation, conformance checking or process optimization (Motahari-Nezhad, Saint-Paul et al., 2011; Musaraj, Yoshida et al., 2010; Tang & Zou 2010; van der Aalst, 2012). van der Aalst (2012) highlights that, 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 of services by their client applications.

Our work is focused on the usage analysis as a support for the service evolution lifecycle (Yamashita, Vollino et al., 2012; Silva, Vollino et al., 2012; Yamashita, Becker et al., 2012). 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 (Yamashita, Vollino et al., 2012) or financial metrics (Silva, Vollino et al., 2012). Another possible application is compatibility assessment. Compatibility has been traditionally addressed in terms of a worst-case scenario, i.e. based on the possibility of breaking existing clients (Andrikopoulos, Benbernou et al., 2012; Becker, Lopes et al., 2008; Fang, Lam et al., 2007). However, clients are bound to specific functionality, rather than the entire service interface, and therefore, incompatible changes may have different effects on clients (Yamashita, Becker et al., 2011; Zou, Fang et al., 2008; Ponnekanti and Fox, 2004). Usage-oriented compatibility assessment can support service evolution management by providing relevant information about the change impact on client applications. For instance, providers can evaluate the trade-offs between the costs of provisioning multiple versions of a service, and the benefits of not breaking clients. Service designers can also proceed with certain incompatible changes they would otherwise hesitate to perform due to the possibility of breaking clients, in case the impact is not considered significant to the business.

The contributions of this paper are twofold: (a) a framework that guides the discovery of usage profiles over monitored clients requests, through a knowledge discovery process (KDD) (Tan, Steinbach et al., 2006), and (b) a profile-based compatibility assessment algorithm, which identifies the changes that are incompatible with regard to the current usage of a specific group of clients at a fine-grain.

The usage discovery 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 require 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 extensively details the ideas

initially sketched in our previous work (Vollino et al., 2012; Silva, Vollino et al., 2012). It extends the work reported at (Vollino & Becker 2013), presenting additional experiments and an usage-oriented compatibility assessment algorithm. The framework contributes with techniques for identifying usage patterns that existing works on service mining (e.g. Liang, Chung, et al., 2006; Yu, 2012; Zhang, Ding et al., 2011; Kang, Liu et al., 2012; Rong, Liu et al., 2009; Zhang, Yin et al., 2009; Wang, Wang et al., 2012; Motahari-Nezhad, Saint-Paul et al., 2011), (Musaraj, Yoshida et al., 2010; Tang & Zou 2010; van der Aalst, 2012) have not addressed yet, namely groups of clients based on detailed service functional properties.

With regard to the profile-based compatibility assessment algorithm, the paper describes the algorithm, and illustrates the kind of result it delivers using a real service and the profiles identified in our experiments for that service. Related work (Ponnekanti & Fox, 2004; Zou, Fang et al., 2008) has proposed usage information in the context of adapting a client application to changes. Our point of view is the provider, who needs an understanding of the overall impact of changes in the whole set of clients applications. The algorithm presented in this paper develops an automated analysis at a fine-grained level (operations and data types), in which the compatibility assessment verdict is dependent on the usage. It complements our previous work (Yamashita, Vollino et al., 2012), in which we developed a method to quantify the impact of incompatible changes in terms of each profile. The case for usage-oriented compatibility assessment was made in (Yamashita, Becker et al., 2011).

The remaining of this paper is structured as follows. First, we present the fundamental concepts underlying KDD and clustering. Then, we provide an overview of the service evolution framework. Usage profiles and the Profile Manager, which is the module of the framework responsible to monitoring the requests and deriving the usage profiles, are then addressed. The KDD process proposed to generate usage profiles is detailed in the section that follows, and experimental results are discussed next. The usage-oriented compatibility assessment approach is then introduced, to illustrate how profiles can be explored for supporting service evolution. Related work is then described and compared to our work. Finally, we draw conclusions and discuss future work.

## KDD AND CLUSTERING

KDD is a process targeted at discovering new, valid and useful information from large datasets (Tan, Steinbach et al., 2006). This complex, 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) are used by Tan et al. (2006) to classify the techniques over orthogonal dimensions. 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. A tree generated by hierarchical clustering can be cut in any level to obtain a set of partitional clusters. The clustering is *exclusive* if each object belongs to a single cluster.

Clusters, on the other hand, 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' proto-

types; 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.

Once a set of clusters is found, it is necessary to assess that the clustering tendency is not a mere random structure, the number of yielded clusters, and how well data objects fit together. Assessment can be performed using supervised and unsupervised techniques. Supervised evaluation compares the discovered model to externally available information (e.g. a golden standard). Metrics such as the pair-counting F-Measure (Pfitzner, Leibbrandt et al., 2009) can be applied to support this comparison. However, in practice such a reference hardly exists, and the evaluation is made based on an expert's knowledge of which clusters are valid and useful with the help of unsupervised, internal indices. Usually, an internal index assumes a particular cluster definition, making this choice similar to the one of a clustering algorithm, and it enables the comparison of clusterings and algorithms of the same type (e.g. to find the best parameterization).

For instance, Silhouette (Rousseeuw, 1987) and SD (Halkidi, Vazirgiannis et al., 2000) are indexes that measure the cohesion and separation of partitional, well-separated clusters. Liu, Li et al. (2010) present a comparative analysis, including several other internal indexes.

## SERVICE EVOLUTION FRAMEWORK OVERVIEW

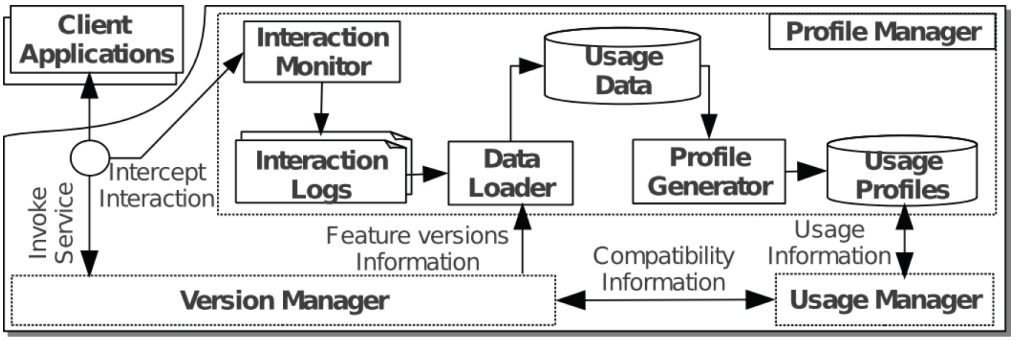
We proposed in (Yamashita, Vollino et al., 2012) a service evolution framework to support actions and decisions underlying service evolution, by considering the actual use clients make of services. As depicted in Figure 1, the framework is composed of the following modules: Version Manager, Profile Manager and Usage Manager.

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 (Yamashita, Becker et al., 2012), 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. This module is detailed in the next two sections.

Finally, the *Usage Manager* encompasses components that explore the profiles to assess

*Figure 1. Service evolution framework*



change impact based on the actual use clients make of a service. In Yamashita, Vollino et al. (2012), we proposed profile-based metrics to quantify the impact of incompatible changes, and in Silva, Vollino et al. (2012), we explored usage profiles to measure the financial impact of changes. In this paper, we present a novel application for this module, namely usage-oriented compatibility assessment.

## 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.

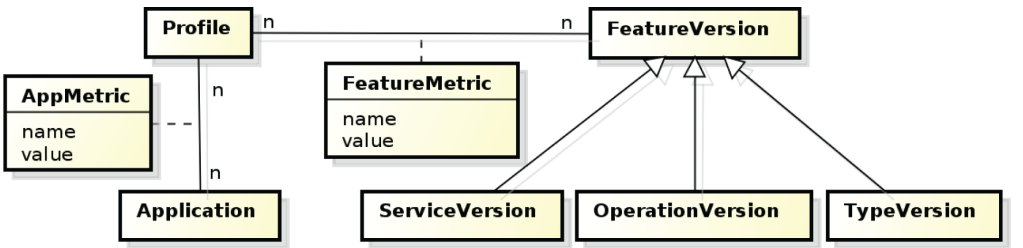
Usage profiles are representations of groups of client applications with similar usage patterns with regard to functionality described in the service interface. Such patterns describe the operations clients make use of, as well as the types of data they exchange. For example, some providers may be interested in understanding whether optional parameters are indeed used within certain groups of applications. The analysis of profiles in such a detailed level can reveal interesting knowledge that suits many applications. For instance, awareness of which operations and types are actually in use may motivate providers/designers 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.

Each profile (Figure 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 or involving a data type). 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.

### 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

*Figure 2. Usage profile structure*



the logging infrastructure will reside in the web service architecture.

The service interactions may be intercepted (Chuvakin & Peterson 2009): 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 (Zhang, Ding et al., 2011) or provider side (Tang & Zou, 2010); or hard coded in the web service itself. Given our purpose of detecting patterns in service usage, 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. Another advantage is that message handlers depend 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. Because we need to identify which application issued each request, 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.

## Data Loader and Usage Database

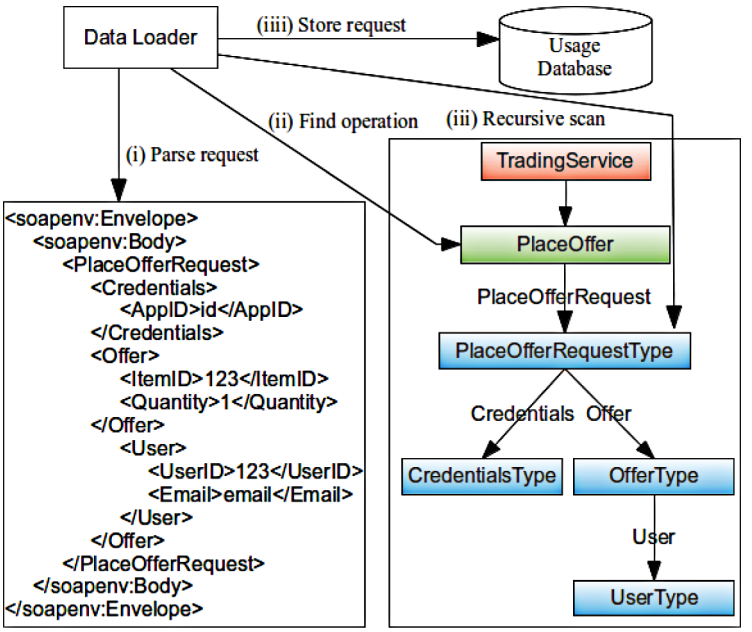
The Usage Database is a general-purpose, centralized repository that contains detailed data about service usage, and which suits different types of analysis. In this way, different criteria for defining the profiles can be experimented, as discussed in the next sections. The Data Loader is responsible 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. Figure 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 Figure 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 enable 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 (e.g. non-conformant to the service description).

The Usage Database schema is depicted in Figure 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

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




to associate each feature/parameter with its respective version in the Version Repository. Information about optionality of a parameter is also retrieved from the service interface and recorded.

## 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 Figure 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.

*Figure 4. Usage database schema*

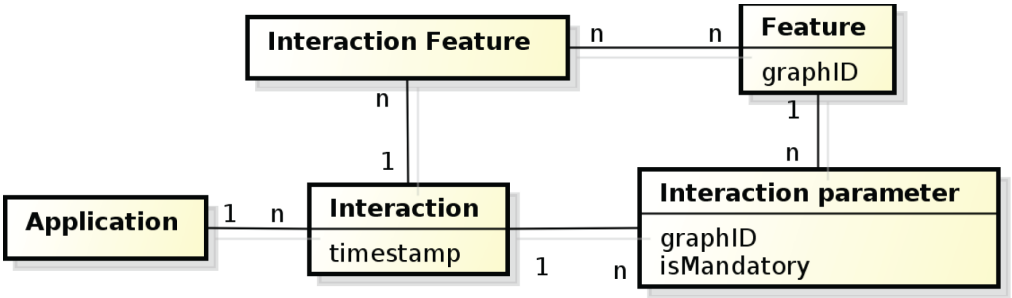
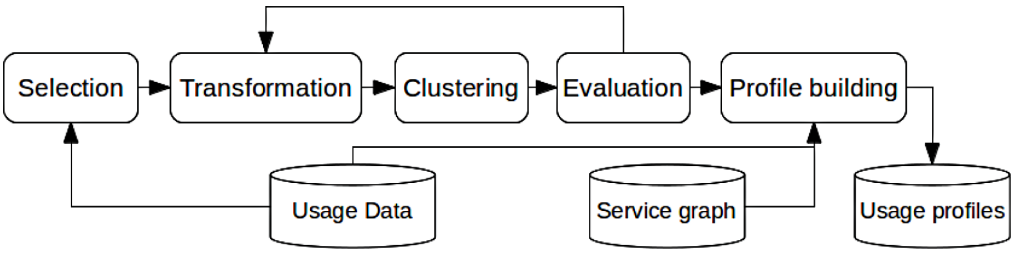


*Figure 5. The tasks of the profile discovery workflow*




## PROFILE GENERATION WORKFLOW

### Web Service Monitoring

Data preparation is crucial in profile discovery, because it influences how mining algorithms will cluster service clients. Results can be affected by filters and transformations that are applied over the data, because similarity functions and clustering algorithms are very sensitive to the characteristics of input data. 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.

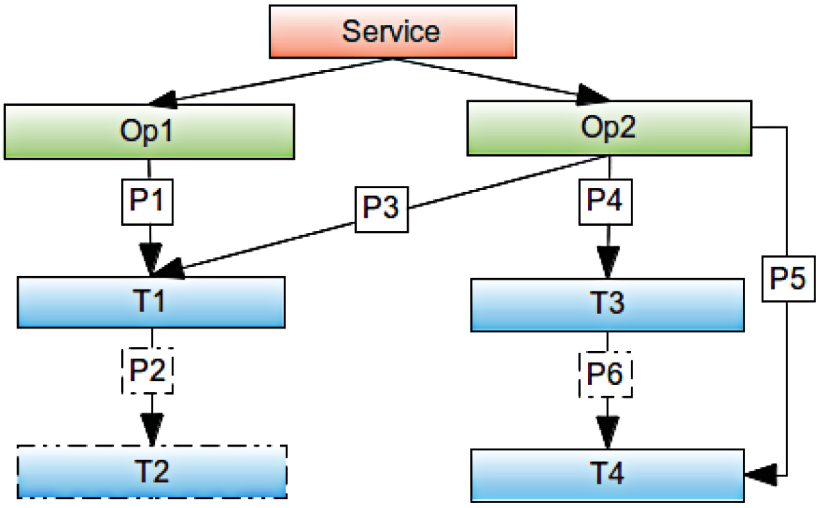
### Data Selection

Data selection is driven by two parameters: *time interval* and *data granularity*. Usage is temporal, which means that clients may change overtime the way they use services (e.g. in terms of operations requested), as expected in the decoupled life-cycles of services and client applications. 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. This choice determines the data to be extracted from the Usage Database. If operation level is chosen, the query to the Usage Database returns all the features in the relationship 'Interaction Feature' (Figure 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' (Figure 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, and the parameters they depend on. If parameters are mandatory, at any level of recursion, their presence is implied by the mere usage of the operation or parameter that depend on them, and therefore they can be disregarded.

To illustrate how relevant data types are retrieved, Figure 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

*Figure 6. Mandatory and optional parameters*




ones that do not. Note that T4 is not selected, because it is also referenced by the mandatory parameter P5.

## Data 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 (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 (Tan, Steinbach et al., 2006), such as normalization or dimension reduction (e.g. eliminate features never used).

The profiles generated by each type of summarization answer 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.

## Clustering

As already mentioned, 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 cluster definition and corresponding clustering technique. Our approach to this problem 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 for these algorithms.

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, 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 have already implemented two internal indexes in the framework, namely Silhouette and SD, to develop experiments and compare their contribution to clustering assessment. We are currently working on the integration of additional ones, particularly S_dbw (Halkidi & Vazirgiannis, 2001), which aims at assessing the quality of clusters of distinct types, such as centroid or density-based, by incorporating parameters that measure the separation and compactness of the clustering.

### 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. Profiles, on the other hand, are an enriched representation of these groups of applications (Figure 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 Figure 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.

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, Figure 1).

The pseudo-algorithm of Figure 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 9-12). Then, it recurses over the operation parameters trees and types subtrees, computing the number of interactions in which each type appears (line 11). 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 (lines 13,14).

## 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

*Figure 7. The algorithm for building profiles*

```
function buildProfile(Cluster c, Timestamp initDate, Timestamp finalDate): boolean
1   prof ← Profile();
2   instances ← cluster.instances()
3   prof.apps ← processAppsAndMetrics(initDate, finalDate, instances)
4   for all app in prof.appplications() do
5       for all attr in cluster.attributes() do
6           feat ← versionRepository.feature(attr.featureId())
7           if feat.isOperation() and instances(app).value(attr) > 0
8           then
9               opUsage ← usageDB.numInteractions(initDate, finalDate, app, feat)
10              app.usageMap(ft) ← opUsage
11              app.usageMap ← recurseParameters(initDate, finalDate, app, feat,
12                                                  app.usageMap, opUsage)
13      for all (f, val) in app.usageMap do
14          prof.usageMap(f) ← prof.usageMap(f) + val
15  return prof;
```

workflows could be combined differently to characterize sets of applications with similar behavior. This is a common approach for validating service mining works, due to the challenges of obtaining real data given to its proprietary nature (Nayak, 2008; Motahari-Nezhad, Saint-Paul et al., 2011).

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 (Hall, Frank et al., 2009): K-Means, 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 the 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 (Pfitzner, Leibbrandt et al., 2009). This supervised assessment metric reflects the homogeneity of applications inside clusters and the heterogeneity of distinct clusters, regardless the number of clusters.

Finally, we also experimented with internal validity indexes, in order to compare the results of the supervised evaluation with the ones using unsupervised clustering validation techniques. The Silhouette Coefficient (Rosseuw 1987) and the SD index (Halkidi, Vazirgiannis et al., 2000) were calculated from the resulting clusterings, and compared with the corresponding F-Measure values. The rationale is that if we want to be capable of providing insights to support the choice of the better algorithms and parameterizations, unsupervised indexes must have a strong correlation with the corresponding F-Measure value.

## Dataset

We adopted version 753 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. Figure 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 1, we have simulated 525 applications distributed in 6 profiles, which performed 448,703 requests for 42 distinct operations. The Venn diagram in Figure 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. 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.

## 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 2(A),

*Figure 8. Example of a simulated application profile*

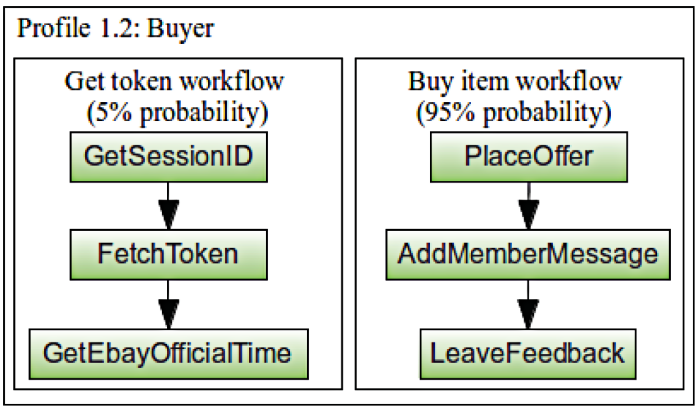
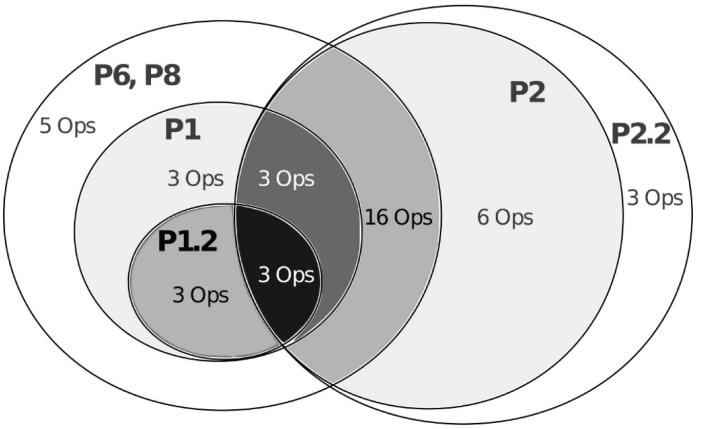



*Table 1. Simulated data profiling*

| 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 |

*Figure 9. The simulated profiles and their intersections*




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 represents 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 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 2(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 (P2, P2T, P2.2, P6 and P6T).

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 requires extensive trial-error to find a good parameterization (minimum points = 6, epsilon = 0.5). These results may be influenced by the number of common operations that are in the intersection among profiles. Nevertheless, in real situations, we hypothesize that some services tend to have a set of core operations that are shared among many profiles (if not most of them), and that clustering techniques must be

*Table 2. Clustering results*

| A. Binary Data in Granularity of Operations | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Binary | | | 10% Noise | | | 30% Noise | | |
| Algorithm | 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 |
| Hierarch. | 5 | 5 | 0.99 | 5 | 5 | 1.00 | 5 | 5 | 1.00 |
| **B. Binary Data in Granularity of Types** | | | | | | | | | |
| | Binary | | | 10% Noise | | | 30% Noise | | |
| Algorithm | 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 |
| Hierarch. | 7 | 7 | 0.99 | 7 | 7 | 1.00 | 7 | 7 | 1.00 |
| **C. Weighted Data in Granularity of Operations** | | | | | | | | | |
| | Weighted | | | 10% Noise | | | 30% Noise | | |
| Algorithm | 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 |
| Hierarch. | 6 | 6 | 0.97 | 6 | 6 | 0.98 | 6 | 6 | 0.98 |

able to handle this degree of similarity among profiles. This hypothesis needs to be confirmed by extensive experimentation, particularly with real services.

## 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 2(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 parameterizations we have found, failed to create one cluster per profile. 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.

As mentioned, our simulated data, based on workflows of operations and compositions of workflows, generate profiles that are close to each other (i.e. not very well-separated), and this characteristic is stressed with the injection of noise. The observed consequences are the reduced number of identified profiles (4 out of 6 expected ones) and low F-measure for the K-means algorithm (0.52 in the worst case), which is particularly susceptible to noisy data. Based on our assumptions about the characteristics of service usage patterns, we can state that the K-means is not a suitable technique to cluster weighted data.

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.

## Evaluation of Internal Validity Indexes

In the two previous subsections, we evaluated the resulting clusterings based on expectations against a golden standard, i.e. the injected profiles. However, in the real world, *a priori* expectations about the profiles are unlikely to exist. Therefore, the experiments are completed by a comparison between the F-measure results presented previously (Table 2), and the results calculated by two internal indexes, namely Silhouette and SD. Our premise is that, to be capable of providing insights about the choice of suitable algorithms and parameterization, the internal indexes used to evaluate the clustering must have a strong correlation with the F-Measure value. In other words, an appropriate internal index should be able to identify if the clustering captures the inherent patterns of the dataset, despite the presence of noise, or, whether at least it places the noisy instances in the clusters to which they resemble the most.

Recall these indexes are targeted at evaluating well-separated, partitional clusterings. According to Liu et al. (2010), Silhouette and SD indexes handle well: (a) noise, which in our dataset correspond to applications with random behavior; b) clusters of different densities, represented by profiles with very distinct numbers of applications; and (c) clusters of different sizes, which occur when applications of a same profile significantly varies either on the features used (binary data) or the number of requests (weighted data).

The results are shown in Table 3, considering the three previous datasets, and the respective noise levels added. The optimal value for the Silhouette coefficient is 1, where 0 means that no clustering tendency was detected. On the other hand, the smallest the value for the SD index, the better is the clustering. It can be observed that, in the absence of noise, both SD and Silhouette (Silh.) indexes match the F-Measure (F-M) results previously discussed, regardless the data preparation (binary and weighted) and data granularity. This means that these three measures agree on the best quality clusterings for noiseless datasets. It also reveals that the resulting clusters are fairly well-separated from each other, despite the similarity among them. However, these results are not observed for noisy data. For instance, in Table 3 B, the best Silhouette value for the dataset with 30% of noise (0.6622) corresponds to the worst clustering according to F-Measure (0.86). Conversely, the best clustering according to F-Measure (1) present the worst value for Silhouette (0.4342), and second worst value for SD index (1.4519). This means that good clusters are evaluated as low quality ones by unsupervised metrics, and vice-versa. However, we acknowledge that, in many cases, the difference between the absolute values are not very relevant for the yielded profiles, in the sense that they could be assessed as good enough for an analyst, independently of the algorithm chosen. Further experiments are necessary for reaching more sound conclusions about the appropriateness of these indexes.

These results can partially be explained by the incapacity of these indexes to deal with clusters that are very close to each other, i.e. which are not well-separated (Liu et al., 2010). In the binary datasets, the clusters are naturally

*Table 3. Internal validity assessment*

| A. Binary Data in Granularity of Operations | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Binary | | | 10% Noise | | | 30% Noise | | |
| Algor. | Sillh. | SD | F-M | Sillh. | SD | F-M | Sillh. | SD | F-M |
| K-Means | 0.9667 | 0.9450 | 1.00 | 0.7366 | 1.1552 | 1.00 | 0.5769 | 1.0529 | 0.94 |
| EM | 0.9667 | 0.9450 | 1.00 | 0.8541 | 0.8402 | 0.94 | 0.7142 | 0.8281 | 0.94 |
| DBSCAN | 0.9340 | 0.9618 | 0.97 | 0.6525 | 1.0059 | 0.97 | 0.4589 | 0.8226 | 0.97 |
| Hierarch. | 0.9576 | 0.9583 | 0.99 | 0.7214 | 1.1654 | 1.00 | 0.4588 | 1.5094 | 1.00 |
| **B. Binary Data in Granularity of Types** | | | | | | | | | |
| | Binary | | | 10% Noise | | | 30% Noise | | |
| Algor. | Sillh. | SD | F-M | Sillh. | SD | F-M | Sillh. | SD | F-M |
| K-Means | 0.9456 | 0.9326 | 0.94 | 0.7473 | 1.0015 | 0.86 | 0.5283 | 0.9352 | 0.95 |
| EM | 0.9676 | 0.9080 | 1.00 | 0.8504 | 0.7678 | 0.95 | 0.6622 | 1.5998 | 0.86 |
| DBSCAN | 0.9004 | 0.9495 | 0.94 | 0.6811 | 1.0512 | 0.94 | 0.5115 | 1.0814 | 0.94 |
| Hierarch. | 0.9605 | 0.9252 | 0.99 | 0.7000 | 1.1465 | 1.00 | 0.4342 | 1.4519 | 1.00 |
| **C. Weighted Data in Granularity of Operations** | | | | | | | | | |
| | Weighted | | | 10% Noise | | | 30% Noise | | |
| Algor. | Sillh. | SD | F-M | Sillh. | SD | F-M | Sillh. | SD | F-M |
| K-Means | 0.3141 | 1.6136 | 0.78 | 0.4787 | 0.6755 | 0.72 | 0.3002 | 1.4660 | 0.52 |
| EM | 0.4035 | 1.6986 | 0.88 | 0.4276 | 0.7579 | 0.95 | 0.3771 | 0.7158 | 0.95 |
| DBSCAN (0.5, 3) | 0.3952 | 0.6576 | 0.93 | 0.3037 | 0.7122 | 0.93 | 0.2312 | 0.7144 | 0.93 |
| DBSCAN (0.5, 7) | 0.3976 | 0.7030 | 0.89 | 0.3287 | 0.7079 | 0.89 | 0.2377 | 0.6906 | 0.89 |
| Hierarch. | 0.4662 | 0.6459 | 0.97 | 0.3731 | 0.7349 | 0.98 | 0.2553 | 0.9174 | 0.98 |

similar to each other due to the high number of common operations, and the injection of noise makes it even harder to separate them. With regard to the weighted preparation, the normalization applied to the dataset also reduces the distance between the instances, resulting in clusters that display high inter-cluster similarity, and therefore, are difficult to separate. The degradation of results when noise is inserted is an evidence of this problem.

These experiments provide evidences that these measures cannot handle noisy data at the appropriate level for this domain, which can hinder their use as quality assessment metrics in usage profiles. However, further experimentations are necessary. We are also implementing the S_dbw index (Halkidi & Vazirgiannis 2001), which, according to Liu et al. (2010), can handle both close clusters and noise.

## USAGE-ORIENTED COMPATIBILITY ASSESSMENT

In this section, we propose a profile-based compatibility assessment algorithm. It is at the core of the Profile-based Compatibility analyzer, one of the applications encompassed in the Usage Manager module of the service evolution framework (Figure 1). Other applications that explore usage profiles are the Usage Analyzer (Silva, Vollino et al., 2012) and the Business

Intelligence environment for supporting decisions about service evolution (Silva, Vollino et al., 2012). The former enables to quantify the impact of (incompatible) changes. The latter provides a central repository that integrates usage and financial metrics, together with analytical resources to gain insight about the impact of changes.

Compatibility, particularly backward compatibility (Andrikopoulos, Benbernou et al., 2012) (Fang, Lam et al., 2007) (Becker, Lopes et al., 2008), is crucial in service change management, because it defines whether existing clients will be affected by changes introduced into newer versions of a service. The assessment of compatibility is traditionally focused on the worst case of total compatibility, which means that if a single element of service version S is incompatible with the same element in version S', then S and S' are totally incompatible (Andrikopoulos, Benbernou et al., 2012; Becker, Lopes et al., 2008; Fang, Lam et al., 2007). However, assessing the compatibility of service versions does not necessarily capture the impact of the incompatible changes, because client applications are not bound to the whole service (as described by the service interface), but rather to specific features within the offered functionality (Yamashita, Vollino et al., 2012; Yamashita, Becker et al., 2012; Zou, Fang et al., 2008; Ponnekanti and Fox, 2004). Therefore, client applications can be impacted in different ways, or even not be impacted at all.

Usage-oriented compatibility assessment focus on identifying the changes that are incompatible with the current usage of a specific group of clients, as represented by a usage profile. It can support service evolution management by providing relevant information regarding the effects of changes on client applications. For instance, providers can evaluate the trade-offs between the costs of provisioning multiple versions of a service, and the benefits of not breaking their clients (e.g. Silva, Vollino et al., 2012). Service designers can also proceed with certain incompatible changes they would otherwise hesitate to perform due to the possibility of breaking clients, in case the impact is not considered significant to the business.

In the remaining of this section, we provide some background on the versioning model and Version Manager, which is necessary for explaining the algorithm for usage-oriented compatibility assessment. We also illustrate how the algorithm could be applied to the same case study developed in the previous section, and discuss the type of insights it yields.

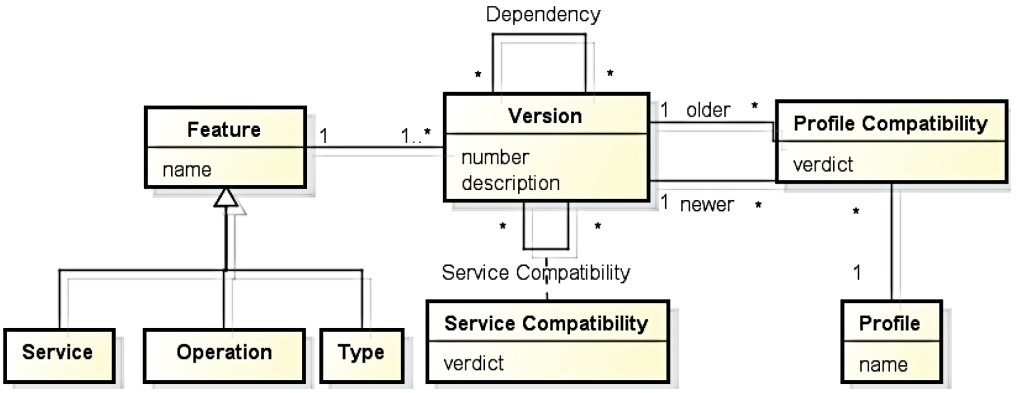
## The Version Manager

The Version Manager is the component of the framework responsible for detecting changes, versioning service descriptions, and assessing version compatibility. It relies on a finer-grained version unit referred to as feature, which corresponds to a portion of an interface description, such as an operation, data type, or information related to the overall service.

The feature-based versioning model is depicted in Figure 10. Each Feature has a unique name, and it is related to at least one Version. A Version is thus a generalization of Service, Operation and Type. In turn, each Version is associated to a number, description (a textual description of the WSDL document), and possibly a set of dependent versions that are used to describe it (relationship 'Dependency'). For instance, a service depends on its operations, an operation depends on the types used to describe its messages, and so forth. Hence, a service version is represented as a graph of interrelated feature versions. Versions are uniquely identified by the pair *Feature.name,Version.number*.

When a new service interface document is exposed, the Version Manager converts it into this abstract internal representation. The features are extracted from the document, their respective descriptions and relationships are compared to the corresponding existing versions, and new versions are created only for detected changes. A new service is represented by a rooted graph that encompasses existing or new versions of the features that compose the service.

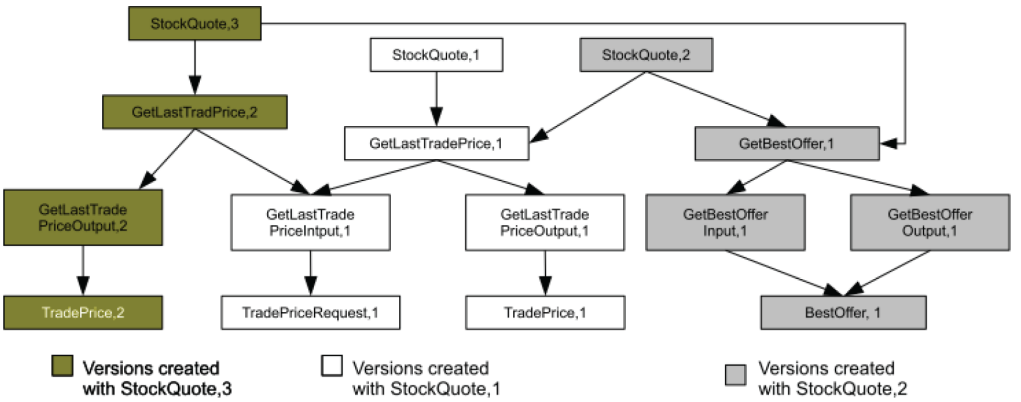
*Figure 10. Feature-based versioning model*




In Figure 11 we illustrate the versioning results after submitting three interface descriptions of the classical W3C *StockQuote* service[3]. The different colors represent the new versions created for each description. When the first service description is submitted, features for the service, operation and data types are created with the corresponding versions (graph rooted at *StockQuote,1)*. Suppose the second description introduces a new operation *GetBestOffer*, with related to new data types. New features and respective versions are created for these additions. However, considering all previously existing features, only *StockQuote* feature is versioned (*StockQuote,2*) due the inclusion of the new operations. Notice *StockQuote,2* is related to existing versions (e.g. *GetLastTradePrice,1*), due to the unchanged parts of the service description. Finally, assume the third description changes the primitive type associated with *TradePrice* to double. A new version is created (*TradePrice,2*), and due to the ripple effect, feature versions that depends on it directly (*GetLastTradePriceOutput*,1), or indirectly (e.g. *StockQuote,3*) are equally versioned. Further details can be obtained in (Yamashita, Becker et al., 2012).

Another component of the Version Manager is the Compatibility Analyzer, It aims at assessing automatically the compatibility of two service descriptions (i.e. total service compatibility), using the algorithm described

*Figure 11. Version graphs of 3 interface descriptions of StockQuote service*



in (Yamashita, Becker et al., 2012). Given the set of compatibility rules displayed in Table 4, the algorithm analyzes and records the compatibility of any two versions of a same feature (relationship 'Service Compatibility' in Figure 10). Any other change is considered incompatible. These compatibility rules represent the consensus on the literature about compatibility (Fang, Lam et al., 2007; Andrikopoulos, Benbernou et al., 2012).

When two service versions are compared, the algorithm assesses recursively all dependent features. For instance, when the graph rooted at *StockQuote,3* is recursively compared to the one rooted at *StockQuote,2*, the services are assessed as incompatible. In this comparison, *TradePrice,2* is compared to *TradePrice,1*; *GetLastTradePriceOutput,2* is compared to *GetLastTradePriceOutput,1*; *GetLastTradePrice,2* is compared to *GetLastTradePrice,*1; and finally, *StockQuote,3* is compared to *StockQuote,2*. Despite the incompatible change actually was applied to *TradePrice,2*, all feature versions that depend on it directly or indirectly are also assessed as incompatible due to the ripple effect. Table 5 summarizes the compatibility assessments for this example: the feature, the versions compared, the compatibility verdict, and whether it corresponds to a direct change or was affected by one. Notice that if the version graphs rooted at *StockQuote,2* and *StockQuote,1* were analyzed, these would be the only two versions compared, as all the remaining changes correspond to new features. According to the rules, they would be assessed as compatible.

## Profile-Oriented Compatibility Assessment

The goal of the usage-oriented compatibility assessment is to detect the incompatible changes with regard to identified usage patterns, such that the impact can be quantified and its relevance analyzed with regard to business objectives. Suppose that two profiles are detected over *StockQuote,2*: clients that invoke both operations (P1), and clients that invoke only *GetBestOffer* (P2). Thus, the following profiles would be created:

*Table 4. Compatibility cases*

| Cases | Change | Feature Type | Description | Verdict |
|---|---|---|---|---|
| 1 | Add | Operation | Add new operation as dependent of a service | Compatible |
| 2 | Add | Type | Add new type as dependent of a new operation/type | Compatible |
| 3 | Add | Type | Add new type as dependent of an existent operation/type | Incompatible |
| 4 | Update | Type | Change in description due to order, cardinality or type update | Incompatible |
| 5 | Remove | Operation | Remove operation as dependent of a service | Incompatible |
| 6 | Remove | Type | Remove type as dependent of a service/type | Incompatible |

*Table 5. Compatibility assessment for StockQuote service versions*

| Feature | Older, Newer | Compatibility Verdict | Reason |
|---|---|---|---|
| TradePrice | v1, v2 | incompatible | changed |
| GetLastTradePriceOutput | v1, v2 | incompatible | affected |
| GetLastTradePrice | v1, v2 | incompatible | affected |
| StockQuote | v2, v3 | incompatible | affected |

- P1 = {*StockQuote,2; GetBestOffer,1; GetBestOfferInput,1; GetBestOfferOutput,1; BestOffer,1; GetLastTradePrice,1; GetLastTradePriceInput,1; GetLastTradePriceOutput,1; TradePrice,1; TradePriceRequest,1*};
- P2 = {*StockQuote,2; GetBestOffer,1; GetBestOfferInput,1; GetBestOfferOutput,1; BestOffer,1*}.

The result of the profile-oriented compatibility assessment of *StockQuote,2* and *StockQuote,3* with regard to each profile is displayed in Table 6. Considering only profile P2, the incompatible changes that were introduced in the graph rooted at *StockQuote,3* do not affect clients because they do not use *GetLastTradePrice* operation. Recall that *StockQuote,3* and *StockQuote,2* were only assessed as incompatible in Table 5 because the former was affected by a change performed on *TradePrice,2,* which was cascaded upwards. However, this version is not in profile P2, nor the ones that depend on it (e.g. *GetLastTradePrice,2*). Thus, *StockQuote,2* and *StockQuote,3* are assessed as compatible, with regard do P2. On the other hand, P1 clients are impacted, and therefore *StockQuote,2* and *StockQuote,3* are assessed as incompatible with regard to P1.

Suppose that, using the quantification and analysis mechanisms proposed in (Silva, Vollino et al., 2012), the provider realizes that 90% of the clients are related to P2, and that they represent the paying clients. P1, on the other hand, encompasses clients which use the service for free. By knowing the impact, a service designer can decide whether he/she should publish the new description, or consider a design alternative that would not break any client. The provider, on the other hand, can weigh the trade-offs between the cost of provisioning two versions against the benefits for the business of not breaking P1 clients.

The algorithm proposed in this paper assesses the compatibility between two feature versions (older and newer) with regard to a profile. It extends the one presented in (Yamashita, Becker et al., 2012) in order to relax some incompatible cases of Table 4, in case the changed/affected incompatible versions are not included in the profile. Once the compatibility is assessed, it is recorded in the Version Repository ('Profile Compatibility' in Figure 10). The pseudo-algorithm is presented in Figure 12. Recall that a Profile contains a set of Feature Versions (Figure 2), which correspond to the concept of Version in the versioning model (Figure 10). These versions can be related to a Service, an Operation, or a Type feature.

The algorithm aims to recursively evaluate the compatibility relationship between two feature versions (*old* and *new*), within the context provided by the usage profile (*prof*) according to the rules summarized in Table 4. We assume that both versions relate to the same feature (i.e. have the same name). The version graph rooted at *old* is traversed and compared to the one rooted at *new* in a depth-first manner, which enables the propagation of detected incompatibilities to the versions that depend directly or indirectly on a version assessed as incompatible.

*Table 6. Profile-based assessment for StockQuote service versions*

| Profile | Feature | Older, Newer | Usage-Oriented Compatibility Verdict |
|---|---|---|---|
| P2 | StockQuote | v2, v3 | compatible |
| P1 | StockQuote | v2, v3 | incompatible |
| P1 | TradePrice | v1, v2 | incompatible |
| P1 | GetLastTradePriceOutput | v1, v2 | incompatible |
| P1 | GetLastTradePrice | v1, v2 | incompatible |

*Figure 12. Usage-oriented compatibility assessment algorithm*

```
function assessCompatibility(Version old, Version new, Profile prof): boolean
1   boolean compat ← true;
2   if prof.contains(old)
3   then begin
4         for all d in old.dependents()
5         do if prof.contains(d) and not new.contains(d)
6            then compat ← false;
7         for all d in new.dependents()
8         do begin
9            v ← old.findCorrespondingVersion(d) ;
10           if v <> null and v.number <> d.number
11           then compat ← compat and assessCompatibility(v, d, prof);
12           if v = null and d.isType()
13           then compat ← false;
14           end;
15        if not new.compatibleDescription(old)
16        then compat ← false;
17        old.setCompatibilityVerdict(new, prof, compat);
18        end;
19  return compat;
```

The algorithm assesses only versions that are present in the profile *prof* (line 2), otherwise it simply disregards the change, because it will not affect the clients represented by the profile. Then, it verifies (lines 4-6) if no dependents were removed (cases 5 and 6), except if the removal concerns a dependent version that is not in the profile. In this case, the removal is also disregarded as an incompatible change. Recall that this situation corresponds to operations that are not invoked by the clients of the profile, or optional parameters that they never include in exchanged messages. Then, it recursively evaluates the compatibility of all corresponding dependent versions of *new* with regard to *old* (lines 7-14). If a corresponding feature version is found (same feature name with different version numbers), the algorithm recursively assesses their compatibility (lines 10-11). If it corresponds to a new feature (lines 12-13), it considers as compatible only the case of operation addition (case 1) and type addition unrelated to existing types (case 2), otherwise it assesses the change as incompatible due to case 3. Finally, the description fragment associated with the compared versions is compared (lines 15-16), and the compatibility relationship, with respective verdict, is recorded (line 17). Function *compatibleDescription* currently evaluates true if: a) in the case of Type versions, it interprets the XML for the rules of case 4, and b) for Operation and Service versions, if the descriptions are exactly the same. Notice that the algorithm does not stop when an incompatibility is detected because we need to assess all feature versions that compose the service description. In the future we plan to adopt less restrictive compatibility rules (e.g. input compatibility (Becker, Lopes et al., 2008), T-shape changes (Andrikopoulos, Benbernou et al., 2012)).

## Illustration

We illustrate the application of the usage-oriented compatibility algorithm using the clusters discovered in the previous section. We adopted again the *eBay Trading* service, for which a new version is released every two weeks. For each version, there is a release notes entry on the eBay website[4] that reports

the explicit changes with regard to the previous version. For this experiment, we considered two successive versions, identified as 753 and 757. For version 757, the release notes reports 14 incompatibly changed operations with regard to the previous one. The service WSDL document is very long (approximately 130.000 lines), and therefore it is very difficult to locate the exact changes, and how they affect applications. With regard to compatibility, it should be noticed that eBay establishes an evolution policy that requires that applications handle unrecognized data. Comparing to the compatibility rules of Table 4, the major difference is that case 3 is relaxed. So we changed our algorithm to apply our understanding of eBay compatibility rules.

For this experiment, we considered the five (5) clusters yielded using the hierarchical clustering technique, applied over the first dataset (i.e. only operations, prepared using binary representation), and with no noise. Three important remarks need to be made over this set of clusters: (a) the dataset did not include the simulated profile P8; (b) we did not consider noisy data because there were no corresponding interactions in the Usage Database to quantify the profile metrics; and (c) recall there were three misplaced applications (F-Measure 0.99), which were clustered together with the applications included in P2, instead of P2.2. So we shall refer to these clusters as P2' and P2.2'. Clusters P1, P1.2 and P6 correspond exactly to the expected simulated profiles described in Table 1.

Table 7 reports the results of our experiments. It compares the results as reported on eBay release notes (columns labeled as *eBay release notes)*, and the ones found by our algorithm (columns labeled as *eBay compatibility rules*). For each case we report the number of incompatibly changed operations with regard to a profile (the number of operations used in the profile is indicated in parenthesis), the number of impacted applications (the number in parenthesis correspond to the total of applications in the profile) and impacted requests. It is possible to see that only 10 out of these 14 operations documented by eBay as incompatible affect the applications of the profiles. However, the rules applied to by our algorithm assessed 34 operations as incompatible, of which 13 affect the applications in the profile. This difference highlights the importance of having very explicit compatibility rules, as discussed in more details in (Yamashita, Becker et al., 2012). Regardless the criteria used assess compatibility, in both cases, none of the applications in profiles P1 and P1.2 were affected (150 applications), whereas 100% of the applications on the other profiles were (250 applications). Thus, usage-oriented compatibility analysis enables to understand that 63% of the applications will be impacted by the changes. However, the number of impacted requests is much smaller (17% and 22%, respectively).

Table 8 displays the impact on the profiles per incompatible operation, which is another result that is output by our algorithm. For each

*Table 7. Usage-oriented compatibility assessment of eBay trading 757 and 753*

| | **eBay Release Notes** | | | **eBay Compatibility Rules** | | |
|---|---|---|---|---|---|---|
| **Profile** | **Incomp. Ops** | **Impacted Apps** | **Impacted Requests** | **Incomp. Ops.** | **Impacted Apps** | **Impacted Requests** |
| P2' | 6 (28) | 103 (103) | 25,898 | 9 (28) | 103 (103) | 32,750 |
| P2.2' | 6 (31) | 22 (22) | 4,651 | 9 (31) | 22 (22) | 5847 |
| P6 | 5 (33) | 125 (103) | 25,830 | 9 (33) | 125 (103) | 34,981 |
| P1.2 | 0 (6) | 0 (100) | 0 | 0 (6) | 0 (100) | 0 |
| P1 | 0 (12) | 0 (50) | 0 | 0 (12) | 0 (50) | 0 |
| **Total** | **10 (48)** | **250 (63%)** | **56,379 (17%)** | **13 (48)** | **250 (63%)** | **73,578 (22%)** |

profile, it shows the impacted requests for an operation, the number of profiles affected by the operation, and the total affected requests. For the sake of illustration, we considered only the operations reported as incompatible by eBay documentation. It is possible to see that, despite these profiles share a number of common operations (Figure 9), only 3 out of the 16 common operations were changed. It is also possible to see that changes that affected P2.2' and P2' refer to 3 out of the 6 operations in the intersection of these two profiles.

In conclusion, profiles and usage-oriented compatibility assessment enable a more fine-grained analysis that could be considered by the designer before proceeding with the changes. It can lead to the insight of the critical operations, in terms of current usage, and possible design alternatives to be considered in order not to break clients.

## RELATED WORK

Liang et al. (2006) define three types of service usage patterns: users access, service composition, and business process. According to them, patterns are organized into three levels: user request, template and instance. User request level is mainly focused on clients and how they submit requests, such that users concerns and interests can be connected with related Web services. At template level, the concern is to explore the abstract structure of services (service interfaces, operations, messages) to understand how the components of services correlate, particularly in terms of compositions and processes. At instance level, service usage concerns the constraints over specific service providers.

Works such as (Yu, 2012; Zhang, Ding et al., 2011; Kang, Liu et al., 2012, Rong, Liu et al., 2009) address user access patterns at user request level, mainly for the purpose of service recommendation. In this type of application, a similarity model is built either over clients (collaborative filtering), items to be recommended (content-based filtering), or both. Clients are clustered in (Yu, 2012) according to historical QoS and similarity over service invocation, in order to build a predictive model for future users. It proposes an approach for the so-called cold start problem, when there is not enough data to characterize users' interests. Known and inferred QoS values for service invocations are represented in a sparse matrix, which is used to cluster users according to QoS similarity. A

*Table 8. Impact of incompatible operations per profile*

| Operations | P2' | P2.2' | P6 | Affected Profiles | Affected Requests |
|---|---|---|---|---|---|
| AddFixedPriceItem | | | 6,160 | **1** | **6,160** |
| AddItem | 6,089 | 1,089 | | **2** | **7,178** |
| GetItem | 6,599 | 1,187 | 6,747 | **3** | **14,533** |
| RelistFixedPriceItem | | | 595 | **1** | **595** |
| RelistItem | 518 | 100 | | **2** | **618** |
| ReviseFixedPriceItem | | | 6,150 | **1** | **6,150** |
| ReviseItem | 6,078 | 1,085 | | **2** | **7,163** |
| GetItemTransactions | 2,283 | 399 | 2,289 | **3** | **4,971** |
| GetMemberMessages | 2,288 | 399 | 2,293 | **3** | **4,980** |
| GetMyeBaySelling | | | 2,283 | **1** | **2,283** |
| **Affected Requests** | **23,855** | **4,259** | **26,517** | | **16%** |

decision tree is then learned for each cluster, which is used to predict the cluster of new users. Similarity is defined in Zhang, Ding et al. (2011), & Kang, Liu et al. (2012) in terms of both QoS and functional attributes used to formulate queries to the registry. These attributes are related to the respective service invocations in order to construct the users' similarity model, which is then used to rank service recommendations. Template level patterns are additionally employed in the approach proposed in Rong, Liu et al. (2009) to improve service recommendation. The approach combines user similarity models with discovered dependencies between service compositions frequently observed, with the premise that dependency information between services is a strong indicative hint for web service selection. It first clusters clients according to service invocation patterns. Afterwards, a database containing all web service composition transactions of these users over a specified time interval is constituted, and mined using association techniques. Finally, the strongest association rules are used to improve the ranking of recommended services. Our work differs from the above mentioned by the criteria used to group users, the usage pattern level, the data mining techniques used, as well as the purpose of the application. We cluster clients based on detailed information about service invocation (i.e. features), whereas these works use functional or non-functional attributes used to find and invoke services. They also disregard the specific functionality invoked, concentrating in a much larger granularity, i.e. the whole service. We integrate in our patterns both user request and template levels, as (Rong, Liu et al., 2009). We also contribute in different ways of preparing invocation-related data to be clustered, through the proposed selection and transformation filters. Finally, although the profiles aim to support decisions about service evolution, they can suit very different applications such as service recommendation, service design, workload balancing, etc.

Template level composition patterns are the focus of works such as (Zhang, Yin et al., 2009; Wang, Wang et al., 2012). The architecture proposed by Zhang, Yin et al. (2009) suggests that the composition engine monitors and logs all requests to services involved in composition, together with timestamp and process identification. A graph that relates services is then constructed, where weights are assigned to the edges to determine the strength of the connection. As compositions are dynamic and evolve overtime, the approach considers specific time-horizons, and use time-related information to determine the weights. The elements of these graphs are clustered, using a graph-based clustering algorithm, in order to find closely related services that compose the so-called service communities, i.e. services that are frequently used in compositions. A semi-empirical composition approach is described by Wang et al. (2012) which aims at supporting on-line service recommendation for optimal compositions. First, the method clusters similar services (based on information available in the registry) and similar service requests (expressed as functional and non-functional properties in historical queries to the registry). Then, statistical analysis is employed to establish probabilistic correspondences between the two types of clusters. These clusters are used to recommend services during real-time service composition. The above mentioned works find clusters of related services, but do not associate them with the clients that invoked similar service compositions. Additionally, our work could be extended to consider profiles based on service composition patterns. For this purpose, instead of extracting only features of a single service version, one should extract the features of all service versions involved in the composition. This perspective must be further investigated.

The discovery of business workflows at template and request level is addressed in works such as in (Motahari-Nezhad, Saint-Paul et al., 2011), (Musaraj, Yoshida et al., 2010; Tang and Zou 2010), in an area more commonly referred to as *process mining* (van der Aalst, 2012). In this case, the patterns sought aim at documenting the actual processes that involve service compositions/business processes, checking the conformance between process instances and a

process model, or optimizing a process. These works focus on discovering the actual usage of a set of related services, but do not relate clients that invoke similar workflows.

Compatibility is addressed by many works that suggest compatibility rules or design guidelines (Andrikopoulos, Benbernou et al., 2012; Fokaefs, Mikhaiel et al., 2011), automatic compatibility assessment (Becker, Lopes et al., 2008; Becker, Pruyne et al., 2011), or functional components for handling versioned services (e.g. Fang, Lam et al., 2007). These approaches are conservative, in the sense that they always assume a worst-case scenario, i.e. clients that possibly will be impacted by incompatible changes. However client applications are bound to specific features within offered functionality. The larger the interface, the more distinct usage patterns it may subsume. Usage has been considered for purposes such as producing custom-made documentation (Zou, Fang et al., 2008) and discovering substitute services in an interoperability context (Ponnekanti & Fox, 2004). However, these works assume the point of view of a single client application, and how it can adapt itself to changes. Our point of view is the provider, who needs an understanding of the overall impact of changes in the whole set of clients applications to make sound decisions about service lifecycle. To the best of our knowledge, no similar usage-oriented assessment algorithm was proposed in the literature. The idea of usage-oriented compatibility was introduced in our early work (Yamashita, Becker et al., 2012). The approach is complimentary to our previous work on service compatibility assessment and quantification of change impact (Yamashita, Vollino et al., 2012).

## CONCLUSION 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.

We also introduced an algorithm for usage-oriented compatibility assessment to illustrate one of the possible uses of profiles within the service evolution context. It allows the detection of incompatible changes with regard to specific profiles at a very detailed level (types and operations). The service evolution framework provides analytical mechanisms that allow quantifying and analyzing such impact, in order to make better decisions about service evolution.

One of the major challenges involved in service mining is the availability of data due to its proprietary nature (Nayak, 2008; Motahari-Nezhad, Saint-Paul et al., 2011). Indeed, data is a valuable business asset, and frequently it is not available at public domain, nor it is released to other parties unless the return is clearly perceived. Our approach is feasible because it involves proprietary data of the party interested in knowing the profiles. However, further studies need to be developed in terms of the costs of collecting such detailed data, and the trade-offs with regard to the main type of analyses providers need to perform over profiles.

The KDD 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. However, the unsupervised metrics experimented do not seem the best ones, as they did not correlate to the supervised assessment technique for noisy data. As mentioned, we are currently implementing other internal indexes for further experimentations. 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.

As mentioned, there is a tremendous difficulty in obtaining real data due to proprietary ownership of data. Notice that we need a moderately complex service with at least hundreds of client applications. In our experiments, we generated data based on a real system, and explored different ways that such a service could be used according to the service documentation. This is a common approach for validating service mining research (e.g. Yu, 2012; Zhang, Ding et al., 2011; Rong, Liu et al., 2009; Zhang, Yin et al., 2009; Motahari-Nezhad, Saint-Paul et al., 2011).

The knowledge extracted by the proposed process cannot be derived merely by investigating the expected service workflows. Even when the provider expect interactions to follow a model, clients may not conform to it (van der Aalst, 2012), or it may include several variability points that enable one to derive at most the worst case scenario, rather than the actual one (Tang & Zou 2010).

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 (Yamashita, Vollino et al., 2012; Silva, Vollino et al., 2012). 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.

## ACKNOWLEDGMENT

We would like to thank Lucas Alves who has helped with the implementation. This research is financially supported by FAPERGS, CNPq and CAPES Brazil. This paper is an extended version of the one accepted as Research Paper at ICWS 2013 (Vollino & Becker 2013).

## REFERENCES

Aalst, W. van der. (2012). Service mining: Using process mining to discover, check, and improve service behavior. *IEEE Transactions on Services Computing, 99*(1).

Andrikopoulos, V., Benbernou, S., & Papazoglou, M. P. (2012). On the evolution of services. *IEEE Transactions on Software Engineering*, *38*(3), 609–628. doi:10.1109/TSE.2011.22.

Becker, K., Lopes, A., Milojicic, D., Pruyne, J., & Singhal, S. (2008). Automatically determining compatibility of evolving services. In *Proceedings of the 2008 IEEE International Conference on Web Services*, Beijing, China (pp. 161–168).

Becker, K., Pruyne, J., Singhal, S., Lopes, A., & Milojicic, D. (2011). Automatic determination of compatibility in evolving services. *International Journal of Web Services Research*, *8*(1), 21–40. doi:10.4018/jwsr.2011010102.

Chuvakin, A., & Peterson, G. (2009). Logging in the age of web services. *IEEE Security and Privacy*, *7*(3), 82–85. doi:10.1109/MSP.2009.70.

Fang, R., Lam, L., Fong, L., Frank, D., Vignola, C., Chen, Y., et al. (2007). A version-aware approach for web service directory. In *Proceedings of the 2007 IEEE International Conference on Web Services*, Salt Lake City, UT (pp. 406–413).

Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., & Lau, A. (2011). An empirical study on web service evolution. *Proceedings of the 2011 IEEE International Conference on Web Services*, Washington, DC (pp. 49–56).

Halkidi, M., & Vazirgiannis, M. (2001). Clustering validity assessment: Finding the optimal partitioning of a data set. In *Proceedings of the IEEE International Conference on Data Mining*, San Jose, CA (pp. 187–194).

Halkidi, M., Vazirgiannis, M., & Batistakis, Y. (2000). Quality scheme assessment in the clustering process. Principles of Data Mining and Knowledge Discovery (pp. 265–276).

Hall, M., Frank, E., & Holmes, G. (2009). The WEKA data mining software: An update. *ACM SIGKDD*, *11*(1), 10–18. doi:10.1145/1656274.1656278.

Kang, G., Liu, J., Tang, M., Liu, X., Cao, B., & Xu, Y. (2012). AWSR: Active web service recommendation based on usage history. In *Proceedings of the 2012 IEEE International Conference on Web Services*, Honolulu, HI (pp. 186-193).

le Zou, Z., Fang, R., Liu, L., Wang, Q., & Wang, H. (2008). On synchronizing with web service evolution. In *Proceedings of the 2008 IEEE International Conference on Web Services*, Beijing, China (pp. 329–336).

Liang, Q. A., Chung, J.-Y., Miller, S., & Ouyang, Y. (2006). Service pattern discovery of web service mining in web service registry-repository. In *Proceedings of the 2006 IEEE International Conference on e-Business Engineering*, Shanghai (pp. 286–293).

Liu, Y., Li, Z., Xiong, H., Gao, X., & Wu, J. (2010). Understanding of internal clustering validation measures. In *Proceedings of the IEEE International Conference on Data Mining*, Sydney (pp. 911–916).

Motahari-Nezhad, H., Saint-Paul, R., Casati, F., & Benatallah, B. (2011). Event correlation for process discovery from web service interaction logs. *The VLDB Journal*, *20*(3), 417–444. doi:10.1007/s00778-010-0203-9.

Musaraj, K., Yoshida, T., Daniel, F., Hacid, M.-S., Casati, F., & Benatallah, B. (2010). Message correlation and web service protocol mining from inaccurate logs. In *Proceedings of the 2010 IEEE International Conference on Web Services*, Miami, FL (pp. 259–266).

Nayak, R. (2008). Data mining in web services discovery and monitoring. *International Journal of Web Services Research*, *5*(1), 63–81. doi:10.4018/jwsr.2008010104.

Papazoglou, M. P., Andrikopoulos, V., & Benbernou, S. (2011). Managing evolving services. *IEEE Software*, *28*(3), 49–55. doi:10.1109/MS.2011.26.

Pfitzner, D., Leibbrandt, R., & Powers, D. (2009). Characterization and evaluation of similarity measures for pairs of clusterings. *Knowledge and Information Systems*, *19*(3), 361–394. doi:10.1007/s10115-008-0150-6.

Ponnekanti, S., & Fox, A. (2004). Interoperability among independently evolving web services. In *Proceedings of the 2004 International Middleware Conference*, Toronto, Canada (pp. 331–351).

Rong, W., Liu, K., & Liang, L. (2009). Personalized web service ranking via user group combining association rule. In *Proceedings of the 2009 IEEE International Conference on Web Services*, Los Angeles, CA (pp. 445–452).

Rousseeuw, P. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*(1), 53–65. doi:10.1016/0377-0427(87)90125-7.

Silva, E., Vollino, B., Becker, K., & Galante, R. (2012). A business intelligence approach to support decision making in service evolution management. In *Proceedings of the IEEE 2012 International Conference on Services Computing*, Honolulu, HI (pp. 41-48).

Tan, P.-N., Steinbach, M., & Kumar, V. (2006). *Introduction to data mining*. Boston, MA: Addison Wesley.

Tang, R., & Zou, Y. (2010). An approach for mining web service composition patterns from execution logs. In *Proceedings of the 12th IEEE International Symposium on Web Systems Evolution*, Timisoara (pp. 53–62).

Vollino, B., & Becker, K. (2013). A framework for web service usage profiles discovery. In *Proceedings of the 2013 IEEE International Conference on Web Services*, Santa Clara Marriott.

Wang, X., Wang, Z., & Xu, X. (2012). Effective service composition in large scale service market: An empirical evidence enhanced approach. *International Journal of Web Services Research*, *9*(1), 74–94. doi:10.4018/jwsr.2012010104.

Yamashita, M., Becker, K., & Galante, R. (2011). Service evolution management based on usage profile. In *Proceedings of the 2011 IEEE International Conference on Web Services*, Washington, DC (pp. 746–747).

Yamashita, M., Becker, K., & Galante, R. (2012). A feature-based versioning approach for assessing service compatibility. *Journal of Information and Data Management*, *3*(2), 120–131.

Yamashita, M., Vollino, B., Becker, K., & Galante, R. (2012). Measuring change impact based on usage profiles. In *Proceedings of the 2012 IEEE International Conference on Web Services*, Honolulu, HI (pp. 226-233).

Yu, Q. (2012). Decision tree learning from incomplete QoS to bootstrap service recommendation. In *Proceedings of the 2012 IEEE International Conference on Web Services*, Honolulu, HI (pp. 194-201).

Zhang, Q., Ding, C., & Chi, C. (2011). Collaborative filtering based service ranking using invocation histories. In *Proceedings of the 2011 IEEE International Conference on Web Services*, Washington, DC (pp. 195–202).

Zhang, X., Yin, Y., Zhang, M., & Zhang, B. (2009). Web service community discovery based on spectrum clustering. In *Proceedings of the 2009 International Conference on Computational Intelligence and Security*, Beijing, China (vol. 2, pp. 187-191).

## ENDNOTES

1. Developer.ebay.com/DevZone/XML/docs/WebHelp
2. jmeter.apache.org
3. www.w3.org/TR/wsdl
4. www.x.com/developers/ebay/products/trading-api

*Bruno Vollino is a graduate student at the Computer Science Institute of Universidade Federal do Rio Grande do Sul (UFRGS), Brazil. He obtained his B. degree in Computer Science from Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Brazil, in 2010. His current interests include data mining, knowledge discovery, software engineering and software development process.*

*Karin Becker received a Ph.D. degree in Computer Science from the Facultés Universitaires Notre-Dame de la Paix (Belgium), and a M.Sc. degree from UFRGS (Brazil). She holds a large background on research and development in both the academia and industry, mainly in the areas of data and web mining, software engineering and service computing. Since 2010, she is an Associate Professor at the Computer Science Institute of UFRGS, where she develops research projects in the area of business intelligence for supporting service evolution, and opinion mining. Her current interests are focused on the application of data mining techniques to web-related data (opinion mining, web services, social networks), as well as agile software development practices.*