Patiniotakis et al. Journal of Internet Services and Applications (2015) 6:26
DOI 10.1186/s13174-015-0042-4

Journal of Internet Services and Applications
a SpringerOpen Journal

RESEARCH Open Access

PuLSaR: preference-based cloud service selection for cloud service brokers

Ioannis Patiniotakis, Yiannis Verginadis* and Gregoris Mentzas

Abstract

Over the last few years, the vast increase of cloud service offerings that are available from heterogeneous cloud vendors, has made the evaluation and selection of desired cloud services, a cumbersome task for service consumers. In that respect, there is an increasing need for user guidance and intermediation during the service selection process but also during the cloud service consumption that should always refer to the best possible choice based on user preferences. In this paper, we discuss the Preference-based cLoud Service Recommender (PuLSaR) that uses a holistic multi-criteria decision making (MCDM) approach for offering optimisation as a brokerage service. The specification and implementation details of this proposed software mechanism are thoroughly discussed while the background method used is summarised. Both method and brokerage service allow for the multi-objective assessment of cloud services in a unified way, taking into account precise and imprecise metrics and dealing with their fuzziness. We cope with the fuzziness of imprecise metrics in the sense that this approach deals with linguistically expressed preferences and cloud service characteristics that lack a fixed or precise value and entail a level of vagueness which can only be captured using the Zadeh's Fuzzy Set Theory. Furthermore, this paper reports on a number of experiments that were conducted in order to measure PuLSaR's performance and scalability.

Keywords: Cloud Service Broker, Optimisation, Service Ranking, MCDM

1 Introduction

Nowadays, enterprises are increasingly moving their IT environments into the cloud, reducing operating costs by converting from a business model reliant on hardware and software ownership, to one based on utility service consumption. This has resulted in an unprecedented rise of cloud providers that serve their offerings as a service but at the same time has created additional challenges (e.g., regarding the quality-of-service [1], security etc.). Examples of such additional challenges include: i) conducting and monitoring service level agreements that refer to cloud applications composed of heterogeneous service offerings from different cloud service vendors, ii) efficiently comparing available cloud offerings, using appropriate methods that consider all the dimensions of cloud consumers' requirements but also the unified and fair use of comparable criteria across heterogeneously described services, iii) addressing in a unified way security aspects (e.g., access control, authentication etc.) of composed cloud applications from offerings that may comply to different vendor's standards. As the multitude and complexity of heterogeneous cloud services increases, the role of cloud brokers in the cloud service ecosystems becomes increasingly important. Technology analysts such as Gartner [2] and Forrester [3] foresee an increasing role for cloud service brokers, intermediaries who already offer related brokerage capabilities such as integration, customization or aggregation of software services. Their analysis regarding cloud service brokers is based on the fact that the unprecedented rise of available cloud services from different vendors, dictate the need for intermediating entities that their sole purpose will be to cope with the challenges mentioned above thus alleviate the heterogeneity existing among such offerings. Nevertheless, even in the emergent commercial cloud services, there is still lack of standard mechanisms that allow for the comparison of cloud service specifications against user

* Correspondence: jverg@mail.ntua.gr
Institute of Communications and Computer Systems, National Technical University of Athens, 9, Iroon Polytechniou Str., Zografou 15773, Greece

requirements, taking into account the implicit uncertainty and vagueness, during the cloud service evaluation and selection.

Recent research work has focused on developing methods and mechanisms to allow the comparison and ranking of competitive cloud services and help the user during the cloud service selection. According to the existing work [4, 5], service evaluation may be affected by a set of quantitative and qualitative service characteristics. Quantitative characteristics are those that can be measured without any uncertainty, e.g., response time, while qualitative characteristics refer mainly to non-functional service characteristics and cannot be quantified in an objective manner, as they are based typically on user experience such as service usability. Although the existence and significance of qualitative characteristics are identified, existing approaches up to now do not provide models and methods to handle qualitative service characteristics in an efficient and objective way. Furthermore, current approaches use quantitative models to insert user requirements [4]. However, imprecise models are closer to the human needs when expressing preferences, since they can capture the vagueness of the user requirements. We define an imprecision model as a set of qualitative cloud service metrics that cannot be objectively quantified or measured. These metrics can be used for both describing a cloud service and expressing requirements during the cloud service selection phase. This imprecision can be entered in the cloud consumer's requirements side even if it refers to quantitative metrics. For instance, while availability is a quantitative metric, it is obviously more intuitive for the user to express her requirements by using expressions such as High or Medium, rather than specifying precise numerical thresholds. For that reason for the rest of the paper, we use the notion of precise metrics/criteria and imprecise metrics/criteria for either describing a cloud service or capturing the cloud consumer's requirements. The precise metrics refer to those that include only crisp values (i.e., quantitative/measurable without any uncertainty) while the imprecise metrics refer to those that cannot be objectively quantified or measured and usually include fuzzy or linguistic values, for both describing and expressing a requirement for a cloud service offering. Fuzzy numbers that we argue they can be used for expressing imprecise criteria, are based on Zadeh's pioneer work on Fuzzy Set Theory [6]. Fuzzy sets are sets of ordered pairs $A = \{(x, \mu_A(x)), x \in A, \mu \in \mathbb{R}\}$, where $\mu(x)$ is called the membership function. They extend the notion of membership of an element in a set, from binary (belongs or not belongs) to a grade of membership, expressed as a real number, usually in *[0,1]* interval.

In this paper, we aim to tackle the aforementioned limitations in cloud service ranking and thus optimize cloud service use, by providing a preference-based cloud service recommender as a brokerage service that allows cloud service evaluation based on a heterogeneous model of service characteristics. We focus on imprecise metrics and on a unified method to manage them along with the precise ones for providing cloud service rankings. Examples of such imprecise metrics include: *Provider's Brand Name Reputation* - the linguistic expression of the reputation of the cloud service provider as perceived by its consumers; *Support Satisfaction* – the linguistic expression that indicates the aggregated perception of the cloud consumers regarding the support that they have received from the vendor; *Provider's business stability* – the fuzzy description of the likelihood that the service provider will continue to exist throughout the contracted term. Cloud consumers need to declare their preferences in a way that retains their inherent vagueness, such as using linguistic terms, which are easier, more intuitive and more comprehensible than using numbers. In order to cope with all the meaningful metrics that should be used for an optimised use of cloud service offerings, we developed the Preference-based cLoud Service Recommender (PuLSaR). PuLSaR is a cloud consumer preference based recommender that uses a holistic multi-criteria decision making approach for offering optimisation as brokerage service.

This paper is structured as follows: In Section 2, we present the related state-of-the-art while in section 3, a discussion on the meaningful attributes for comparing cloud service offerings and optimizing their selection and use, is given. In Section 4, we summarise our proposed cloud service recommendation method, while in Section 5, we propose a formal way to express preferences using a Linked Unified Service Description Language (Linked – USDL) extension. In Section 6, we give technical details about the preference-based cloud service recommender. Section 7 accommodates the details of our proposed mechanism's performance evaluation and we conclude with the next steps in Section 8.

2 Related work

The majority of the existing approaches focus on optimisation issues before or during the cloud migration process from the legacy systems and mostly target cost optimisation issues [7] in terms of resource allocation and deployment [8]. For example, Litoiu et al. [7] have presented a business driven cloud optimization architecture, called CERAS that enables cross-layer services optimization which considers platform profit and hardware utilization as main optimization goals. They

implement approximate optimization using available techniques such as linear programming, to balance complex business factors and come close to optimal profits at all cloud layers. Similarly, Huu [8] proposed four different resource allocation strategies in an effort to reduce infrastructure costs while also optimizing application performance. Such efforts, mainly target Infrastructure as a Service (IaaS) level optimisation issues from the perspective of the cloud provider. Other efforts, that examine cloud service optimisation from the cloud consumer perspective, focus on comparing low level performance aspects of cloud services such as CPU and network throughput [9] addressing mainly IaaS-related issues [5, 10] or they are concerned with comparing similar services based on mainly cost issues [11]. Although these works aim to identify optimal cloud services from a cloud consumer perspective by considering user preferences, they are limited because they only consider quantitative metrics and they do not use advanced MCDM strategies to solve the multi-criteria optimisation problem. Similarly, in another interesting approach [12], the authors proposed an indexing technique for managing the comparable information of a large number of cloud service providers and developed an efficient service selection algorithm that ranks potential service providers, instead of their offerings. In particular, they proposed the Cloud Service Provider (CSP) index which was built based on an encoding technique that captures similarity among various properties of service providers. But, this work, although it is offered as a cloud brokerage service, it focuses only on ten properties (e.g., service type, security, operating system etc.) of the cloud service providers by considering one offering per provider thus limiting this ranking service to infrastructure and platform as a service offerings. Moreover, there is no support for any vagueness in the expression of cloud consumers' requirements, even if they refer to qualitative properties.

In other more recent works, there is some effort to consider qualitative metrics and try to address the inherit vagueness that exists in the cloud service optimisation by adopting MCDM techniques [13–15]. In one of the most interesting approaches, Garg et al. [4] present a framework for comparing and ranking cloud services based on QoS requirements and on current performance measurements of services' attributes. Their work is based on the Service Measurement Index (SMI), which is a set of business-relevant key performance indicators (KPI's) that provide a standardized method for measuring and comparing business services [16]. Garg et al. [4], try to quantify some of these KPIs in order to model several quality dimensions of cloud services. They use an Analytic Hierarchy Process (AHP) [17] based ranking mechanism to solve the multi-criteria decision making problem of finding the optimal cloud service. This quantification however, although helpful, in many cases it seems arbitrary and vague as mentioned in the introduction section.

In this paper, we cope with such inefficiencies or arbitrary assumptions by providing a more realistic approach that takes under consideration the implicit vagueness in certain criteria along with the fuzziness when dealing with user's preferences or requirements, expressed linguistically. Therefore, we use fuzzy and linguistic values that are more appropriate for dealing with imprecise criteria. Moreover, in contrast to the above-mentioned work that is using eigenvectors, we use extent analysis [18] for deriving the relative value-based weights for ranking Cloud services. This compared to the conventional AHP, makes it simpler and easier to implement the calculation of weights, for prioritizing consumer satisfaction. For optimisation purposes, we consider as criteria values the ones that appear in the Linked-USDL based service description regarding the precise criteria, while we use the aggregated user feedback for imprecise criteria values. Moreover the approach in [18] uses single instance performance analysis focusing only on IaaS aspects while do not consider any aggregation of measurable values.

3 Attributes for cloud service optimisation

In order to develop a software system, capable of comparing cloud service offerings and optimizing their selection and use during any phase of the cloud service lifecycle, an appropriate model for describing their comparable characteristics is required. This model should encapsulate all the necessary performance indicators that will allow for comparisons between cloud services. From the state-of-the-art-analysis, it is evident that a lot of efforts have been dedicated to exactly this issue. One of the most accepted and cited work is the Service Measurement Index (SMI) [16]. SMI is currently being developed by the Cloud Services Measurement Initiative Consortium (CSMIC) and involves a set of business-relevant Key Performance Indicators (KPI's). SMI is a hierarchical framework that divides the measurement space into 7 top level categories [16] that are further refined by three or more level of attributes. These top level categories are:

- *Accountability* - measures the properties characterizing the cloud service provider organizations related to standards, processes, and policies that they follow;
- *Agility* - indicates the impact of a service upon a client's ability to change direction, strategy, or tactics quickly and with minimal disruption;

- *Assurance* - indicates how likely it is that the service will be available as specified;
- *Financial* – relates to the amount of money spent on the service by the client;
- *Performance* – covers the performance features and functions of the provided services;
- *Security and Privacy* - indicates the effectiveness of a cloud service provider's controls on access to services, service data, and the physical facilities from which services are provided;
- *Usability* - relates to the ease with which a service can be used by its consumers.

It is evident from the literature review (see section 2) that most of the research efforts focus only on quantitative metrics (or at best on quantifiable metrics), thus only crisp numbers are used in the methods and techniques implemented for ranking cloud services [4]. However, sometimes it can be hard to classify the characteristics in one of the two categories, since even for some quantitative attributes it makes sense that the users express their preferences in a qualitative manner. There are a number of metrics that can be seen as qualitative but at the same time with some reasonable assumptions they can be quantified (e.g., Interoperability, Usability etc.) or they can be resolved in a number of lower level metrics, involving both quantitative and qualitative attributes (e.g., *Serviceability* - the ease and efficiency of performing maintenance and correcting problems with the service) or involving both precise and imprecise values. For instance, the Usability metric has been defined as a quantifiable attribute [4] in the sense of the average time experienced by users of the cloud service to install, learn, understand and operate it. But, often this average time is not enough to define how usable a cloud service is, since this information is often vague and imprecise. It might be the case that the average learning time for a cloud customer about a specific service is relative short because of the customer's huge experience in the specific domain and not because the service is really usable for an average user. It would be an oversight to ignore the degree of difficulty that other users experience based on their degree of expertise, when they tried to learn, understand and operate the specific cloud service. This value is highly subjective, uncertain and often is available through linguistic terms when previous users are expressing their opinions. In this example the experienced user might indicate that it is difficult to operate the specific cloud service (for an average user) although it didn't take her too long to learn how to operate it.

Therefore, we believe that a blending of precise and imprecise metrics is more meaningful for characterizing and ranking a cloud service. We adopt and extend the SMI model [16] in order to use widely acceptable criteria for cloud service ranking and selection. The extensions made are mostly related to imprecise attributes and took into account the state of the art and the relevant needs of real pilot cloud platforms (i.e., CAS Open of the German company CAS and Orbi of the Greek company Singular Logic) that were formulated after three series of interviews. In Fig. 1, an overview of the hierarchical structure of the defined criteria is presented. Because of the size of its current version not all the attributes can be depicted. The reader can find the complete list of attributes used, online here: http://imu.ntua.gr/software/pulsar. The main extensions over SMI [16] involved the introduction of a new top-level attribute called Reputation. It is related to the reputation of the Service provider and of the cloud service offerings that are provided and involves the following 2nd level attributes: Brand name, Service Reputation, Contracting Experience, Ease of doing business, Provider business stability, Provider Ethicality and Sustainability. For example, the Brand name attribute refers to the linguistic expression of the reputation of the cloud service provider as perceived by its consumers. The rest of the extensions involved the introduction of 2nd level imprecise attributes: Robustness, Monitoring, Reusability under the Agility, Performance, Usability top level attributes, respectively and the introduction of 3rd level imprecise attributes: Technical competency of the support employees, Support Satisfaction and Documentation, Interoperability level and Monitoring level under the Accountability, Assurance and Performance top level attributes.

4 Cloud service recommendation method

The proposed method aims at providing a cloud service ranking technique capable to exploit both crisp and fuzzy information in order to be used for optimisation purposes. It extends the SMICloud approach [4]. Based on our method the service KPI and user requirement values can be fuzzy numbers and intervals, or linguistic terms. In the latter case, linguistic terms are mapped onto fuzzy numbers in order to ensure unified processing, both of the imprecise and precise, user-provided values. Using techniques similar to SMICloud's, we derive fuzzy comparison matrices and subsequently using a fuzzy Analytical Hierarchical Process (AHP) method, we rank services. For example, this method is able to cope in unified way with the following heterogeneous criteria values:

– *service response time* expressed as an integer in the cloud service descriptions (e.g., 20msec) and as a fuzzy number when a potential cloud consumer provides her requirements (e.g., (10,20,25) msec). We note that it is the Broker's responsibility that the

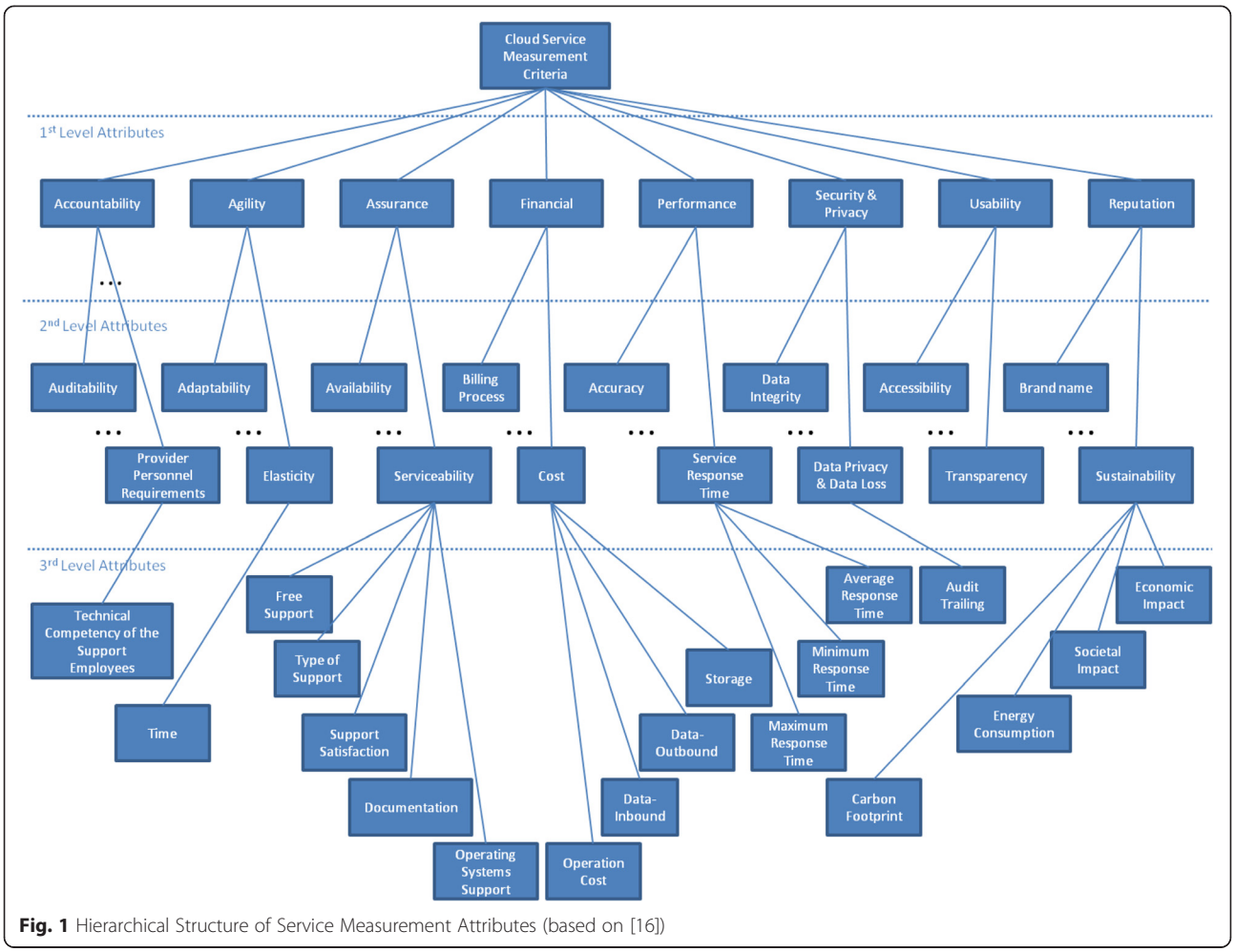

Fig. 1 Hierarchical Structure of Service Measurement Attributes (based on [16])

measurements are acquired in a way that allows for fair comparisons between different services of different providers. This is achieved by setting the appropriate broker policy that dictates for example the use of similar probes thus considering the network latency in all response times.

– *support satisfaction* expressed as a linguistic value for both the cloud service description (e.g., High) and requirement (e.g., ≥Medium). It refers to the consumers' aggregated satisfaction (expressed linguistically) when performing maintenance or correcting problems with the service. The algorithm considers this criterion by mapping the linguistic values to fuzzy numbers.

In our work we have also selected the method presented in [18], appropriately adapted for service ranking purposes. In addition, we have also chosen to use triangular fuzzy numbers and trapezoidal intervals due to their simplicity and broad use. Our approach provides more expressive and unified way to capture user opinions and preferences, both precise and imprecise, than traditional service ranking methods. This proposed service ranking method involves the following four phases that have been thoroughly discussed in [19]:

- *Phase 1: Expressing ranking problem into a hierarchical structure*
- *Phase 2: Computation of relative quality of service (QoS) attribute weights*
- *Phase 3: Computation of relative service performances*
- *Phase 4: Aggregation of relative service weights*

Next, we are presenting a high-level view of the service ranking algorithm (Fig. 2), where the main steps of this process are depicted, while we provide excerpts of the algorithm in pseudocode for revealing additional important details (Algorithm 1). The thorough description of our cloud service recommendation method can be found in our previous work [19].

Algorithm 1. Service Ranking Algorithm in Pseudocode

```
1.  Input: profile, criteria_hierarchy
2.  Output: list of services
3.  criteria ← get_criteria(profile)
4.  services ← get_services(profile)
5.  pairs ← get_comparison_pairs(profile)
6.  /* Generate and populate criteria comparison matrices */
7.  comparison_matrices ← {}
8.  crit_set ← clone_of(criteria)
9.  foreach crit in crit_set do {
10.     siblings ← get_siblings(crit, criteria_hierarchy) ∩ crit_set
11.     if sizeof(siblings)>1 then {
12.         comp_mat ← create_comparison_matrix(siblings)
13.         populate(comp_mat, pairs)
14.         comparison_matrices ← comparison_matrices U {siblings, comp_mat}
15.         crit_set ← crit_set / siblings
16.     }
17. }
18. /* Calculate aggregate relative criteria weights */
19. criteria_weights_matrix ← {}
20. foreach {siblings, comp_mat} in comparion_matrices do {
21.     weights_vector ← calculate_relative_weights(comp_mat)
22.     criteria_weights_matrix ← criteria_weights_matrix U {siblings, weights_vector}
23. }
24. criteria_weights ← aggregate_weights(criteria_weights_matrix, criteria_hierarchy)
25. /* Filter out services not meeting required performance values */
26. foreach crit in criteria do {
27.     type ← get_criterion_type(crit)
28.     scale ← get_criterion_scale(crit)
29.     mandatory ← is_criterion_mandatory(crit)
30.     required_value_range ← get_criterion_required_value_range(crit)
31.     foreach S in services do {
32.         S_kpi ← get_service_attribute_value(S, crit)
33.         normalize(S_kpi, type, scale)
34.         if mandatory and not is_in(S_kpi, required_value_range) then {
35.             services ← services / {S}
36.         }
37.     }
38. }
39. /* Calculate relative service weights per criterion */
40. service_pairs ← get_service_pairs(services)
41. comp_mat ← create_comparison_matrix(service_pairs)
42. weights_matrix ← {}
43. foreach crit in criteria do {
44.     type ← get_criterion_type(crit)
45.     scale ← get_criterion_scale(crit)
46.     mandatory ← is_criterion_mandatory(crit)
47.     required_value_range ← get_criterion_required_value_range (crit)
48.     foreach pair in service_pairs do {
49.         S1 ← pair.service1
50.         S2 ← pair.service2
51.         S1_kpi ← get_service_attribute_value(S1, crit)
52.         S2_kpi ← get_service_attribute_value(S2, crit)
53.         normalize(S1_kpi, type, scale)
54.         normalize(S2_kpi, type, scale)
55.         comp_mat[S1][S2] ← calculate_relative_service_importance(S1_kpi, S2_kpi, type)
56.         comp_mat[S2][S1] ← fuzzy_invertion(comp_mat[S1][S2])/* (1 / comp_mat[S1][S2]) */
57.     }
58.     weights_vector ← calculate_relative_weights(comp_mat)
59.     weights_matrix ← weights_matrix U {crit, weights_vector}
60. }
61. /* Rank services */
62. overall_weights ← aggregate_weight_matrices(weights_matrix, criteria_weights)
63. sort_services_by_weight(services, overall_weights)
64. return { (Si,wi) | Si in services, wi in overall_weights, such that 'i' in [1..length] of services }
```

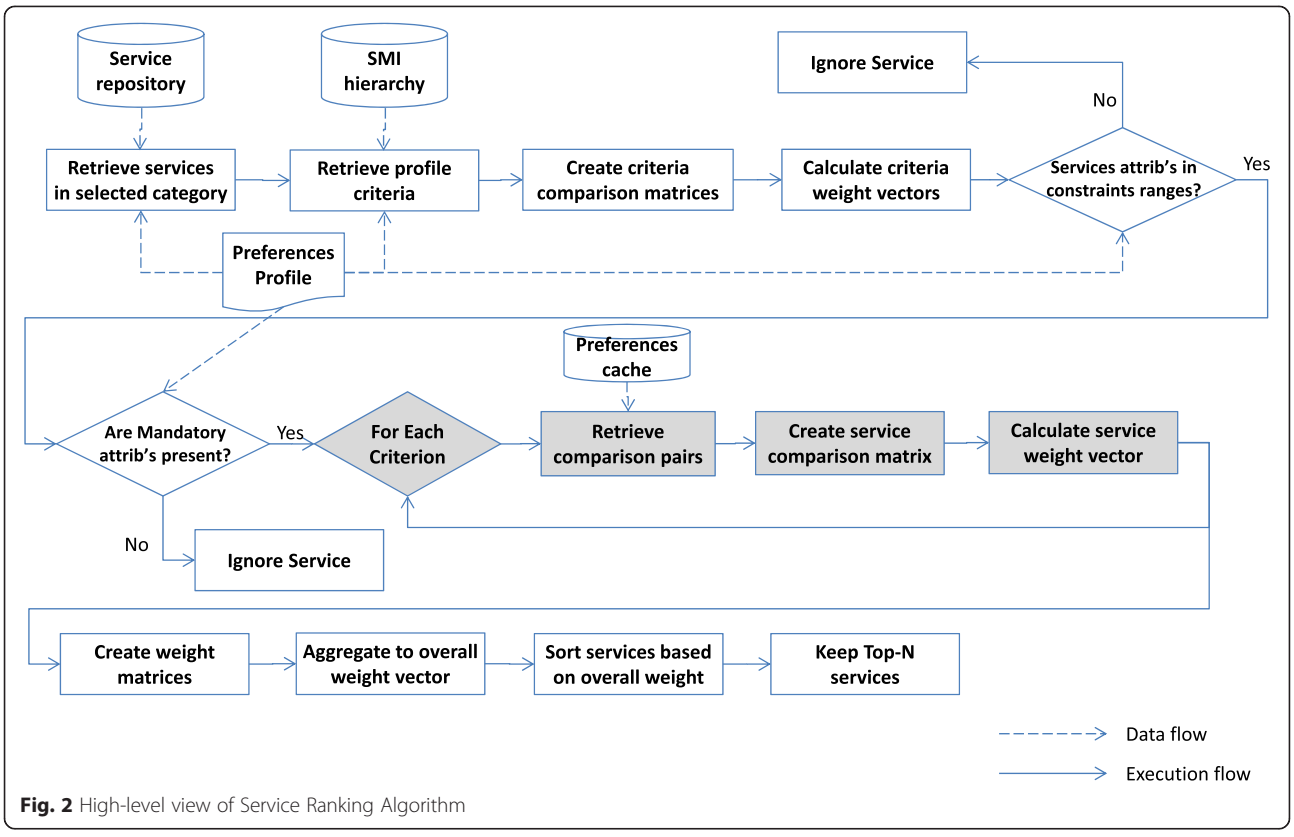

Fig. 2 High-level view of Service Ranking Algorithm

However, the ranked list of cloud services calculated based on this approach is not yet suitable for recommendation to the service consumer directly. This is due to several reasons. For instance, some of the services in the recommended list might have been recently replaced by another cloud application, due to adaptation actions initiated by a failure prevention and recovery mechanism that a broker may offer. Another case is that the service consumer has repeatedly ignored a suggested service (in the context of a previous recommendation). For the aforementioned reasons the ranked list returned from service ranking algorithm needs to be filtered first before it can be presented to service consumer.

The proposed method also provides the capability to define a selection policy on the ranked and filtered list of services. The selection policy picks a certain number of services from the ranked and filtered list and creates a recommendation that is sent to the service consumer. Specifically, the selection policy defines the number of items (services) included in the recommendation, for example the top one or top-3 or top-N services. Selection can also occur based on a score, taking into account the relative service weight calculated during service ranking. It is possible to define a relevance threshold and filter out services with lower scores. The threshold can either be a specific value or a percentage of the value of the top ranked service. For example, let the top ranked service has score 0.37 and let the percentage be set to 20%. Then all services with scores from 20% below 0.37 (i.e., 0.296) up to 0.37 will be accepted as recommendable items. Furthermore, service consumer might also send a response (to the recommendation event) indicating that he/she accepts certain recommended services. These responses can be used to filter out services from future recommendations.

5 Formally expressing preferences

In order to implement the above mentioned cloud service recommendation method, it is desired to be able to formally express and register cloud consumers' preferences. Therefore we are using and appropriately extending Linked-USDL [20], as an adequate, acceptable and easily extensible ontological framework for describing services. We propose Linked USDL Preferences – a novel Linked USDL schema which aims at: (i) providing an open, linked, and interoperable framework for capturing consumer-expressed preferences with respect to a set of precise and imprecise attributes that characterise cloud services; (ii) establishing a clear relationship between consumer preferences, service-level profiles, and SDs. The Linked USDL Preferences schema takes into account the vagueness, or fuzziness, often implicit in consumer-expressed preferences, whilst allowing for the intuitive expression of preferences using linguistic terms

and imprecise values. This is particularly important as the majority of research efforts in modelling consumer preferences predominantly focus on attributes that can be measured precisely, i.e., in terms of crisp numbers [4]. The basic concepts of the Linked-USDL Preferences schema are outlined below.

A cloud consumer is potentially associated with a number of diverse preference profiles, assumed to be drawn from the class *ConsumerPreferenceProfile* (see Fig. 3). Each such profile aggregates, through the *hasPref* property, the consumer's preferences relative to a particular *functional category* of cloud services – i.e., of a grouping of cloud services of comparable functionality, e.g.,: 'CRM Apps', 'Project Management Services', 'Sales Services' etc. . Consumer preference profiles are associated with their corresponding functional categories via the *adheresTo* property. Now a consumer preference is invariably expressed in terms of the *OptimisationAttribute* class. This class is assumed to comprise all service attributes used for optimisation purposes. In fact, a consumer preference expresses a constraint on such an attribute – one which is represented here in terms of a preference expression. Such an expression is associated with the attribute that it constrains via the property *hasOptAttr*. Consumer preferences are associated with their corresponding expressions via the property *hasPrefExpr*.

Optimisation attributes may be either precisely or imprecisely measurable. Precisely-measurable attributes may characterise functional aspects of a service, in which case they are associated, through the *belongsTo* property (see Fig. 3), with the functional category to which the service pertains. Precisely-measured attributes are assumed to draw their values from the GoodRelations class *gr: QuantitativeValue*. Analogously, imprecisely-measured attributes draw their values from the class *gr: QuantitativeValue*, and fuzzy attributes draw their values from the class *FuzzyValue*.

Each preference variable in a consumer's preference profile is additionally associated with a *weight* which indicates its significance relative to the rest of the preference variables in the same profile. In this respect, the Linked USDL Preferences model introduces the class *Weight*, a subclass of the *gr: QuantitativeValueFloat* class, which draws its instances from the range [0,1].

Clearly based on this model, different preference profiles may be associated with diverse sets of preference variables and hence different consumer preferences. Moreover, a consumer may express different preferences for the same cloud service depending on the particular circumstances under which the service is consumed; this may occur, for example, when the same cloud service is consumed as part of two different cloud applications.

6 Preference-based cloud service recommender

6.1 Architecture

In this section we discuss the conceptual architecture of PuLSaR. Certain technical choices that have been made are depicted in Fig. 4. This conceptual architecture encompasses both the subcomponents of PuLSaR as well as the interactions with "external" components that may exist in a modern Broker [21]. These external to PulSaR components involve the service governance and quality control for assuring quality in all the brokerage phases

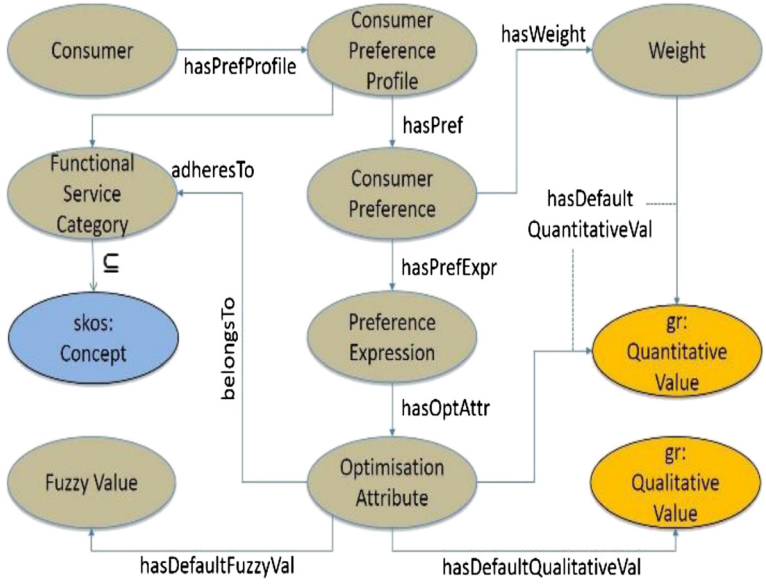

Fig. 3 Linked USDL Preferences Model

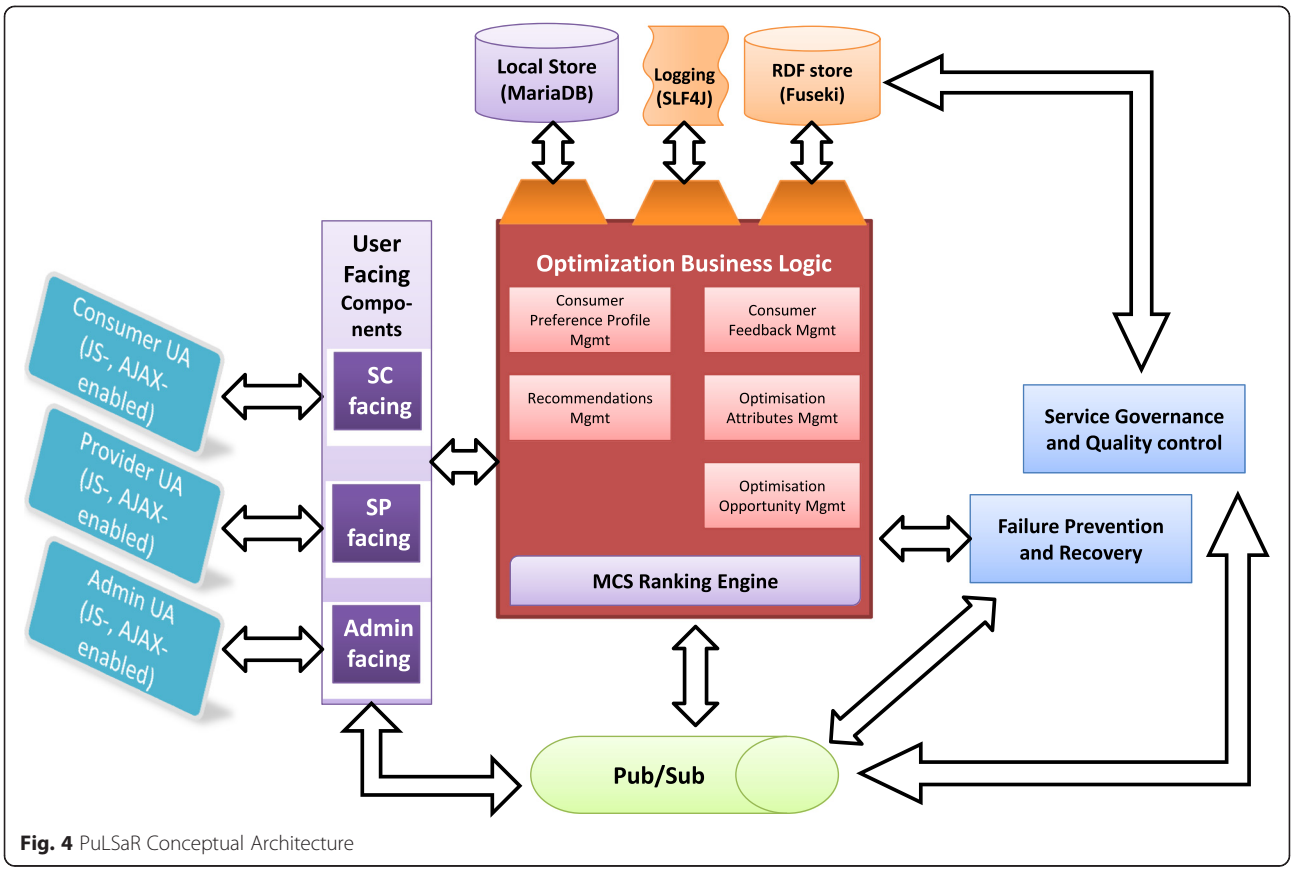

Fig. 4 PuLSaR Conceptual Architecture

and the failure prevention and recovery mechanism for adapting failing cloud service offerings.

If PuLSaR is embedded into or integrated with another system, such as a broker's platform, the user-facing components and user agents (considered as external to PuLSaR) are expected to be substituted from platform specific services or parts. The user-facing components can also co-exist with the services of a hosting system in order to provide any functionality missing from the hosting system (for instance consumer preference profile management).

The core of PuLSaR is shown as a central box in the architectural diagram. It encompasses six subcomponents, which implement the relevant functionalities and mechanisms. Five of them provide the means for humans or external software to interact with PuLSaR whereas the sixth one implements the ranking algorithm. Therefore, PuLSaR involves the following subcomponents presented below.

Consumer Preference Profile Management subcomponent. It provides APIs and functionality for creating, updating and deleting consumer preference profiles. Consumer preference profiles are logical containers where all selection preferences and configuration pertaining to a specific selection problem are stored. A service consumer can have any number of preference profiles pertaining to different applications and purposes. These profiles are subsequently used by the Multicriteria Cloud Service ranking engine to rank services and generate recommendations. Each preference profile provides a different set of selection criteria and options therefore leading to a possibly different service ranking, even for the same service selection problem.

Consumer Feedback Management subcomponent. It is responsible for collecting service consumer feedback for services she has already used. It generates questionnaires based on imprecise selection criteria and gives the consumer the opportunity to provide her subjective opinion about them. Of course, the service consumer is asked to provide feedback only for services she has already used. Furthermore, this subcomponent is able to aggregate different consumers' opinions and upon reaching a parameterized threshold of participation, it can provide an average value for the related imprecise attributes and notify providers accordingly. For example, this means that if for a certain cloud service the imprecise attribute "Support Satisfaction" has value "Very High" but PuLSaR aggregates the value "Medium" for more than 80% of its consumers (based on their feedback) then the corresponding provider will be notified about the opinion of its cloud consumers with respect to the certain attribute. Using this subcomponent, consumers may also update their previous feedback.

Recommendations Management subcomponent. It is used to store new recommendations and collect consumer responses to recommendations. More specifically, this subcomponent receives ranked lists of services from the Multicriteria Cloud Service ranking engine and it subsequently filters them according to the selection policy chosen. To this end, it can contact a failure prevention and recovery component in order to check if any of the top ranked services have recently been adapted in order to be omitted by the PuLSaR recommendations list.

Optimisation Attributes Management subcomponent. It is used to manage the optimisation attribute model used by PuLSaR. Currently, this model is based on and extends [16] work on service selection attributes (criteria). This subcomponent provides search-create-retrieve-update-delete (SCRUD) operations on that model.

Optimisation Opportunity Management subcomponent. It is responsible for subscribing, receiving and processing Service Life-cycle Management (SLM) events from the platform and initiating the recommendation process if appropriate. More specifically, when Service On-boarded (i.e., made available through the Broker platform), Service Deprecated, Service Description changed, then SLM events are published through an event bus. Based on these events the subcomponent checks if any existing service with an associated preference profile is affected (i.e., deprecated or updated or an alternative service has been on-boarded). In any such case it calls the Multicriteria Cloud Service ranking engine passing the relevant SLM event and consumer preference profile.

Multicriteria Cloud Service Ranking engine. This subcomponent implements the service ranking algorithm. It is called either by the Optimisation Opportunity Management subcomponent (as a reaction to a relevant SLM event) or by Consumer Preference Profile Management subcomponent in order to provide on-demand recommendations and assist the service consumer during profile development or maintenance.

The subcomponents of PuLSaR rely on a few infrastructural capabilities for their operation that are discussed below.

RDF store. The Jena Fuseki server is used. Fuseki server accepts SPARQL queries and statements via HTTP protocol and forwards them to an underlying RDF store. This gives considerable flexibility to the overall platform architecture since the RDF store can be accessed remotely, shared with other components, replaced seamlessly and it can also abstract the actual RDF store intricacies, since only standard SPARQL is exchanged with Fuseki.

RDF persistence. We have implemented an RDF persistence framework that converts java object instances into sets of RDF triplets of the form (subject, predicate, object). Based on them it builds SPARQL queries or updates, which are then sent to Fuseki server for processing. Vice versa, the results returned from Fuseki server are converted back to java objects.

Local data store. It is used to store information pertaining to PuLSaR and not shared or exchanged with other components (for instance consumer feedback or responses to recommendations). It also caches temporary information, intermediary calculations or data of technical nature. In the context of PuLSaR we have chosen MariaDB database as the local data store.

Logging framework. PuLSaR logging mechanism relies on the well-known SLF4J logging abstraction mechanism for Java.

Pub/Sub mechanism. It is used to send and receive events to/from other Broker platform components. WSO2 pub/sub is used for exchanging information between components.

As seen in Fig. 4, PuLSaR should be able to interact with the following Broker platform components:

Service Governance and Quality Control (SGQC). This component indirectly interacts with PuLSaR since it is responsible for checking and uploading broker policies and service descriptions to the fuseki store. Such information is queried by PuLSaR before issuing recommendations. In addition, any update on service descriptions (i.e., new service onboarded) is published by the SGQC to the Pub/Sub system for propagation in the form of SLM events. Such events may trigger new PuLSaR recommendations.

Failure Prevention and Recovery component (FPR). This component is contacted in order to retrieve the recommendations it has recently sent to service consumers in order to avoid recommending cloud services that have been recently adapted. FPR component can contact PuLSaR, through its RESTful API, in order to retrieve PuLSaR recommendations already sent to the consumers, in order to use this information for appropriately adjusting any adaptation actions.

User-Facing components. These components constitute the front-end of PuLSaR. They provide GUIs that enable users of a Broker platform to access information and use its services. User-facing components can either be stand-alone applications or they can be part of (or integrated to) a third-party brokerage or cloud provider infrastructure. At present, user-facing components have been implemented as stand-alone web-based applications using Web2.0 technologies like JavaScript, DOM, XHR and JSON format for the exchanged data. Currently, we consider three major user roles: service consumers, service providers and platform administrator(s). These facts are depicted in PuLSaR architecture diagram (Fig. 4) as the purple boxes “SC facing”, “SP facing” and “Admin facing” components (denoting the server-side part of them), as well as “Consumer User Agent (UA)”, “Provider UA” and

"Admin UA" cyan boxes (denoting the client-side part of them, presented to the corresponding user roles).

6.2 Illustrative walkthrough of using PuLSaR

PuLSaR can enhance any cloud platform with brokerage capabilities with respect to optimized consumption of cloud offerings as long as the platform is able to describe in a neutral way the available cloud services and the broker policy that needs to be enforced. In this work, Linked-USDL has been used for that specific purpose. Based on the expressed broker policy, PuLSar can undertake the optimisation attributes management. This enables the maintenance of the optimisation attribute model used to describe services in terms of selection criteria. This means that the offered model with service attributes discussed in section 2, can be specialised for each brokerage platform and instantiated according to the broker's hosting platform policies (e.g., Learnability is a linguistic attribute with allowed values {Low, Medium, High}). After this, the prototype implementation performs the collection of service consumer preferences on specific service selection problems expressed and captured as Consumer Preference Profiles. PuLSaR guides consumers to interactively develop their consumer preference profiles, add selection criteria and assign importance values to them (as weights) through a series of pairwise comparisons. Specifically, based on the selected classification dimensions all the associated service attributes are presented in order for the consumer to declare his choices (Fig. 5 - screen 1). Next, the consumer is invited to declare the weights of the selected attributes through pairwise comparisons (Fig. 5 - screen 2). It is possible to display recommendations based on the current settings of the profile being edited thus helping consumers to pick a service for the initial composition of their application. PuLSaR presents the calculated weights and allows for defining constraints over them (Fig. 5 - screen 3). Last, it generates recommendations, either on-demand (during consumer preference profile creation) or in response to an SLM event (Fig. 5 - screen 4). In both cases, recommendations are stored in the local data store and published as recommendation events to the corresponding event

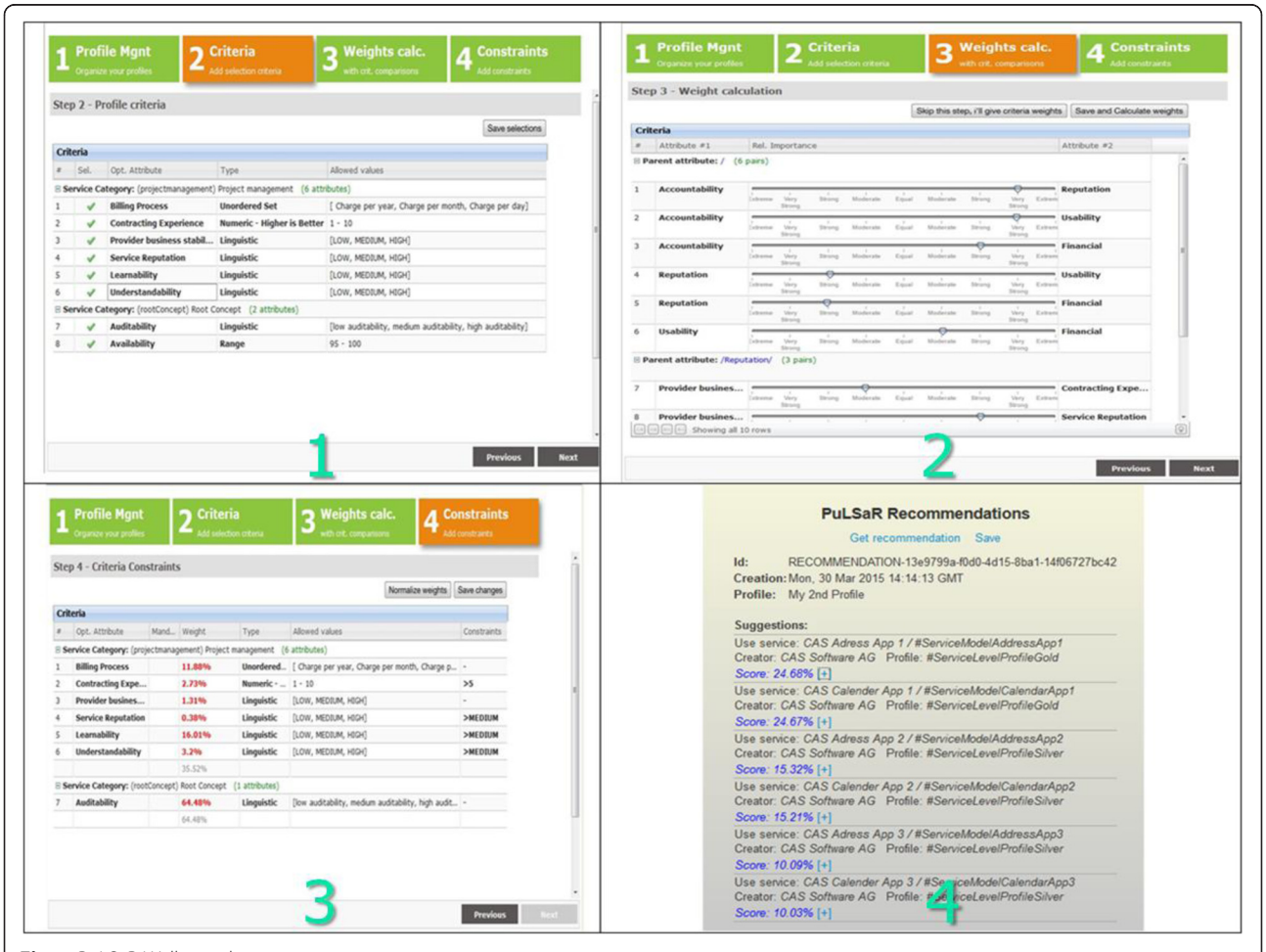

Fig. 5 PuLSaR Walktrough

topics. These recommendations are based on the ranking of cloud services based on the consumer preference profiles.

The PuLSaR prototype was implemented in Java using the Apache Jena (Core and ARQ) APIs and was released as open source under the Apache Dual License. The reader can find PuLSaR open source, detailed illustrative walkthrough and explanatory videos here: http://imu.ntua.gr/software/pulsar.

7 Evaluation of PuLSaR

In order to evaluate the performance of PuLSaR against an increasing set of requirements that could be met in a cloud service broker, we conducted a number of experiments and measured certain key performance indicators (KPI's) that assess the behaviour of PuLSaR. These experiments involve an increasing number of fabricated consumer preferences profiles (i.e., 1–1000) that trigger PuLSaR to issue recommendations for finding the optimal cloud offerings in a hypothetical SaaS cloud marketplace. The size of available offerings also varies (i.e., 1–1000) while linguistic (imprecise) attributes are used to describe offerings, since they require more complex operations than numeric attributes during processing. The two variant aspects (profiles and cloud services, keeping the selected attributes fixed to 10) constitute 8 experiments that we conducted. For each of these variants the CPU, memory and time consumed for issuing PuLSaR recommendations were measured. Specifically, these 8 experiments have clustered to two series of experiments. The variable increased in each series of these experiments, respectively, is:

1. *the number of consumer preference profiles to be processed* - Tests for 1, 10, 100 and 1000 profiles were executed, keeping other parameters fixed (i.e., available service descriptions to 10 and 10 criteria per profile)
2. *the number of available service descriptions in the triplestore* - Tests for 1, 10, 100 and 1000 service descriptions were executed, keeping other parameters fixed (i.e., profiles to be processed to 10 with 10 criteria per profile)

All of our experiments have been performed on an Ubuntu SMP, kernel version 3.13.0-49-generic, installed on a virtual machine with 8 cores of Intel(R) Xeon(R) 64-bit CPU E5-2470 v2 @ 2.40GHz, 25600 KB cache and 10GB of RAM. Java version is 1.7.07 OpenJDK 64-bit server VM. It should be also noted that after some preliminary tests, PuLSaR has been enhanced with multi-threading capabilities in order to deal with scalability issues evident in the cloud computing domain.

In the results of the first series of tests, involving an increasing number of consumer preference profiles, it is evident that test duration (i.e., processing all profiles) increases linearly as the number of profiles climbs up (Fig. 6a). This is the expected behaviour since each profile is handled separately from the others. The only data shared between profile processing iterations (and threads) are service descriptions and broker policy elements, which are cached the first time they are retrieved from Fuseki. Adding extra cores would reduce the inclination of the trend line. Memory consumption also exhibits a linear behaviour as profiles in this test require roughly the same amount of memory (Fig. 6b) with the exception of the beginning of the test where a lot of profile pre-fetching and queuing takes place, constituting steeper inclination. CPU load is measured both for the whole system and for PuLSaR (JVM process) separately. Fuseki and PuLSaR were collocated in the same platform during tests, therefore system load gives an idea of the impact of PuLSaR requests onto Fuseki. Figure 6c suggests that system and PuLSaR CPU loads exhibit similar behaviours although system load is higher than that of PuLSaR. This is reasonable since Fuseki serves requests (and consumes CPU) at the rate PuLSaR submits them. Figure 6c also indicates increased CPU load for low profile numbers (1 and 10). This effect stems from the logging strategy used and the fact that PuLSaR retrieves and queues profiles at the beginning of the experiment.

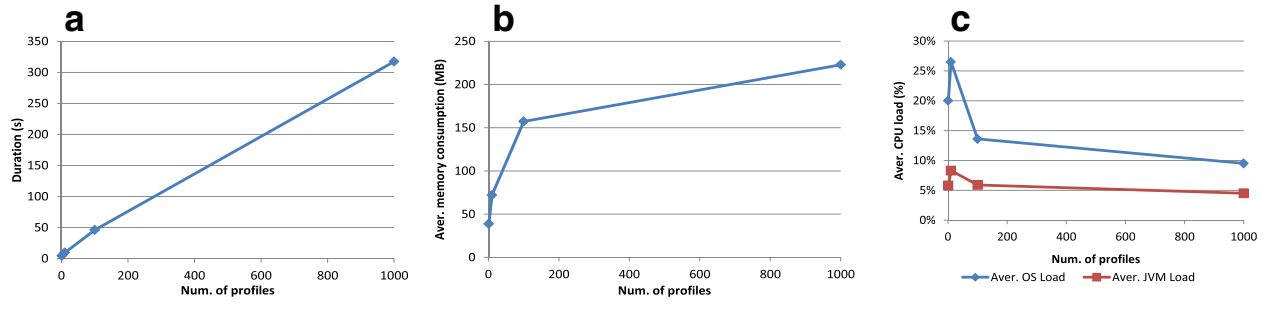

Fig. 6 First Series of Evaluation Tests, (**a**) Duration vs. Profiles, (**b**) Av. mem. consumption vs. Profiles, (**c**) Av. CPU load vs. Profiles

Regarding the second series of tests involving an increasing number of cloud offerings it is obvious that test duration increases abruptly as their number climbs up (Fig. 7a). This is due to the nature of the algorithm which requires N(N−1)/2 pairwise comparisons of cloud offerings per criterion, where N is the number of offerings. For instance if N = 10 then only 45 comparisons are required but if N = 100 then 4950 comparisons are required, which is 110 times larger than 45. Obviously this step's complexity is $O(N^2)$ thus leading in significant increase of test duration as cloud offerings increase from 100 to 1000. For low numbers of cloud offerings (<10) test duration seems to remain unchanged (or climbing slowly) because overhead is comparable to or exceeds the processing time of services and profiles. Memory consumption seems to increase in a linear fashion with the number of cloud offerings, since their descriptions are cached and in this test require roughly the same amount of memory (Fig. 7b). The CPU usage of PuLSaR increases up to a point (approx. at 100 offerings) and then remains constant (Fig. 7c). As the number of offerings increases the cloud offerings comparison step of the recommendation algorithm, takes considerably more time than the other steps, therefore we conjecture the maximum software throughput (processing speed) is reached at around that point, for the specific test. Hence CPU usage becomes almost constant. The system CPU usage increases with the number of cloud offerings up to around 100 and then drops to reach the average PuLSaR usage (Fig. 7c). This behaviour occurs because PuLSaR retrieves all needed information from Fuseki at the initial phase of the recommendation process and then continues with offerings comparisons. In small numbers these phases may overlap between simultaneously executing threads causing both PuLSaR and Fuseki compete for CPU time. Contrary, in higher numbers the cloud offerings' comparisons step take considerably more time (during which only minimal interaction with Fuseki might occur) than other information retrieval activities and Fuseki seems to become idle leaving PuLSaR occupy most of CPU time. Therefore the system CPU load appears to initially increase up to 100 offerings and then decrease down to PuLSaR average CPU usage level, which as already explained becomes constant at high offering numbers.

Since PuLSaR uses processing threads to create recommendations for consumer preference profiles, it is possible to increase its throughput (profiles processed per unit of time) by making available extra cores and heap memory to it. Also, in terms of memory consumption PuLSaR exhibits a linear behaviour whereas in terms of CPU load it reaches a plateau and remains steady. These findings suggest that PuLSaR can acceptably scale up and operate in increasing numbers of offerings and profiles.

8 Conclusions

In this work we presented a method and a tool for optimising the cloud service usage by performing cloud service evaluations based on a heterogeneous model of service characteristics. Specifically, we presented and implemented a fuzzy AHP approach that solves the problem of service ranking and allows the multi-objective assessment of cloud services. This approach provides a more expressive and unified way to capture and process user opinions and preferences (both precise and imprecise) than traditional service ranking methods. Based on this method and a proposed Linked-USDL model for formally expressing preferences, we designed and implemented the Preference-based cLoud Service Recommender (PuLSaR). PuLSaR is a cloud consumer preference based recommender that uses a multi-criteria decision making approach for offering optimisation as brokerage capability. The specification and implementation details of this dedicated software component were presented. In addition, PuLSaR was evaluated with respect to performance though a number of experiments that measured certain KPIs of the mechanism. Based on the evaluation findings it became evident that PuLSaR can be considered as an efficient and scalable software mechanism that can enhance as a service any cloud service broker with optimisation capabilities.

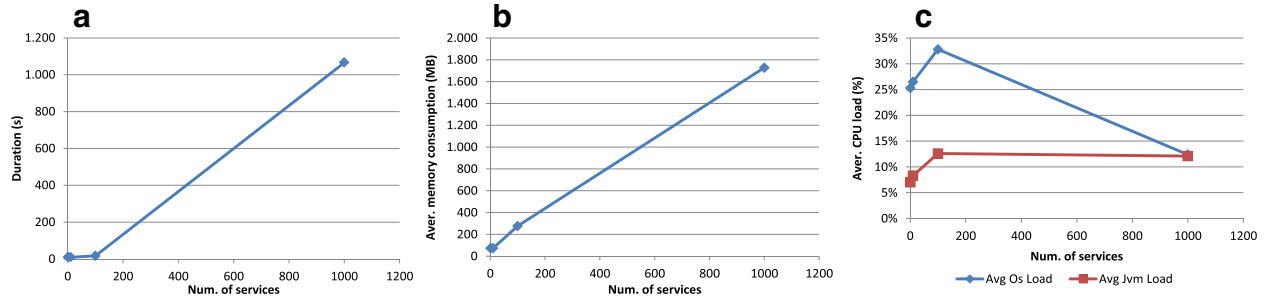

Fig. 7 Second Series of Evaluation Tests, (**a**) Duration vs. Services, (**b**) Av. mem. consumption vs. Services, (**c**) Av. CPU load vs. Services

The next steps of this work, involve the evaluation of PuLSaR's optimisation capabilities in real-life brokerage scenarios using heterogeneous hosting platforms and enriching them with valuable brokerage capabilities. PuLSaR's capabilities will be enhanced in real cloud platforms that aspire to add brokerage functionalities in their offerings. Such evaluation will be done with respect to usability issues by conducting surveys targeting real cloud service consumers. Additionally, PuLSaR will be extended to support distributed processing (i.e., several PuLSaR processing nodes working in parallel), thus further increasing scalability. Eventually, extensions of the proposed method are considered to the direction of reducing the number of cloud offerings pairwise comparisons and the faster calculation of relative weights.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
IP has participated in the design of this study, led the implementation work of the PuLSaR software component and conducted the evaluation experiments. YV has contributed to the conception and implementation of this study, led the Linked-USDL extension work and contributed in the design and analysis of the evaluation experiments. GM have made substantial contributions to the conception and design of PuLSaR software component, led the state-of-the-art analysis and drafted the manuscript. All authors read and approved the final manuscript.

Acknowledgments
The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013), the Broker@Cloud project (www.broker-cloud.eu). The authors would like to thank the project partners for their valuable feedback.

Received: 29 April 2015 Accepted: 24 November 2015
Published online: 07 December 2015

References

1. Ardagna D, Casale G, Ciavotta M, Pérez J, Wang W. Quality-of-service in cloud computing: modeling techniques and their applications. J Internet Serv Appl. 2014;5(1):1–17.
2. Gartner. Defining Cloud Services Brokerage: Taking Intermediation to the Next Level. 2010. https://www.gartner.com/doc/1448121/defining-cloud-services-brokerage-taking. Accessed 1 Jun 2015.
3. Forrester. Cloud Brokers Become Change Agents: Understanding The Cloud Broker Opportunity. 2012. www.forrester.com/Cloud+Brokers+Become+Change+Agents/fulltext/-/E-res71622. Accessed 1 Jun 2015.
4. Garg SK, Versteeg S, Buyya R. A framework for ranking of cloud computing services. Future Gen Comp Syst Elsevier. 2013;29:1012–23.
5. Godse M, Mulik S. An Approach for Selecting Software-as-a-Service (SaaS) Product. In: International Conference on Cloud Computing, CLOUD'09. Bangalore, India: IEEE; 2009. pp 155–158.
6. Zadeh LA. Fuzzy sets. Inf Control. 1965;8(3):338–53.
7. Litoiu M, Woodside M, Wong J, Ng J, Iszlai G. A business driven cloud optimisation architecture. In: Proceedings of the 2010 ACM Symposium on Applied Computing, SAC '10. Sierre, Switzerland: ACM, New York; 2010. 380–385.
8. Huu TT, Koslovski G, Anhalt F, Montagnat J, Primet PV-B. Joint elastic cloud and virtual network framework for application performance-cost optimisation. Grid Computing J. 2011;9(1):27–47.
9. Iosup A, Ostermann S, Yigitbasi N, Prodan R, Fahringer T, Epema D. Performance analysis of cloud computing services for many-tasks scientific computing. IEEE Trans Parallel Distrib Syst J. 2011;22(6):931–45.
10. Chintapalli VR. A deadline and budget constrained cost and time optimisation algorithm for cloud computing. In: Advances in Computing and Communications in Computer and Information Science 193. Berlin, Heidelberg: Springer; 2011. pp 455–62.
11. Rehman ZU. Towards Multi-criteria Cloud Service Selection. In: Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. Seoul, South Korea: IEEE; 2011. pp 44–48.
12. Sundareswaran S, Squicciarini A, Lin D. A brokerage-based approach for cloud service selection. In: 5th International Conference on Cloud Computing, CLOUD'12. Honolulu, USA: IEEE; 2012. pp 558–565.
13. Shivakumar U, Ravi V, Gangadharan GR. Ranking cloud services using fuzzy multi-attribute decision making. In: International Conference on Fuzzy Systems, FUZZ'13. Hyderabad, India: IEEE; 2013. pp 1–8.
14. Qu L, Wang Y, Orgun MA. Cloud Service Selection Based on the Aggregation of User Feedback and Quantitative Performance Assessment. In: 10th International Conference on Services Computing, SCC'13. Santa Clara, USA: IEEE; 2013. pp 152–159.
15. Baliyan N, Kumar S. Quality Assessment of Software as a Service on Cloud Using Fuzzy Logic. In: International Conference on Cloud Computing in Emerging Markets, CCEM'13. Bangalore, India: IEEE; 2013. pp 1–6.
16. CSMIC. Cloud Service Measurement Index Consortium: SMI framework Version 2.1. 2014. http://csmic.org/downloads/SMI_Overview_TwoPointOne.pdf. Accessed 1 Jun 2015.
17. Saaty TL. A scaling method for priorities in hierarchical structures. J Math Psychology. 1977;15(3):234–81.
18. Chan KY, Kwong CK, Dillon TS. An enhanced Fuzzy AHP method with extent analysis for determining importance of customer requirements computational intelligence techniques for new product design. In: Springer-Verlag, editor. Computational Intelligence Techniques for New Product Design, 403. 2012. p. 79–93.
19. Patiniotiakis I, Rizou S, Verginadis Y, Mentzas G. Managing Imprecise Criteria in Cloud Service Ranking with a Fuzzy Multi-Criteria Decision Making Method. In: Proceedings of the European Conference on Service-Oriented and Cloud Computing, vol 8135. Malaga, Spain: Springer Berlin Heidelberg; 2013. pp 34–48.
20. Pedrinaci, C, Cardoso, J, Leidig, T. Linked USDL: a Vocabulary for Web-scale Service Trading. In: Proceedings of the 11th Extended Semantic Web Conference, ESWC'14. Crete, Greece: Springer International Publishing; 2014. 8465:68–82.
21. Veloudis S, Paraskakis I, Friesen A, Verginadis Y, Patiniotakis I, Rossini A. Continuous Quality Assurance and Optimisation in Cloud-Based Virtual Enterprises. In: Proceedings of the 15th IFIP Working Conference on Virtual Enterprises, PRO-VE'14. Amsterdam, Netherlands: Springer Berlin Heidelberg; 2014. pp 621–632.