2015 IEEE International Conference on Mobile Services

Web Service Migration using the Analytic Hierarchy Process

M. Mohanned Kazzaz
Department of Information Systems
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
Email: ikazzaz@fit.vutbr.cz

Marek Rychlý
Department of Information Systems
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
Email: rychly@fit.vutbr.cz

***Abstract*—In this paper, we present a framework for Web service migration in Service-oriented Architecture (SOA). The framework utilizes service migration between devices acting as Web service providers to increase the adaptability of SOA in a mobile environment. It allows an automatic discovery of new providers and their services joining a system and extraction of their context, preferences, and rules for a Web service migration ontology. If the preferences and the rules, which specify requirements of each provider to its hosted services and requirements of the services to be hosted by a provider, do not meet the current status and context of providers or services, the framework initiates a Web service migration. In this case, the ontology is added to a core domain model and reasoned together by Jena reasoners to retrieve a set of alternate migration decisions. These are processed by Analytic Hierarchy Process (AHP) decision-making method to find the best possible Web service migrations which modify the status and context of providers and services to better meet their preferences and rules. By the Web service migration, the framework extends significantly the adaptability of SOA systems and helps to keep a required quality of their services (QoS).**

***Keywords*-service-oriented architecture; Web service; service migration; devices profile for Web services; Web service discovery**

I. Introduction

Software mobility [1], which is the ability of moving software across the nodes of a network, is a very interesting approach addressing many issues of distributed systems such as interoperability, self-adaptability, fault-tolerance, or load balancing. In Service-oriented Architectures (SOA), software mobility can be comprehended as service migration, which is the ability of services to be moved across the service providers at runtime. As services in SOA should be designed with respect to SOA principles [2], such as re-usability and statelessness, software mobility, that is service migration, in the case of SOA is much easier than in the case of non-SOA distributed systems.

In our previous work [3], we proposed an ontology-based framework for Web service migration in SOA. We defined an ontology-based semantic context model of migratable Web services and their providers. The context model allows to describe in the OWL/RDF format status properties of migratable Web services, properties of their providers, and also preferences and rules which specify requirements of each provider to its hosted services and requirements of the services to be hosted by a provider. We also proposed utilization of the context model with Apache Jena reasoners and rule engines for RDFS and OWL inference to make migration decisions, that is (1) to detect a situation when the current status or context of the providers or the services does not meet their requirements and some of the services need to be migrated and (2) to find services which should be migrated and where they should be migrated.

The situation above can arise from two reasons: a migratable service with particular requirements on its runtime environment is currently hosted by a provider which does not meet the requirements, or a provider with particular requirements on hosted services is currently hosting a migratable service which does not meet the requirements. Moreover, this situation results into a set of migration decisions which indicate possible services to migrate and possible destination service providers of the migration. Unfortunately, the reasoners and rule engines for RDFS and OWL inference produce all the feasible migration decisions, i.e., all services which need to be migrated and all providers which are able to host such services after the migration, and are not able to determine which migration decision is best and will fix the most important broken requirements.

This paper deals with the problem of selection of the best service migration decisions from a set of found migration decisions produced by the RDFS/OWL reasoners and rule engines. The selection process is based on the Analytic Hierarchy Process (AHP, [4]) multi-criteria decision making method.

The paper is organized as follows. Section II discusses related work on service migration in SOA. In Section III, we describe our Web service migration framework with focus on the migration decision process. Section IV deals with the utilization of AHP method in the best migration selection. An example of the migration decision making and selection process with AHP is provided in Section V. Finally, we draw conclusions in Section VI.

II. Related Work

Several works directly address or touch on the decision making issues in migration of services in SOA. For example, in [5], a service migration approach was proposed as a solution

978-1-4673-7284-8/15 $31.00 © 2015 IEEE
DOI 10.1109/MS.2015.64

to maintain continuous service availability with migration decisions based on Quality of Service (QoS) goals. In [6], authors introduced a virtual machines migration framework with several migration decision making strategies for different resource reservations goals and schemes during migration of virtual machines, namely: sequential migration, parallel migration and workload-aware migration strategies. In [7], authors proposed an algorithm for dynamic placement of services to servers based on available server resources (such as CPU or memory) and network performance given by SLAs.

In [8], authors described a framework for service migration in cloud computing environments using a genetic algorithm to search and select possible migrations. The algorithm utilized a cost model with various service migration costs, including the costs of consistency maintenance and communication during migrations, a service table with information of all migrated and replicated services, and a general computing platform registry with information about hosted services. Another approach to selection of migratable services in a cloud was proposed in [9]. The selection process considered pre-defined criteria related to QoS of the migratable services in the cloud, namely: response time, throughput, availability, reliability, and cost, and it utilized the Analytic Hierarchy Process (AHP, [4]) method with comparison matrices defined by a consumer's judgments on the QoS criteria. Although the AHP method is utilized also in our approach presented in this paper, the comparison matrices are, in our case, defined by the ontology of dynamic properties and preferences of automatically discovered providers and their services, which supports a multi-criteria migration decision making process.

In [10], a decentralized migration approach was introduced based on monitoring of health of Web Services Resource Framework (WSRF) containers [11]. The approach was trying to minimize possible threats of service level agreements (SLA) violations to preserve QoS of provided services by their migration. A service priority was proposed to be used as a weighting factor in migration decisions in addition to a health metric of each WSRF container. However, this approach was limited by the static health metric of WSRF containers based on their available memory and CPU performance factors, while, in our paper, we are addressing this limitation by dynamic service or provider properties which enable considering property values in different weighting factors during the migration decision process.

III. Web Service Migration Framework

In this section, a Web service migration framework will be introduced with focus on the migration decision process. Other parts of the framework have been already described in more details in [12] and [3].

A. Context Model

Migratable services and service providers which support service migration have to publish status properties to describe their status (e.g., the importance of a service, or available resources of a provider) and preference rules to describe their

```
<hasProperty>
  <!-- contains the related properties. -->
  <ExampleProperty>
    <!-- An example of service's or provider's property -->
    <propertyValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
      <!-- The value of the property is noted here. -->
    </propertyValue>
    <propertyType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      <!--
        The type of the property is noted here. i.e., "INT"
      -->
    </propertyType>
    <criteria rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      <!--
        The 'criteria' node specifies a criterion that controls this property.
      -->
    </criteria>
  </ExampleProperty>
</hasProperty>
<rules>
  <!--
    contains service's/provider's preferences represented in JENA rules.
  -->
  <rule>
    <!-- [JENA rule: ...] -->
  </rule>
</rules>
<noPreferenceRules rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
  <!--
    "true" if there is 'rules' node in the model, "false" if there is not
  -->
</noPreferenceRules>
```

Fig. 1. A simplified schema of the partial model for services and providers.

preferences (e.g., preferred types of services hosted by a provider, or resources required by a service) which should be considered in the migration decision making process. These properties and rules are defined in an ontology-based *context model* [3], which consists of a system core model and partial models of individual services and service providers.

The *system core model* provides the terminology used to define system components of *Service* and *ServiceProvider* classes including description of their properties, relationships, and subclasses. The partial models define status properties and preference rules of *Service* and *ServiceProvider* instances representing services and their providers currently available in a system, as it is described in Figure 1.

For a particular service or service provider, the *hasProperty* element contains a list of status properties of the service or the provider. Each property, which is formally a subclass of the *Property* class in the service migration ontology, has its data-type, value, and a set of criteria. Each property can be mapped by to several criteria which describe its importance for different parties in service migration and which are used later to select the best migration by AHP method (see Section IV).

Besides the *hasProperty* element, the service or service provider model defines also the *rules* element with a list of preference rules written in the Jena rules language. Each rule defines a pre-condition that has to be met by service providers which will be hosting the service, in the case of the service model, or by services which will be hosted by the service provider, in the case of the service provider model. Finally, the *noPreferenceRules* element just indicates that there are some preference rules in the model (if it is set to *true*) or there are not any such rules (if it is set to *false*). Existence of the *noPreferenceRules* element, as it does bring any new information, just improves performance (i.e., decreases complexity) of processing of the model in Jena.

In the case of a service, the information according to the model is integrated in the service's WSDL description, i.e., a standardized description of a Web service interface which is managed by common tools of a SOA implementation platform (an application server). In the case of a service provider, the information according to the model has to be published by the provider by *getContext* method of the provider's grounding service *ProviderContextWS*.

B. Service Provider Discovery and Service Migration

When a new service provider emerges into a network of a system which is utilizing the framework and the emerging service provider is able to participate in service migration in the system, that means both to pass its migratable services to other available providers and to accept migratable services from other providers if needed, the provider should be automatically discovered and considered in future migration decisions. To enable the automatic discovery of emerging service providers, the Devices Profile for Web Services[1] (DPWS) standard has been utilized by integration of WS4D.org Java Multi Edition DPWS Stack[2] (WS4D-JMEDS).

DPWS specifies a minimal set of built-in services which needs to be implemented in order to: enable discovery of service providers; support the meta-data exchange for publishing services provided by individual, previously discovered, service providers; and enable publish/subscribe communication model for producing and consuming asynchronous event messages by the services. WS4D-JMEDS is a light-weight DPWS stack that implements DPWS build-in services mentioned above on Java-based and Android platforms. By means of DPWS/WS4D-JMEDS, our framework is able to discover emerging service providers represented by mobile devices (e.g., tablets or cell phones) and their migratable services, and to retrieve information about the service providers and services needed to build the context model as it was described in Section III-A and as it will be utilized later during the migration decision making process described in this section.

The architecture of the Web service migration framework is depicted in Figure 2. The emerging service providers are discovered by *searchForDevices()* method of *Controller* class provided by our framework. In this method, a new instance of JMED *DefaultDevice* class is created for each discovered service provider and a new instance of JMED *DefaultService* class is created for each mobile or immobile service hosted by the provider. Then, the instances of both classes *DefaultDevice* and *DefaultService* are added into the ontology-based context model by calling *addDevicesToModel* and *addServicesToModel* methods, respectively.

Later, the ontology-based context model is processed by Jena reasoners. The resulting inferred model contains information on *possibleProvidedService* for each service provider, which lists all known services which can be provided in future by a particular service provider, and *possibleDestinationProvider*

[1]http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01
[2]http://ws4d.e-technik.uni-rostock.de/jmeds/

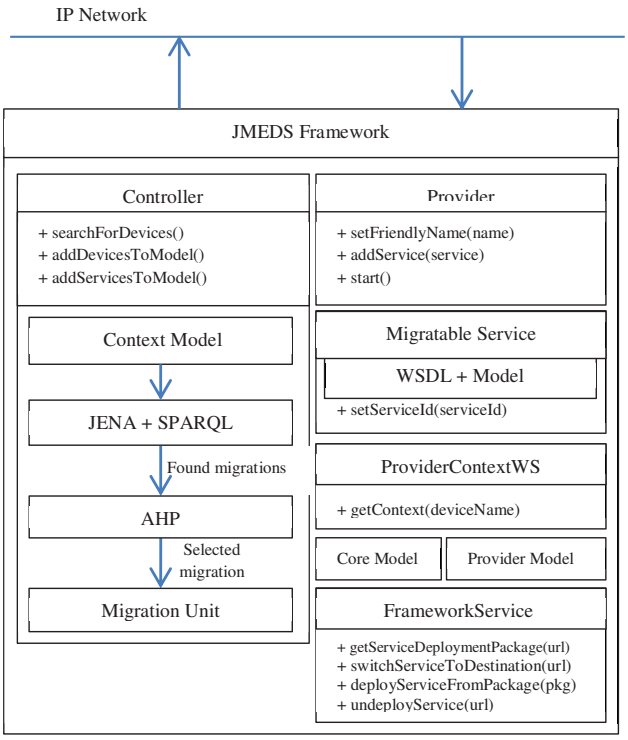

Fig. 2. Architecture and modules of the Web service migration framework.

for each service, which lists all known service providers which can provide a particular service in future. Finally, a SPARQL query is used on the inferred model to find such services and providers which satisfy both *possibleProvidedService* and *possibleDestinationProvider*, i.e., to find providers which can host given services and also can server in potential service migrations as destinations for the migrating services. In other words, by this query, we find a set of all possible migrations.

To find the best migrations in the set of all possible migrations, we choose to implement the Analytic Hierarchy Process (AHP, [4]) decision-making method. Application of this method is described later in Section IV. The results of this method, i.e., the best migrations, are processed and their services are migrated to corresponding destination providers. The migration of a migratable service from a source provider to a destination provider is performed in class *FrameworkService* by calling its methods *getServiceDeploymentPackage* on the source provider, *switchServiceToDestination* on the service, *deployServiceFromPackage* on the destination provider, and *undeployService* on the source provider. These methods are called sequentially to manage packaging the service (i.e., storing its implementation and internal status in a package), moving it to the destination provider (moving the package over network), deploying the service on the destination provider (deploying from the package and restoring the last-known internal state of the service), and removing the original instance of the service from the source provider.

IV. AHP IN THE MIGRATION DECISION PROCESS

In the migration decision process, the Analytic Hierarchy Process (AHP, [4]) method is utilized to select the best migrations in the set of all possible migrations previously found by ontology reasoning on the context model. The best migrations are those migrations which address (and fix by the service migration) violations of services or service providers preference rules with the highest priority, i.e., such rules that are based on the most critical properties of the services or service providers, respectively.

To introduce the priorities of the preference rules of services and service providers based on the criticality of their properties, the ontology-based context model described in Section III-A has been further extended by *decision making criteria*, which are defined in the system core model and referred from individual properties in partial models of the services and service providers (we can say that the properties are controlled by referred criteria; see element *criteria* in Figure 1). Formally, the criteria are instances of *CriteriaProperty* class with the following attributes: *name*, *owner*, *valueWithHighestWeight*, *valueWithLowestWeight*, and *criteriaPriority*.

For each criterion, the *name* attribute contains a name of the criterion (which is utilized in the references from properties in the partial models). Attribute *owner* can be one of values "origin", "destination", and "service", and limits possible owners of the properties which are referring to a particular criteria (for example, a criterion with the owner value set to "service" can be referred only from service context models properties, i.e., it can be applied only on services, not on service providers). Attributes *valueWithHighestWeight* and *valueWithLowestWeight* indicate which values of properties referring a particular criteria are considered to be the most important and the least important, respectively, while attribute *criteriaPriority* indicates a general importance of a particular criteria by integer values between 1 and 10. These last three attributes then control the overall importance of properties referring the particular criteria and determine the priority of related preference rules.

To select the best migrations, possible migrations previously found are processed by AHP in the *Controller* module (see Figure 2) that extends each input migration by its weight which is calculated based on the priorities of the preference rules of services and service providers participating in the migration. *Controller* then selects the migrations with the highest weight. The AHP starts with creating comparison criteria matrix A, which is an $m \times m$ matrix of real numbers where m is the number of considered criteria. In the matrix, a_{ij} is the importance of the i^{th} criterion over the j^{th} one (diagonal entries a_{ii} are set to 1). If the i^{th} criterion has the same importance as the j^{th} one, a_{ij} entry is set to 1, otherwise values of a_{ij} entry range over 3, 5, 7, or 9, which indicate that the i^{th} criterion is slightly more important, more important, strongly more important, or absolutely more important, than the j^{th} criterion, respectively.

Usually, in existing applications of the AHP method, entries of the matrix above are set directly through a user's judgment. In our case, we can automate this process by utilization of *decision making criteria* (and the related properties and preference rules published by services and service providers) to compute individual a_{ij} entries of the matrix. We use the *InitializeCriteriaMatrix* algorithm (see Figure 3) to calculate the a_{ij} entries in the upper-right triangular part of matrix A by comparing criteria priorities and in the lower-left triangular part of matrix A as reciprocal values of the symmetric entries in the upper-right triangular part. The algorithm guarantees the consistency of judgments between the criteria and satisfies a consistency ratio condition of comparison matrix A to be less than 10% as required for AHP.

Require: $\langle p_1, p_2, \ldots, p_m \rangle$ as values of *CriteriaPriority* of criteria $\langle c_1, c_2, \ldots, c_m \rangle$
Ensure: A is a pair-wise criteria comparison matrix for given criteria $\langle c_1, c_2, \ldots, c_m \rangle$
1: **for** $i \leftarrow 1$ **to** m **do**
2: $a_{ii} \leftarrow 1$
3: **for** $j \leftarrow i{+}1$ **to** m **do**
4: $difference \leftarrow |p_i - p_j|$
5: **if** $difference \geq 8$ **then**
6: $judgment \leftarrow 9$
7: **else**
8: $judgment \leftarrow 2\lfloor \frac{difference}{2} \rfloor + 1$
9: **end if**
10: **if** $p_i > p_j$ **then**
11: $a_{ij} \leftarrow judgment$
12: **else**
13: $a_{ij} \leftarrow judgment^{-1}$
14: **end if**
15: $a_{ji} \leftarrow a_{ij}^{-1}$
16: **end for**
17: **end for**
18: **return** A

Fig. 3. The *InitializeCriteriaMatrix* algorithm to compute a pair-wise criteria comparison matrix for AHP based on *CriteriaPriorities* of individual criteria.

After initializing A matrix, AHP computes matrix $\bar{A}$ by normalizing A entries to make the sum of each column entries equals to 1 through equation

$$\bar{a}_{ij} = \frac{a_{ij}}{\sum_{k=1}^{m} a_{kj}} \tag{1}$$

Then, AHP computes weight vector w of criteria by computing the average value of each row of normalized matrix $\bar{A}$ through equation

$$w_i = \frac{\sum_{k=1}^{m} \bar{a}_{ik}}{m} \tag{2}$$

Finally, the *InitializeMigrationMatrices* algorithm (see Figure 4) is executed for each considered criterion c_k, where $k = 1, \ldots, m$, of criteria set $C = \{c_1, c_2, \ldots, c_m\}$. The multiple executions of the *InitializeMigrationMatrices* algorithm are used to create $n \times m$ matrix $V = [V^{(1)}, V^{(2)}, \ldots, V^{(m)}]$, where n is the number of possible migrations found before. In

Require: criterion k and its attributes *valueWithHighestWeight*$^{(k)}$ and *valueWithLowestWeight*$^{(k)}$; $\langle p_1, p_2, \ldots, p_n \rangle$ as values of services or service providers properties that consider k as their criterion

Ensure: $n \times n$ matrix $S^{(k)}$ as a migration comparison matrix based on criterion k

for $i \leftarrow 1$ **to** n **do**
  $s_{ii} \leftarrow 1$
  for $j \leftarrow i+1$ **to** n **do**
    $range \leftarrow valueWithHighestWeight^{(k)} - valueWithLowestWeight^{(k)}$
    $diffValue \leftarrow p_i - p_j$
    $fifthOfDiff \leftarrow range/5$
    if $diffValue > 4 * fifthOfDiff$ **then**
      $s_{ij} \leftarrow 9$
    else if $diffValue \leq 4 * fifthOfDiff \wedge diffValue > 3 * fifthOfDiff$ **then**
      $s_{ij} \leftarrow 7$
    else if $diffValue \leq 3 * fifthOfDiff \wedge diffValue > 2 * fifthOfDiff$ **then**
      $s_{ij} \leftarrow 5$
    else if $diffValue \leq 2 * fifthOfDiff \wedge diffValue > fifthOfDiff$ **then**
      $s_{ij} \leftarrow 3$
    else if $diffValue \leq fifthOfDiff$ **then** $s_{ij} \leftarrow 1$
    end if
    if $range * diffValue < 0$ **then**
      $s_{ij} \leftarrow s_{ij}^{-1}$
    else if $range * diffValue = 0$ **then**
      $s_{ij} \leftarrow 1$
    end if
    $s_{ji} \leftarrow s_{ij}^{-1}$
  end for
end for
return S

Fig. 4. The *InitializeMigrationMatrices* algorithm to compute a migration comparison matrix based on a given criterion.

the matrix V, each $V^{(k)}$ is a transpose of the weight vector of matrix $S^{(k)}$ obtained by an individual execution of the *InitializeMigrationMatrices* algorithm.

The result of the *InitializeMigrationMatrices* algorithm, which is matrix $S^{(k)}$, is an $n \times n$ matrix where each $s_{ij}^{(k)}$ entry represents a judgment value between the i^{th} migration and the j^{th} one based on the criterion k. The algorithm use *valueWithHighestWeight* and *valueWithLowestWeight* criterion attributes to map the judgment into one of values of set $\{1, 3, 5, 7, 9\}$ or their reciprocals, which are accepted by AHP.

By applying the *InitializeMigrationMatrices* algorithm for all m considered criteria we get m weight vectors of possible migrations, where each vector is related to one criterion. Finally, AHP computes the composite weight vector p of all possible n migrations through equation

$$p = V \cdot w \tag{3}$$

TABLE I
VALUES OF THE STATUS PROPERTIES PUBLISHED IN PARTIAL MODELS OF THE SERVICE PROVIDERS.

Provider	noPreference-Rules	Free-Memory	Permanent-StorageSize	Battery-LifeTime
XProvider	True	512 MB	512 MB	1 hour
YProvider	False	2048 MB	2048 MB	2 hours
ZProvider	True	2048 MB	2048 MB	3 hours

```
<hasProperty>
  <FreeMemory>
    <propertyValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2048</propertyValue>
    <propertyType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">INT</propertyType>
  </FreeMemory>
</hasProperty>
<hasProperty>
  <PermanentStorageSize>
    <propertyValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2048</propertyValue>
    <propertyType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">INT</propertyType>
  </PermanentStorageSize>
</hasProperty>
<hasProperty>
  <BatteryLifeTime>
    <propertyValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2</propertyValue>
    <propertyType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">INT</propertyType>
    <criteria
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">core:OriginBatteryLifeTimeCriteria</criteria>
    <criteria
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">core:DestinationBatteryLifeTimeCriteria</criteria>
  </BatteryLifeTime>
</hasProperty>
<rules>
  <rule>
    [YProviderPreference: (?service rdf:type core:MigratableService), (?origin rdf:type
    core:CandidateOriginServiceProvider), (?destination rdf:type core:CandidateDestinationServiceProvider),
    equal(?destination, core:YProvider), (?origin core:provides ?service), (?service core:hasProperty ?
    property),(?property rdf:type core:ServiceType), (?property core:propertyValue ?v1), eq(?v1,
    "major"^^http://www.w3.org/2001/XMLSchema#sting) -> (?destination core:possibleProvidedService ?
    service)]
  </rule>
</rules>
<noPreferenceRules rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">false</noPreferenceRules>
```

Fig. 5. The partial model for service provider *YProvider* with information on the provider's status properties and preference rules. Preference rule *YProviderPreference* written in the Jena rules language allows the provider to host only services with *ServiceType* set to value "major" (other providers *XProvider* and *ZProvider* do not have this restriction).

where V is the $n \times m$ matrix obtained by multiple executions of the *InitializeMigrationMatrices* algorithm as described before (one execution for each each criterion) and w is the weight vector of criteria from Equation 2.

The migration with the highest composite weight entry of the vector p is considered to be the best migration and it is selected to be executed as it is described in Section III-B.

V. AN EXAMPLE

In this section we provide an illustrative example to evaluate the proposed approach. The example presents a system of service providers hosting services which cooperate to achieve a particular business logic. Web service migration allows to migrate services from one provider to another in cases of unexpected violations of services or service providers preferences. The migration guarantees better availability of the services and increases fault-tolerance of the system. Let us suppose the system consists of the following three service providers: *XProvider*, *YProvider*, *ZProvider*. The status properties of these providers, which are published in their partial context model together with their preference rules (see Section III-A), are listed in Table I. For example, the information published by *YProvider* service provider, that is its status properties and preference rules, is listed in Figure 5.

For simplicity reasons, let us suppose that there are only two migratable services, namely service *Service1* currently provided/hosted by service provider *XProvider* and service *Service2* currently provided/hosted by service provider *YProvider*. A

TABLE II
VALUES OF THE STATUS PROPERTIES PUBLISHED IN PARTIAL MODELS OF THE SERVICES.

Service	noPreferenceRules	ServicePriority	ServiceType
Service1	False	50%	major
Service2	True	20%	minor

partial context model of each of these two services is available as a part of their WSDL description published by their service providers. The status properties published in the context models of these services are listed in Table II. Moreover, in the context model, *Service1* declares its two preferences described as Jena rules called *Service1Preference1* and *Service1Preference2* that, respectively, limit its migration to providers with *FreeMemory* ≥ 2048 MB and *PermanentStorageSize* ≥ 2048 MB only (in terms of the decision making process described in Section III-B, only providers with status property values meeting the limits above can be considered, in the ontology reasoning, as *possibleDestinationProvider*-s for service *Service1*).

According to the properties published by the individual service providers (see Table I and Figure 5), *OriginBatteryLifeTimeCriterion* and *DestinationBatteryLifeTimeCriterion* are considered as criteria to be included in the decision making process dealing with *BatteryLifeTime* properties of the providers. Similarly, the services publish their *ServicePriority* properties as related to criterion *ServicePriorityCriterion* to make it considered in the decision making process dealing with the properties.

When the service providers are discovered by module *Controller* utilizing WS4D-JMEDS stack (see Section III-B), status properties and preference rules of individual service providers are obtained by calling the *ProviderContextWS* grounding service of each provider. Moreover, for all migratable services hosted by the service providers, status properties and preference rules of the services are extracted from their WSDL descriptions. The resulting status properties and preference rules of both service providers and their migratable services are added into the ontology-base context model for future reasoning based on the information.

Later, when the systems starts to look for possible migrations, it detects two violations, i.e., two cases when the current values of status properties of service providers or services do not meet their preference rules. In the first case, *Service1Preference1* and *Service1Preference2* preference rules of service *Service1* are violated. These preference rules, which permit to host the service only by providers with *FreeMemory* ≥ 2048 MB and *PermanentStorageSize* ≥ 2048 MB, are violated by service provider *XProvider* which is currently hosting the service and has its both status properties *FreeMemory* and *PermanentStorageSize* set to value 512 MB (see Table I). In the second case, preference rule *YProviderPreference* of service provider *YProvider* is violated. This preference rule, which permits the provider to host only services with *ServiceType* set to value "major" (see Figure 5), is violated by service *Service2* which is currently hosted by the provider and has status property *ServiceType* set to value "minor" (see Table II).

TABLE III
THE MIGRATIONS FOUND TO FIX THE VIOLATED PREFERENCE RULE.

Migration	Service	Origin	Destination
mig1	Service1	XProvider	YProvider
mig2	Service1	XProvider	ZProvider
mig3	Service2	YProvider	XProvider
mig4	Service2	YProvider	ZProvider

TABLE IV
THE UTILIZED CRITERIA AND VALUES OF THEIR ATTRIBUTES.

	Criteria-Priority	owner	valueWith-Highest-Weight	valueWith-Lowest-Weight
ServicePriority-Criterion	7	service	100	0
OriginBattery-LifeTimeCriterion	9	origin	1	5
DestinationBattery-LifeTimeCriterion	3	destination	5	1

By ontology reasoning with Jena reasoners, four possible migration decisions are found to fix the violations above when performed in the system. These migrations, which are listed in Table III, are *mig1*, *mig2*, *mig3* and *mig4*. Migration *mig1* suggests to migrate *Service1* from *XProvider* to *YProvider*; migration *mig2* suggests to migrate *Service1* from *XProvider* to *ZProvider*, and so on (see Table III). First two migrations are addressing the first violation (of *Service1Preference1* and *Service1Preference2* preference rules of service *Service1*), while the other two migrations are addressing the second violation (of preference rule *YProviderPreference* of service provider *YProvider*).

The list of all possible migrations needs to be processed by AHP to find the best migration to be performed in the system. In order to use AHP, we define the following criteria in the context model: *ServicePriorityCriterion*, *OriginBatteryLifeTimeCriterion*, and *DestinationBatteryLifeTimeCriterion*. Values of individual attributes of the defined criteria are listed in Table IV. By application of the *InitializeCriteriaMatrix* algorithm (see Figure 3), criteria comparison matrix A is generated as shown in Table V. In the last column of Table V, there are also values of the weight vector for the criteria which is computed by application of Equation 2 on the normalized comparison matrix.

By application of the *InitializeMigrationMatrices* algorithm (see Figure 4) on the results above, migration comparison matrices $S^{(1)}$, $S^{(2)}$, and $S^{(3)}$ are generated for criteria *ServicePriorityCriterion*, *OriginBatteryLifeTimeCriterion*, and *DestinationBatteryLifeTimeCriterion*, respectively. These matrices are listed in Table VI together with their weight vectors. For $k \in \{1, 2, 3\}$, weight vector $v^{(k)}$ is computed for matrix $S^{(k)}$ by application of Equation 2 on corresponding normalized matrix $\bar{S}^{(k)}$.

TABLE V
COMPARISON MATRIX A GENERATED BY THE *InitializeCriteriaMatrix* ALGORITHM AND WEIGHT VECTOR w COMPUTED FROM THE MATRIX.

$\lambda_{max} = 3.0967$; $CR = 0.0833 < 0.1$; matrix A is consistent.				
	Service Priority Criterion	Origin Battery LifeTime Criterion	Destination Battery LifeTime Criterion	w_i
Service Priority Criterion	1.0	0.1428	0.2	0.0737
Origin Battery LifeTime Criterion	7.0	1.0	3.0	0.6433
Destination Battery LifeTime Criterion	5.0	0.3333	1.0	0.2828

TABLE VI
MIGRATION COMPARISON MATRICES $S^{(k)}$ GENERATED BY THE *InitializeMigrationMatrices* ALGORITHM AND WEIGHT VECTORS $v^{(k)}$ COMPUTED FROM THE MATRICES.

$\lambda_{max} = 4, CR = 0$; matrix $S^{(1)}$ is perfectly consistent.					
ServicePriority Criterion	mig1	mig2	mig3	mig4	$v^{(1)}$
mig1	1.0	1.0	3.0	3.0	0.375
mig2	1.0	1.0	3.0	3.0	0.375
mig3	0.3333	0.3333	1.0	1.0	0.1249
mig4	0.3333	0.3333	1.0	1.0	0.1249
$\lambda_{max} = 4, CR = 0$; matrix $S^{(2)}$ is perfectly consistent.					
OriginBattery LifeTimeCriterion	mig1	mig2	mig3	mig4	$v^{(1)}$
mig1	1.0	1.0	3.0	3.0	0.375
mig2	1.0	1.0	3.0	3.0	0.375
mig3	0.3333	0.3333	1.0	1.0	0.1249
mig4	0.3333	0.3333	1.0	1.0	0.1249
$\lambda_{max} = 4.0575, CR = 0.0213$; matrix $S^{(3)}$ is consistent.					
DestinationBattery LifeTimeCriterion	mig1	mig2	mig3	mig4	$v^{(3)}$
mig1	1.0	0.3333	3.0	0.3333	0.1534
mig2	3.0	1.0	5.0	1.0	0.3889
mig3	0.3333	0.2	1.0	0.2	0.0686
mig4	3.0	1.0	5.0	1.0	0.3889

Finally, after generating and computing all required matrices and vectors, AHP computes the composite weight vector p through Equation 3. The resulting vector is

$$p = \begin{pmatrix} 0.3586 \\ 0.376 \\ 0.1208 \\ 0.1444 \end{pmatrix} \quad (4)$$

where p_{11}, p_{21}, p_{31}, and p_{41} entries represent the weights of *mig1*, *mig2*, *mig3*, and *mig4*, respectively.

Finally, *mig2* is chosen to be performed as it has the highest priority ($p_{21} = 0.376$) and *Service1* will be migrated from service provider *XProvider* to service provider *ZProvider* which will fix the violated preference rules of *Service1*, namely *Service1Preference1* and *Service1Preference2*, and satisfy them with the current *FreeMemory* and *PermanentStorageSize* status property values of *ZProvider*.

Please note, that the resulting migration will fix just the most important violations of preference rules found before (not all of those violations), which is exactly the goal of our approach. For example, preference rule *YProviderPreference* of service provider *YProvider* is still violated by hosting service *Service2* (see Figure 5 and Table II) and should be fixed by future migrations. Informally said, migration *mig2* was selected by AHP because it deals with *Service1* which has greater *ServicePriority* than *Service2* in other migrations and because it has destination service provider *ZProvider* which has the best *BatteryLifeTime* in comparison with other possible destination providers (see Tables II and I, respectively). Both of this criteria, *ServicePriority* and *BatteryLifeTime* are listed in Table IV.

VI. SUMMARY

In this paper, we described an extension of Web service migration framework by the AHP method to select the best migrations from a list of all possible migrations previously found by ontology reasoning in a migration decision and selection process. The migration decisions are taken based on status properties and preference rules of service providers and their migratable services which were previously automatically discovered in a system. The framework has been evaluated on an example presented in the paper with focus on the AHP utilization.

In our ongoing work, we are studying possible applications of the framework in the state-of-the-art Web service platforms and trying to improve scalability and performance of the presented solution. Future work will mainly cover integration of advanced features into the framework from other existing approaches to software mobility, for example, from agent-oriented approaches or approaches dealing with mobile cloud computing. To be more specific, we would like to utilize the underlying implementation of the JMEDS framework for Android mobile devices to implement a general Web service provider for the Android devices that will be fully capable of receiving, hosting, and passing mobile native Web services participating in mobile cloud computing.

ACKNOWLEDGMENT

This work was supported by the research program MSM 0021630528 “Security-Oriented Research in Information Technology” and by the BUT FIT grant FIT-S-11-2.

REFERENCES

[1] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 342–361, May 1998.

[2] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, Aug. 2005.

[3] M. M. Kazzaz and M. Rychlý, "Ontology-based context modelling and reasoning in the Web service migration framework," *Acta Electrotechnica et Informatica*, vol. 13, no. 4, pp. 5–12, 2013.

[4] T. Saaty, *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. RWS, 1990.

[5] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 311–327, 2004.

[6] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang, "Live migration of multiple virtual machines with resource reservation in cloud computing environments," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 267–274.

[7] J. Famaey, T. Wauters, F. De Turck, B. Dhoedt, and P. Demeester, "Network-aware service placement and selection algorithms on large-scale overlay networks," *Computer Communications*, vol. 34, no. 15, pp. 1777–1787, 2011.

[8] W. Hao, I.-L. Yen, and B. Thuraisingham, "Dynamic service and data migration in the clouds," in *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*, vol. 2. IEEE, 2009, pp. 134–139.

[9] M. Sun, T. Zang, X. Xu, and R. Wang, "Consumer-centered cloud services selection using AHP," in *Service Sciences (ICSS), 2013 International Conference on*. IEEE, 2013, pp. 1–6.

[10] C. Reich, K. Bubendorfer, M. Banholzer, and R. Buyya, "A SLA-oriented management of containers for hosting stateful web services," in *e-Science and Grid Computing, IEEE International Conference on*. IEEE, 2007, pp. 85–92.

[11] T. Banks, "Web services resource framework WSRF-primer v1.2," *OASIS committee draft*, 2006.

[12] M. M. Kazzaz and M. Rychlý, "A Web service migration framework," in *ICIW'13, The Eighth International Conference on Internet and Web Applications and Services*. IARIA, Jun. 2013, pp. 58–62.