Order No:

Serial No:

People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

ECHAHID HAMMA LAKHDAR UNIVERSITY OF EL-OUED

FACULTY OF EXACT SCIENCES

COMPUTER SCIENCE DEPARTMENT

End of study thesis

ACADEMIC MASTER

Field: Mathematics and Computer Science

Sector: Computer Science

Speciality: Distributed Systems and Artificial Intelligence (SDIA)

Theme

Context-aware mobile Cloud service adaptation using semantic web and planning algorithms

Submitted by:

SOLTANA Tehani
ZENINA Zineb

Supervised by:

Dr. BEGGAS Mounir

Academic year: 2019/2020

Context-aware mobile Cloud service adaptation using semantic web and planning algorithms

Submitted by:

SOLTANA Tehani
ZENINA Zineb

Members of jury:

Mr. BEGGAS Mounir	Supervisor	Univ. of El Oued
Mr.LEJDEL Brahim	President	Univ. of El Oued
Mr.ZAIZ Faouzi	Examiner	Univ. of El Oued

Date of discussion:28 /10 /2020

شكر وتقدير

قال تعالى:(وَلَقَدْ آتَيْنَا لُقْمَانَ الحِكْمَةَ أَنِ اشْكُرْ للهِ وَمَنْ يَشْكُرْ فَإِنَّمَا يَشْكُرُ لِنَفْسِهِ)

وقال رسوله الكريم ﷺ :(من لم يشكر الناس لم يشكر الله عز و جل)

نحمد الله عز وجل حمدا كثيرا طيبا مباركا ملئ السموات والارض على ما أكرمنا به لاتمام هذه المذكرة التي نرجو أن تنال رضاه.

ثم نتوجه بعظيم الشكر وجزيل الامتنان إلى كل من:

*الدكتور الفاضل:بقاص منير لاشرافه على هذا العمل ونصحه لنا لاتمام هذه الدراسة.

*اعضاء لجنة المناقشة الكرام لقبولهم مناقشة هذه الدراسة.

* الى جميع الزملاء والاصدقاء الذين وقفوا معنا لإتمام هذه المذكرة.

كما نتقدم بشكر خاص إلى الصديقة العزيزة صحراوي لطيفة.

إهداء

نهدي هذا العمل المتواضع إلى:

الوالدين الكريمين حفظمهما الله وأطال في عمرهما.

وإلى كل أفراد أسرتينا.

وإلى أرواح أجدادنا رحمهم الله وأسكنهم فسيح جناته.

وإلى جميع الأصدقاء والأحباء الذين كانو معنا وبرفقتنا في كل مراحل دراستنا الجامعية.

وإلى كل من لم يدخر جهدا في مساعدتنا.

وإلى كل من ساهم في تلقيننا ولو بحرف في حياتنا الدراسية.

الطالبتان:

سلطانة تهاني/زنينة زينب

ABSTRACT

With the large use of mobile devices, mobile applications are spread as a convenient alternative to classical computer applications. Due to the nature of mobile application and resource limitations of mobile devices, cloud services is an ideal alternative to software installed on these devices. Mobile environment is a dynamic environment in which the user context is changed dynamically. Cloud service adaptation is required to deal with this changing. We adopt and extend the proposed model of a cloud service adaptation approach that adapts cloud service to fit mobile context. We extend also the proposed semantic web API mashup planning algorithm based on goal/postcondition validation and input/output matching. For web API description, we used the augmentation of OpenAPI description by JSON-LD fields in order to annotate inputs/outputs by RDF and SPARQL statements to describe web API preconditions/postconditions. The adaptation is performed by mashing up automatically adaptation services based on their semantic description. We validate the proposition by the implementation of a prototype to demonstrate the applicability of our proposal.

Key words:

Mobile Cloud Service , Adaptation, Rest API Mashup, Context

RÉSUMÉ

Avec la large utilisation des appareils mobiles, les applications mobiles se sont répandues comme une alternative pratique aux applications informatiques classiques. En raison de la nature des applications mobiles et des ressources limitées des appareils mobiles, les services cloud sont une alternative idéale aux logiciels installés sur ces appareils. L'environnement mobile est un environnement dynamique dans lequel le contexte utilisateur est modifié de manière dynamique. En effet, l'adaptation du service cloud est nécessaire pour faire face à ce changement. Dans lequel nous adoptons et étendons le modèle proposé d'une approche d'adaptation de service cloud qui adapte le service cloud au contexte mobile. Nous étendons également l'algorithme de planification de mashup d'API Web sémantique proposé basé sur la validation goal / postcondition et la correspondance entrée / sortie. Pour la description de l'API Web, nous avons utilisé l'augmentation de la description OpenAPI par des champs JSON-LD afin d'annoter les entrées / sorties par des annottaions RDF et SPARQL pour décrire les pré-conditions / post-conditions de l'API Web. L'adaptation est effectuée par le mashup automatique des services d'adaptation en fonction de leur description sémantique. Nous validons la proposition par la mise en œuvre d'un prototype pour démontrer l'applicabilité de notre proposition.

Les mots clés:

Service Cloud Mobile, Adaptation ,Rest API Mashup, Contexte.

الملخص

مع الاستخدام الكبير للأجهزة المحمولة ، تنتشر تطبيقات الهاتف المحمول كبديل مناسب لتطبيقات الكمبيوتر الكلاسيكية. نظرًا لطبيعة تطبيقات الهاتف المحمول ومحدودية الموارد للأجهزة المحمولة ، تعد الخدمات السحابية بديلاً مثاليًا للبرامج المثبتة على هذه الأجهزة. بيئة الهاتف المحمول هي بيئة ديناميكية يتم فيها تغيير سياق المستخدم ديناميكيًا. مطلوب تكييف الخدمة السحابية للتعامل مع هذا التغيير .سنعتمد ونوسع النموذج المقترح لنهج تكيف الخدمة السحابية الذي يكيف الخدمة السحابية لتلائم سياق الهاتف المحمول . كما سنقوم أيضًا بتوسيع خوارزمية تخطيط مزيج واجهة برمجة تطبيقات الويب الدلالية المقترحة استنادًا إلى التحقق من صحة الهدف / ما بعد الحالة ومطابقة المدخلات / المخرجات . بالنسبة لوصف واجهة برمجة تطبيقات الويب ، استخدمنا زيادة وصف OpenAPI بواسطة حقول JSON-LD من أجل التعليق على المدخلات / المخرجات بواسطة عبارات RDF و SPARQL لوصف(preconditions/postconditions) لواجه برمجة تطبيقات الويب .يتم إجراء التكيف عن طريق مزج خدمات التكيف تلقائيًا بناءً على الوصف الدلالي .نحن نتحقق من صحة الاقتراح من خلال تنفيذ نموذج أولي لإثبات قابلية تطبيق اقتراحنا.

الكلماتالمفتاحية :

خدمة السحابة المتنقلة ، التكيف، واجهة برمجة التطبيقات ، السياق.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GENERAL INTRODUCTION

Throughout the last decade, mobile devices became constant companions allowing to access information and services anytime and anywhere. Due to the ongoing miniaturization and cost-reduction of constituents, these devices are nowadays powerful enough to support lots of our everyday routines. The mentioned situation has led to increasing demand for system support that can exploit the potentials of spontaneous interaction and therefore needs to be able to dynamically adapt to the quickly and constantly changing context of mobile ad-hoc scenarios. At present, many of the proposed solutions provide only limited context-awareness or just specific prediction capabilities that moreover often require configuration by the developers. Consequently, MCC tries to weaken the restrictions of mobile applications by offering centralized resources to augment mobile devices. According to [36], it is defined as the integration of cloud computing into the mobile environment to overcome obstacles related to performance (e.g., battery life, storage, and bandwidth), environment (e.g., heterogeneity, scalability, and availability), and security (e.g., reliability and privacy). We agree with this definition, extending the consideration of environmental restrictions to the more general problem of context adaptation.

Web API mashup refers to the process of combining different Web APIs to offer new composite value-added API. A mashup often takes the form of a Web resource that combines information and services from multiple sources on the Web, offering user-oriented value-added services . From the point of view of their purpose, mashups can broadly be classified into the following categories:

• Data-oriented mashups involve converting, transforming or combining data elements provided by resources into value-added outputs.

• Process-oriented mashups imply a step forward from data-oriented mashups, allowing the composition of business services and the integration of these latter within existing business processes.

Context-aware cloud service adaptation consists of transforming cloud service output into a new one that well fits current context situation. Therefore, we consider the adaptation process as data oriented API mashup process.

In this work, we adopt and extend the work of [37] to propose a cloud service adaptation process that deduces, according to mobile context, the relevant API mashup plan. The plan is

defined by an orchestration of data-driven APIs to transform primary output to context/relevant one. Automatic adaptation process is carried out by a gateway in which each cloud service is represented by a proxy API. When mobile application calls cloud mobile service API proxy, the gateway triggers adaptation process after extracting mobile context parameters from request headers and use them to get expected output profile (set of parameters that describe the well adapted output of the cloud service in current mobile context situation). The adaptation is performed by automatic data-driven mash up process that orchestrates adaptation APIs so as to transform cloud service output to corresponding one to expect output profile.

Chapter 01:Rest API Description and Mashup

1) Introduction

As the growth of service computing in the modern services industry, web services have evolved to web-based “cloud services”, which consist of not only classical SOAP/WSDL web services but also RESTful services, data, open-source codes, and any programmable application components accessible through the Internet.

A RESTful API is an architectural style for an application program interface (API) that uses HTTP requests to access and use data. That data can be used to GET, PUT, POST and DELETE data types.

An API Mashup is an API that coordinates countless APIs. This process takes place by categorizing and revealing them as one API for the consumer. It also allows you to generate many methods from different API Gateways. This process also helps to expose the performance as a single and distinct API.

In this chapter, we describe REST API, REST API mashup, and its syntactic and semantic description. Finally we presentAutomatic Rest API Mashup.

2) RESTAPI

To learn more about REST API we will define and explain API , Rest and Rest API.

2.1) API

API (Application Programming Interface) is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an API [1].

2.2) Rest

REST (Representational State Transfer) is a hybrid style that aims to induce certain architectural properties that are important to distributed hypermedia systems. These properties include usability, simplicity, scalability and extensibility. By introducing these properties with carefully considered tradeoffs,REST attempts to minimize latency and network communications, and at the same time, maximizing the independence and scalability of component implementations, including user agent, proxy, gateway, origin server and various connectors, in distributed hypermedia systems[2].

2.3) Rest API

REST or RESTful API design (Representational State Transfer) is designed to take advantage of existing protocols. While REST can be used over nearly any protocol, it usually takes advantage of HTTP when used for Web APIs. This means that developers do not need to install libraries or additional software in order to take advantage of a REST API design. REST API Design was defined by Dr. Roy Fielding in his 2000 doctorate dissertation. It is notable for its incredible layer of flexibility. Since data is not tied to methods and resources, REST has the ability to handle multiple types of calls, return different data formats and even change structurally with the correct implementation of hypermedia.[3]

REST APIs are modeled as a set of interfaces implemented by resources. Although this design supports the REST uniform interface constraint, it inevitably creates fixed resource names, types and hierarchies that violate the REST API design rules prescribed by Roy Fielding[4] . This kind of violations leads to an API that depends on the out-of-band information, instead of hypermedia, to drive the interactions between components.

3) Syntactic REST API description

Web services enable the publishing and consuming of functionalities of existing applications, facilitating the development of systems based on decoupled and distributed components.

3.1) OpenAPI Description

OpenAPI ex Swagger is an API description format that is readable both by humans and machines. The description file (aka the spec file) is written in JSON or YAML and contains all the details of REST API for every available type of request and response. This means that it is possible to automatically generate documentation, as well as tools, server/client code, tests to drive the requests themselves[5].

An OpenAPI description file allows to describe the entire API, including:

- Available endpoints (example: /users) and operations on each endpoint (example: GET/users, POST/users)
- Operation parameters Input and output for each operation
- Authentication methods
- Contact information, license, terms of use and other information.[6]

- OpenAPI Specification

a) OpenAPI Format

An OpenAPI document that conforms to the OpenAPI Specification is itself a JSON object, which may be represented either in JSON or YAML format. While APIs may be defined by OpenAPI documents in either YAML or JSON format, the API request and response bodies and other content are not required to be JSON or YAML.

b) OpenAPI Document

A document (or set of documents) that defines or describes an API. An OpenAPI definition uses and conforms to the OpenAPI Specification.

c) Document Structure

An OpenAPI document may be made up of a single document or be divided into multiple, connected parts at the discretion of the user. In the latter case, $ref fields must be used in the specification to reference those parts as follows from the JSON Schema definitions.

d) OpenAPI Data Types

Primitive data types in the OpenAPI, are based on the types supported by the JSON Schema Specification Wright Draft 00. The formats defined by the OpenAPI are :

type	format	Comments
integer	int32	signed 32 bits
integer	int64	signed 64 bits (a.k.a long)
number	float	
number	double	
string		
string	byte	base64 encoded characters
string	binary	any sequence of octets
boolean		
string	date	As defined by full-date - RFC3339
string	date-time	As defined by date-time - RFC3339
string	password	A hint to UIs to obscure input.

Table 1: OpenAPI Data Types formats.

e) OpenAPI examples

We can write OpenAPI definitions in YAML or JSON. In this example, we use only YAML examples but JSON works in the same manner This exemple is written in YAML according to OpenAPI 3.0 specification.

```yaml
openapi: 3.0.0
info:
  title: Sample API
  description: Optional multiline or single-line description in [CommonMark](http://commonma
  version: 0.1.9

servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server for testing

paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        '200':    # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

Figure 1: OpenAPI examples

1. Metadata

Every API definition must include the version of the OpenAPI Specification that this definition is based on

```yaml
openapi: 3.0.0
```

Figure 2: Metadata.

The info section contains API information: title, description (optional), version:

```yaml
info:
  title: Sample API
  description: Optional multiline or single-line description in [CommonMark](http://commonma
  version: 0.1.9
```

Figure 3: info section.

2. Servers

The servers section specifies the API server and base URL.Define one or several servers, such as production and sandbox.

```yaml
servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server for testing
```

Figure 4: servers section.

3. Paths

The paths section defines individual endpoints (paths) in the API, and the HTTP methods (operations) supported by these endpoints. For example, GET /users can be described as [7]:

```yaml
paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML
      responses:
        '200':
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

Figure 5: paths section.

4) Semantic REST API description

Semantic RESTful APIs combine the power of the REST architectural style, the Semantic Web and Linked Data.

4.1) What is Semantic REST API description?

REST APIs take programmatic inputs and produce programmatic outputs [8]. Although programmatic inputs/outputs provide an easy and intuitive way of using APIs, their

limitations must also be considered. As input/output parameters are freely and arbitrarily chosen instead of relying on a controlled vocabulary, parameter ambiguity will likely cause a mismatch between APIs. Another problem is that parameters only represent a flat structure with no hierarchy, thus the large number of parameters might cause difficulties for developers in matching compatible APIs [9].

4.2) Methods of Semantic REST API description

4.2.1) Ontology Learning Method

REST APIs accept programmatic inputs and produce programmatic outputs [10]. Although programmatic inputs/outputs provide an easy and intuitive method of using APIs, their limitations must also be considered. As input/output parameters are freely and arbitrarily chosen, instead of relying on a controlled vocabulary, parameter ambiguity will likely cause a mismatch between APIs. Furthermore, parameters only represent a flat structure with no hierarchy, thus a large number of parameters could cause difficulties for developers in matching compatible APIs. The solution to automatic Web API composition presented here is based on the ontology learning method, whose principles and techniques have been presented in [11]. this section provide a presentation of the method.

a. Parameter Hierarchical Clustering

Developed a hierarchical clustering technique to derive several semantically meaningful concepts from the API parameters. Consider the syntactic information that resides in the API descriptions and apply a mining algorithm to obtain their underlying semantics. The main idea is to measure the co-occurrence of terms and then cluster the terms into a set of concepts. The input/output parameters are often combined as a sequence of several terms .For example ‹in the parameter ArrivalTimeOfAirplan , the terms are specified by their first letter capitalized Arrival ‹Time ‹Of ‹Airplane .When clustering terms residing in the parameters into several meaningful semantic concepts , they consider the cnoccurrence of terms .A common heuristic is that the terms tend to express the same concept if they frequently occur together .This allows us to cluster the terms by exploiting the conditional probability of their occurrence in the inputs and outputs of the APIs.

b. Parameter Pattern Analysis

The parameter pattern analysis technique captures relationships between the terms contained in a parameter and matches the parameters if both terms are similar and the relationships are equivalent. This approach is derived from the observation that peopleemploy similar patterns when composing a parameter out of multiple terms. The distribution of Web APIs in Programmable. This ontology is generated from the set of parameters. it's able to match a query and the ontology. Two ontological concepts are matched if and only if one of the following is true:

a. one concept is a property of the other concept (i.e., Parameter propertyOf Noun),

b. one concept is a subclass of the other concept (i.e., Parameter subClassOf Noun).

An agent can be able to find a match based on the similarities of the API. For example, assume that a parameter CityName was to be compared against another parameter CodeOfCity. The keyword search would not count these as a possible match. However, if the City term had the relationship “X propertyOf Y” in its pattern rule, the matching logic will return a matching score because these two parameters are closely related [12] .

c. Input/Output Semantic Matching

The semantic matching technique estimates the similarity of the input and output by considering the underlying concepts of the input/output parameters. Formally, the input is presented as a vector $I = < pi , Ci >$ (similarly, the output can be represented in the form $O = < po , Co >$), where *pi* is the set of input parameters and *Ci* is the set of concepts that are associated with pi . Then, the similarity of the input can be found using the following two steps (the output can be processed in a similar fashion):

- divide *pi* into a set of terms and then find synonyms for these term,
- replace each term with its corresponding concepts and then compute a similarity score[13].

The similarity score is defined to select the best matches for the given input. Consider a pair of candidate parameters A and B, the similarity between A and B is given by the following formula :

$$Sim(A,B) = \frac{2x|| Match(A,B)||}{n1+ n2}$$

where n1 and n2 denote the number of valid terms in the parameters and ‖ Match (A,B)‖ returns the number of matching terms. The similarity of each parameter is calculated by the best matching parameter having a larger number of semantically related terms. The overall similarity is computed by a linear combination to combine the similarity of each parameter. Because matching techniques based on clustering consider all the terms in a cluster as an equivalent concept and ignore any hierarchical relationships between the terms, matches might exist that are irrelevant to the user's intention (i.e., false positives) [14].

5) REST API Mashup

To learn more about REST API Mashup we will define and explain it and we will mention the Structure of an API Mashup,Automatic Rest API Mashup and Composition Algorithm

5.1) REST API Mashup

An API Mashup is an API that coordinates countless APIs. This process takes place by categorizing and revealing them as one API for the consumer. It also allows you to generate many methods from different API Gateways. This process also helps servers that provide an API may expose a vast set of functionality. However, each individual service in the API usually provides a very specific functionality. While this is usually effective, sometimes it is useful or required to consolidate a few services and expose them as a single service. In other situations, you might want to extend a service with the functionality provided by an external API. API mashups address these requirements for grouping services and exposing them as a single service. The APIs that are included in an API mashup (participating APIs) can be connected to each other in the following ways:

- API chaining: Two or more participating APIs are connected and invoked in a sequence one after the other.
- API aggregation: Two or more participating APIs are connected to a common aggregator step.

The aggregator step captures the response of the aggregated APIs. The aggregator step enables you to:

- Collate the responses and pass to the next step.
- Process the responses and pass the processed data to the next step [15].

5.2) Structure of an API Mashup

An API mashup consists of one or more mashup steps, and each step invokes an API. A mashup step defines the request for the API that it invokes. A step can use the data objects provided by API Gateway to access data in the initial request sent to the operation that has the mashup and any of the previous steps.

The following table summarizes the data objects and variables that are available in API Gateway [16] .

Object/Variable Type	Possible values
- parama Stage	- reque - response
paramType	- payload or body - headers - query - path -httpMethod - statusCode - statusMessage
queryType	- xpath - jsonPath - regex

Table 2: the data objects and variables that are available in API Gateway.

5.3) Automatic Rest API Mashup

Developing a data mashup is a process composed of the following phases:

- suitable APIs must be selected from a registry
- IO mappings and pairing among different APIs must be established
- programming code to actually combine the selected APIs must be written to complete the final application.

5.4) Composition Algorithm

Given a query and a collection of APIs, in the case that a matching API is not found, searching a sequence of APIs that can be combined is the composition problem of Web APIs. This means that the output generated by one API can be accepted as the input of another API. For example, we are looking for APIs to find a hotel's location. Fig 6 shows the input/output parameters of a query Q, and two Web APIs W1 and W2 in the registry. Assume that the agent cannot find a single API that matches the criteria. Then, it composes n APIs from the set of Web APIs available in the registry. In this figure, W returns HotelName as the output. W2 receives this as the input and returns Location as the result. Therefore, the subsequent W2 may use the output produced by the preceding W1 as the input.

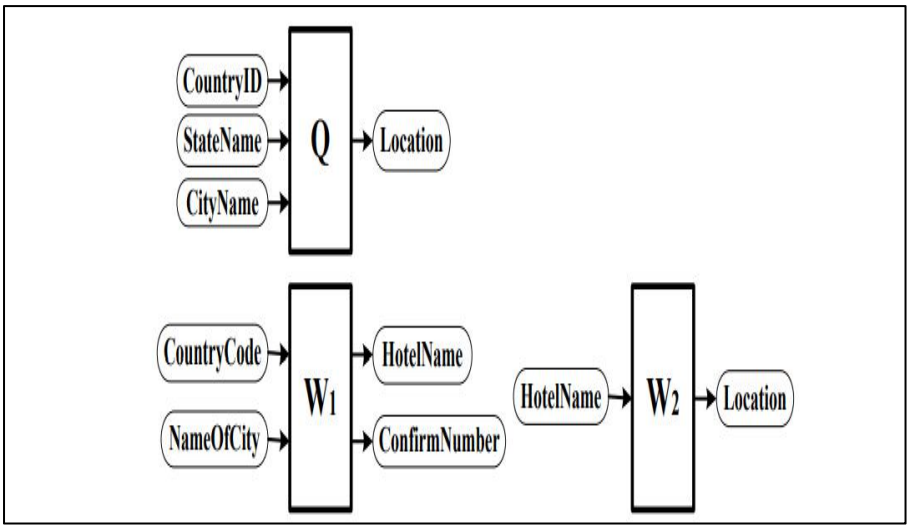

Figure 6: Example of Composition Problem .

a. Construction of Composable Similarity Graph

To accelerate the calculation of possible composition plans, use a precomputed CSG that chains the output of one API into the input of another API. This graph, G=(V, E), is based on a set of nodes, where V is the vertex set and E is the edge set. The connection of the nodes is based on the semantic similarity between the output and input of the nodes. Algorithm 1 illustrates the construction procedure for the graph. First, assign each API W in the registry to vertexes iteratively. then establish edges between the vertexes. For each vertex vi,determine if its corresponding output can be accepted as an input by vj by computing the similarity score. If the output of vi is semantically similar to the input of vj (i.e.;*Semantic Match* (vi, l,vi,O) > 0), then we add a directed edge from vi to vj (in the reverse direction) and assign a similarity score. also verify if there exists a vertex vj, whose output can be consumed by vi as an input, in the similar manner. After constructing the CSG, solve the composition problem within this graph. The initial graph is dynamically modified if new APIs become available [17].

b. Algorithm: CSG Construction Algorithm

For each W in Registry

vi = addVertex(W)

Endfor

For each vi ϵ V

For each vi ϵ V

If Semantic Match (vi, l,vi,O) > 0) then addEdge(vi,vj)

If Semantic Match (vi, l,vj,O) > 0) then addEdge(vj,vi)

EndFor

EndFor

---

6) Conclusion

This chapter presents a definition of Rest API and REST API mashup, and it presents syntactic, semantic REST API description, and algorithms for automatic Web API discovery and composition. REST API Mashup provides a better user experience. Since it comes together as one API for the user.

Chapter 02: Mobile Cloud Service

1) Inroduction

Cloud computing is a technology that facilitates the delivery of services by providing hardware and software in data centers over the Internet. With the increased use of mobile phone lead to the need of “Information anywhere, anytime”. But, due to inadequacy of computing and storage resources on mobile phones as compared to PCs and laptops, cloud computing brings opportunities for mobile phones

Mobile Cloud Computing (MCC) is the combination of cloud computing and mobile computing to bring rich computational resources to mobile users, network operators, as well as cloud computing providers. In this chapter we will describe mobile cloud computing and adaptation to mobile context of mobile cloud service.

2) Mobile Cloud Service

Mobile cloud computing refers to an infrastructure where both the data storage and data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones to the cloud, bringing applications and MC to not just smartphone users but to a much broader range of mobile subscribers [18].

Mobile Cloud Computing (MCC) allows to develop mobile cloud infrastructures, platforms, and service applications for mobile clients. Its goal is to deliver them with secure mobile cloud resources, service applications, and data using energy-efficient mobile cloud resources in a pay-as-you-use model. MCC is known to be a promising solution for mobile computing because it overcomes obstacles in mobile computing, for example, improving data storage capacity and processing power by enabling mobile users to store/access data on the cloud.

New mobile data service technologies and solutions are required to cope with the new demands on mobile data services in mobile cloud computing. There are some required distinct service features and requirements for mobile cloud data service technologies in the mobile cloud computing paradigm. Here we list the primary ones:

- Offer on-demand mobile data services on a mobile cloud based on service-level agreements.
- Pay-as-you-use business model.

- Offloading computing power demands from mobile devices to clouds.
- Multi-tenanted mobile data service.
- Highly elastic and scalable data services.
- Highly reliable and available mobile data service.

3) Architectures of Mobile Cloud Computing

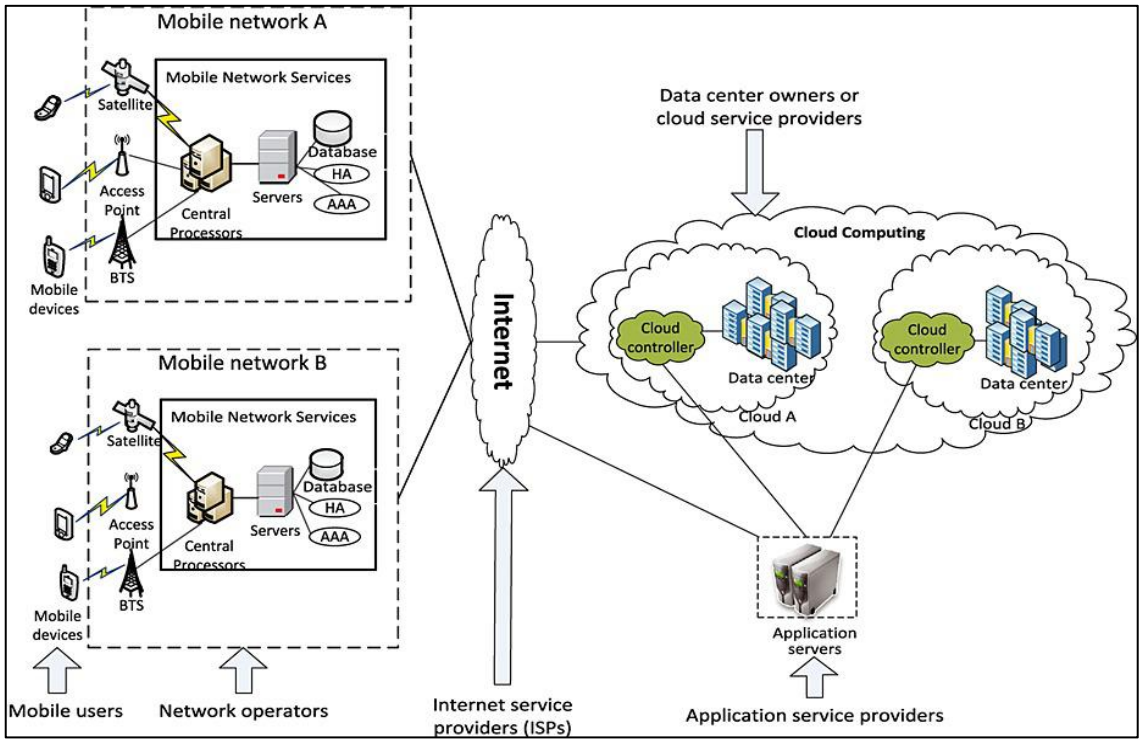

Figure 8: Mobile cloud computing architecture.

From the concept of MCC, the general architecture of MCC can be shown in Figure 8. Mobile devices are connected to the mobile networks via base stations (e.g., base transceiver station, access point, or satellite) that establish and control the connections (air links) and functional interfaces between the networks and mobile devices. Mobile users' requests and information (e.g., ID and location) are transmitted to the central processors that are connected to servers providing mobile network services. Here, mobile network operators can provide services to mobile users as authentication, authorization, and accounting based on the home agent and subscribers' data stored in databases. After that, the subscribers' requests are delivered to a cloud through the Internet. In the cloud, cloud controllers process the requests to provide mobile users with the corresponding cloud services. These services are developed

with the concepts of utility computing, virtualization, and service-oriented architecture (e.g., web, application, and database servers).

Generally, a CC is a large-scale distributed network system implemented based on a number of servers in data centers. The cloud services are generally classified based on a layer concept (Figure9). In the upper layers of this paradigm, Infrastructure as a Service (**IaaS**), Platform as a Service (**PaaS**), and Software as a Service (**SaaS**) are stacked.

a) **Data centers layer :** This layer provides the hardware facility and infrastructure for clouds. In data center layer, a number of servers are linked with high-speed networks to provide services for customers. Typically, data centers are built in less populated places, with a high power supply stability and a low risk of disaster.

b) **IaaS :** Infrastructure as a Service is built on top of the data center layer. IaaS enables the provision of storage, hardware, servers, and networking components. The client typically pays on a per-use basis. Thus, clients can save cost as the payment is only based on how much resource they really use. Infrastructure can be expanded or shrunk dynamically as needed. The examples of IaaS are Amazon Elastic Cloud Computing and Simple Storage Service (S3).

c) **PaaS :** Platform as a Service offers an advanced integrated environment for building, testing, and deploying custom applications. The examples of PaaS are Google App Engine, Microsoft Azure, and Amazon Map Reduce/Simple Storage Service.

d) **SaaS :** Software as a Service supports a software distribution with specific requirements. In this layer, the users can access an application and information remotely via the Internet and pay only for that they use. Salesforce is one of the pioneers in providing this service model. Microsoft's Live Mesh also allows sharing files and folders across multiple devices simultaneously.

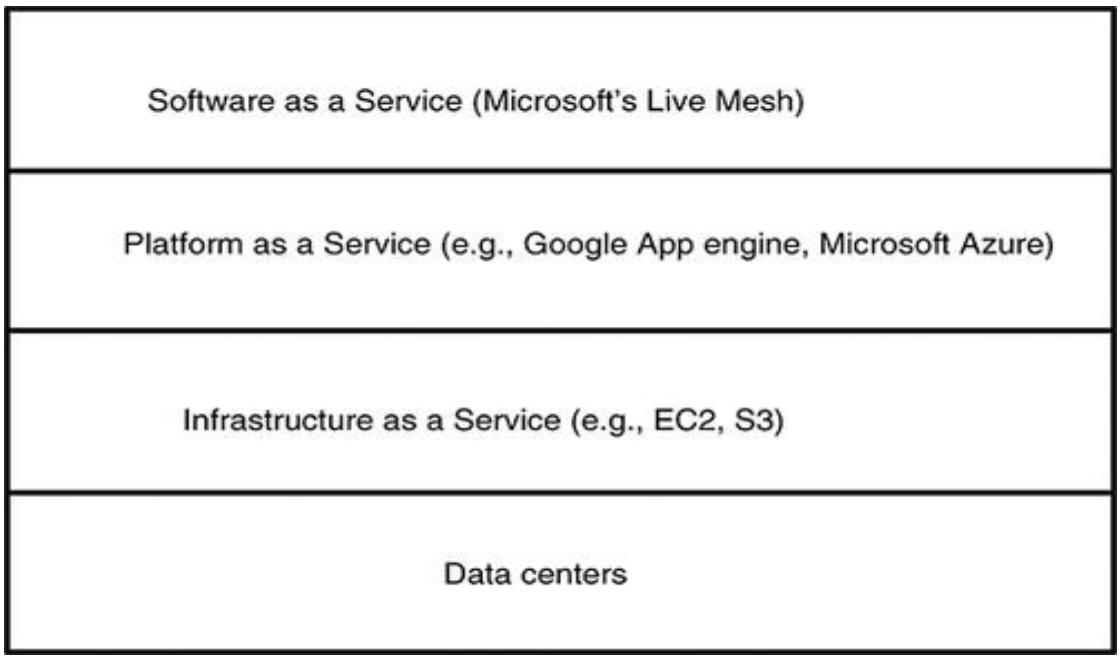

Figure 9: Service-oriented cloud computing architecture.

4) Application of Mobile Cloud Computing

Mobile applications gain increasing share in a global mobile market. Various mobile applications have taken the advantages of MCC. In this section, some typical MCC applications are introduced.

a) Mobile commerce :

Mobile commerce (m-commerce) is a business model for commerce using mobile devices. The m-commerce applications generally fulfill some tasks that require mobility (e.g., mobile transactions and payments, mobile messaging, and mobile ticketing). The m-commerce applications can be classified into few classes including finance, advertising, and shopping.

b) Mobile learning :

Mobile learning (m-learning) is designed based on electronic learning (e-learning) and mobility. However, traditional m-learning applications have limitations in terms of high cost of devices and network, low network transmission rate, and limited educational resources. Cloud-based m-learning applications are introduced to solve these limitations.

For example, utilizing a cloud with the large storage capacity and powerful processing ability, the applications provide learners with much richer services in terms of data (information) size, faster processing speed, and longer battery life

c) Mobile healthcare :

The purpose of applying MCC in medical applications is to minimize the limitations of traditional medical treatment (e.g., small physical storage, security and privacy, and medical errors . Mobile healthcare (m-healthcare) provides mobile users with convenient helps to access resources (e.g., patient health records) easily and efficiently. Besides, m-healthcare offers hospitals and healthcare organizations a variety of on-demand services on clouds rather than owning standalone applications on local servers.

d) Mobile gaming :

Mobile game (m-game) is a potential market generating revenues for service providers. M-game can completely offload game engine requiring large computing resource (e.g., graphic rendering) to the server in the cloud, and gamers only interact with the screen interface on their devices.

e) Other practical applications :

A cloud becomes a useful tool to help mobile users share photos and video clips efficiently and tag their friends in popular social networks as Twitter and Facebook. MeLog 51 is an MCC application that enables mobile users to share real-time experience (e.g., travel, shopping, and event) over clouds through an automatic blogging. The mobile users (e.g., travelers) are supported by several cloud services such as guiding their trip, showing maps, recording itinerary, and storing images and video [19] .

5) Adaptation of the Mobile Cloud service to mobile context

According to Baldauf et al [20] : ”Context-aware systems are able to adapt their operations to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking environmental context into account.”. To do so, they need to be able to monitor information about their internal and external state. But as they often not only just monitor but react to a changing state, many context-aware applications are

as well self-adaptive. Although there exist several classifications for context-data, a common one is the differentiation between environmental-, user- and resource context. Environmental context subsumes all perceived characteristics of the physical environment like temperature, air quality, etc. The user context can be the user's current destination and the device context may contain information about the current battery level or the amount of available memory. Using this information, context-aware systems are for example able to format the information to be presented to the user according to the hardware and software features of the device (e.g., request a low-res video instead of a high-res one)[21].

5.1) Context Parameters and Context Awareness in Mobile Cloud Computing

Collection Campaign (LDCC), an effort to collect sensor data from smartphones created by almost 200 volunteers in the Lake Geneva region over 18 months .It is the largest dataset that contains information about mobile devices as well as application usage statistics [22]:

- General information about the mobile device itself: time since the last user interaction, silent mode switch status, charging state, battery level, free memory.
- Date, time, location, calendar events, average and predicted remaining duration of stay at the current location.
- Reasoned attributes: remaining duration of stay at the same WiFi access point or GSM cell, user is at home/work, traveling, moving, resting.
- Application usage data: The applications the user interacts with and the specific screens of these applications [22].

5.2) Computing Model for Context-aware Cloud Computing

In CC, all the cloud services are located on the provider's side, and consumers look up and invoke the services. Since consumers' mobile devices can sense and monitor their context, the context information can help to invoke cloud services. In context-aware service provisioning, context specific behavior should be added [23], as shown in Figure10.

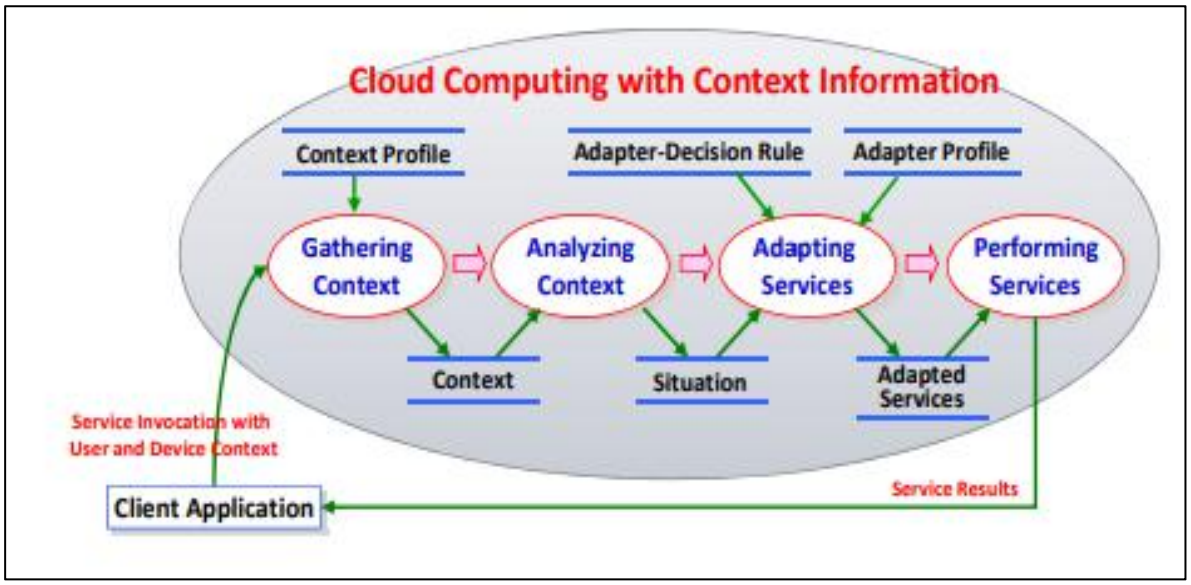

Figure 10: Computing Model of Context-Aware Cloud Services.

Main difference between conventional and context aware methods lies in the process performed by service providers after services gets invoked. Without context information, a searched service is just bound to the consumer. On the other hand, in CC with context information, the context information is accompanied with the service invocation, and analyzed to determine the current situation of the consumer. Then, a most appropriate service is selected at the stage of adapting services, and the service is invoked. While a service is running, its associated context can change and the service needs to be adapted dynamically.

In summary, the context information provides a key clue to adapt service dynamically in terms of service personalization and fault remedy. That is, there are two cases of utilizing context information :

- First, context is used to personalize a service invoked by a service consumer.
- Secondly, context is used to dynamically remedy low QoS problems such as faults at service execution time.

5.3) Types of Context-based Service Adapter

According to changes on the context information, services may need to be adapted to tailor them to a consumer or to remedy service faults. Therefore, we derive several types of context-based adapters in this section by analyzing types of gaps, types of causes to the gap, and types of adapters for the given cause.

We consider multi-layered relationships among context related elements as shown in Figure 11. The figure consists of four layers in deriving context-aware service adapters; Contexts, Gaps, Causes to gaps, and Adapters

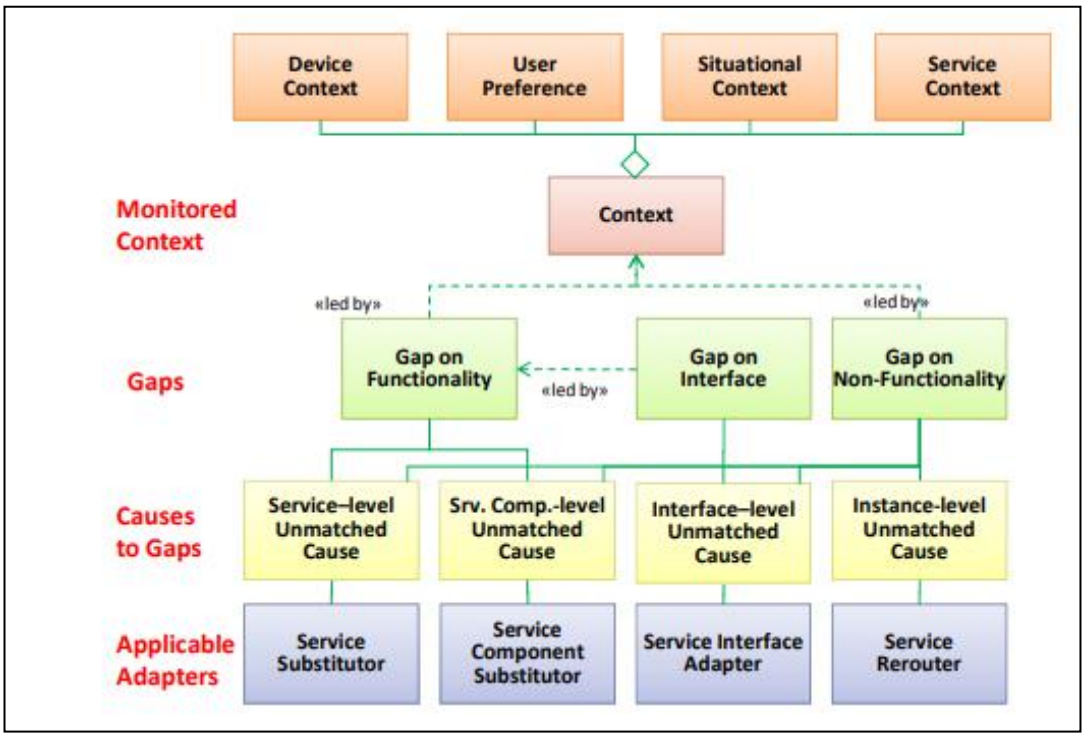

Figure 11: Context-Concerned Hierarchy.

The top layer of the context-concerned hierarchy is the layer of Monitored Context, which represents the currently monitored context. There are different meta-models of context in services which are presented in representative works [23][24][25][26]. there is four elements of contexts in Figure 11 ; Device Context, User Preference, Situational Context and Service Context.

a. Device Context is the environmental settings and configurations of the user's device.

b. User Preference specifies user-specific preference settings, especially those regarding to services selection and invocation.

c. Situational Context is a set of monitored data and information regarding the user's location, time, and other current settings related to the user.

d. Service Context captures the current status of a service provided such as monitored values of QoS attributes.

In context-aware service provisioning, target services should be tailored for the given context, and there often exists a gap between an available service and the service needed for the context. The second layer in the figure is for Types of Gaps, which specifies three different types of gaps that can occur. To derive types of gaps resulting from context changes, we need to know targets where the gaps occur. As shown in Figure 12, as a user moves with his own MID, the user's context can be also changed from Context1 to Context2. Along with the context change, a service to be bound can be also changed since a former service does not fully match to a new context. Hence, there exist some gaps between Service1 and Service2[27].

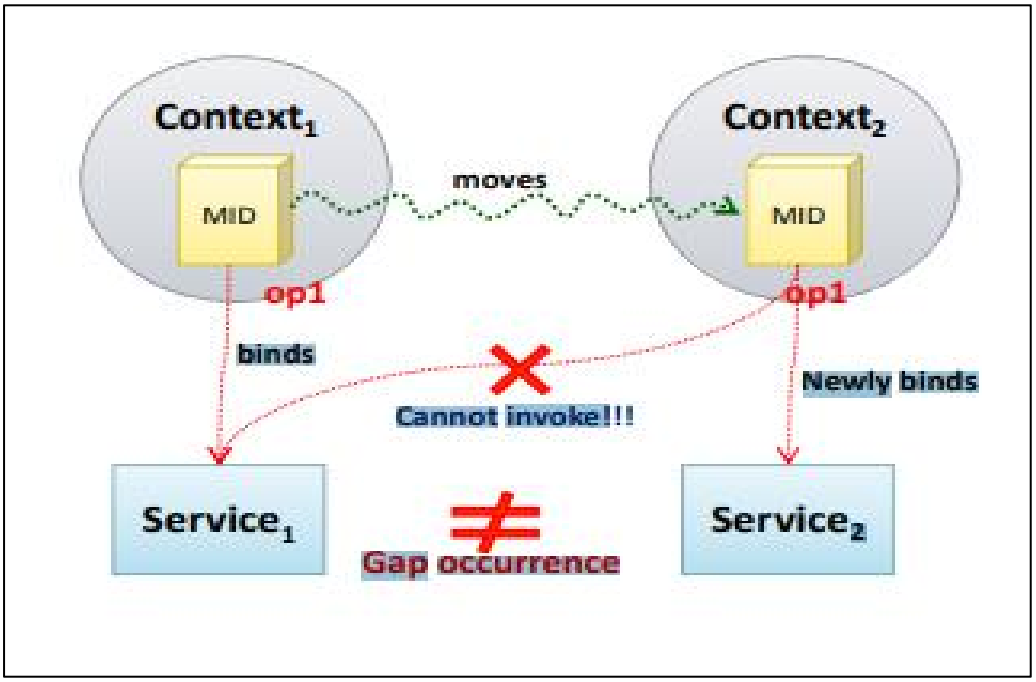

Figure 12: Different Levels of Service Realization.

6) Conclusion

This chapter provides definition of MCC , architecture , application and his Adaptation to mobile context of mobile cloud service.

Mobile devices and their applications take advantage of cloud computing not only to overcome their limitations in computing, memory and storage resources but also to easily scale up to a large number of users.Many common mobile applications we use every day, such as e-mail, news, social networking and games.

Mobile cloud computing allows you to store and retrieve data from anywhere in the world through any device as long as it is connected to the internet.This allows the smooth exchange of data whenever there is a need for information.

Chapter 03:Proposition and Implementation of the Adaptive Mobile Cloud Service

1) Introduction

Cloud service adaptation is the process that transforms cloud service output to the one that fits the context situation, ie. the expected output profile. For this purpose, we use a data-driven service composition method in which adaptation APIs are orchestrated to transform cloud service output to the adapted one to the mobile application.

In this chapter, we explain in detail the proposed method for mobile cloud service adaptation to mobile context. We present also the implementation of the proposition and the obtained results.

2) Proposition of the Adaptive Mobile Cloud Service

2.1) General Architecture of the Proposed Adaptation Method

A cloud proxy is proxy that is based in the cloud instead of in a hardware appliance residing in a corporate data center. A proxy server acts as a gateway between you and the internet, and verifies and forwards incoming client requests to other servers for further communication[35].

The API proxy sends a message containing context parameters to API mashup execution engine. The later asks context manager for expected output profile that fits sent context parameters. After that, it requests adaptation plan from adaptation planner by sending expected output profile. After receiving adaptation plan, it executes the mashup plan and responds proxy API by execution result. The later sends response to the mobile application. Cloud platform that manage this adaptation process is depicted in Figure 13.

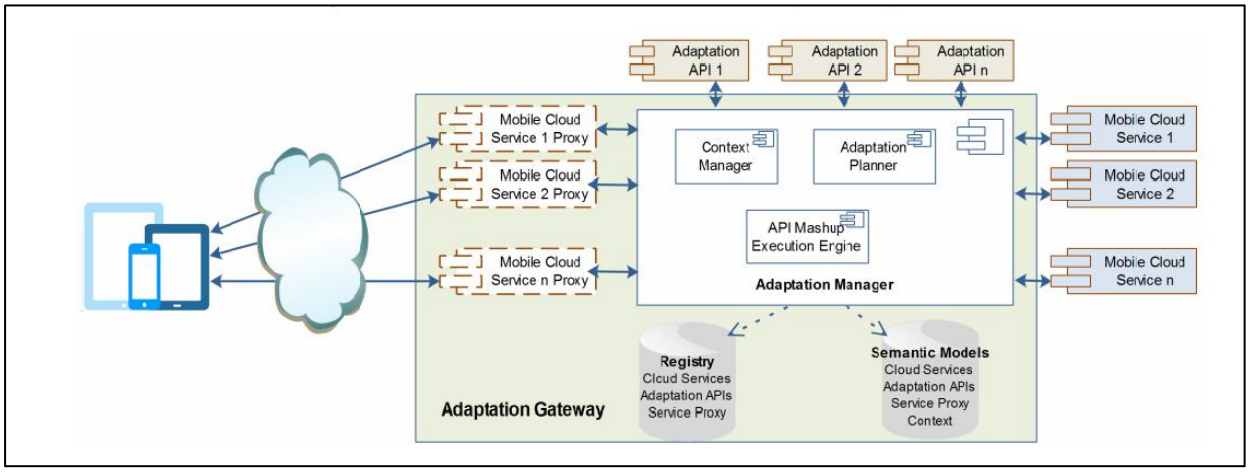

Figure 13:Architecture of context aware cloud service adaptation framework .

2.2) Adaptation API Mashup: Model and Algorithm

Our work revolves around Adaptive Mobile Cloud Service by a Semantic Data-Driven Web API Mashup. we will look for cloud service output that corresponds to the context of the user's context. The graph of API mashup is generated on the fly according to the mobile context while trying to generate the sought mashup plan. Mashup plan is executed by mashup execution engine described in Figure 13.

2.2.1) Definitions

Before detailing the proposed algorithm, we start by defining different used elements and explaining them by examples:

Definition 1: (REST API). A REST API is defined as: AP I = {I, O, P, E}

Inputs I, outputs O, precondition P and postcondition or effect E.

Definition 2 : (Inputs-Outputs' Annotation). It is established by linking each input and output to ontology class or property. They are used when matching inputs to outputs to ensure their adequacy.

Definition 3 : (Precondition-Postcondition). A precondition is a logical predicate about what must be true about the inputs of a web API to guarantee that it will behave as expected. Whereas, a postcondition is a logical predicate that should describe what side effects a web API has after its execution. Example:

```
openapi: "3.0.0"
info:
   version : 1.0.0
   title : image manipulation
"@context":
   class: http://example.org/class/
   prop: http://example.org/property/
   res: http://example.org/resource/
servers:
   - url: http://apis.example.org/
paths:
   /getimage/{imageID}:
      get:
         parameters:
            - name: imageID
              in: path
              required: true
              schema:
                 type: string
                 "@type": class:imageID
         responses:
            200:
               description: returns binary image
               content:
                  application/octet-stream:
                     schema:
                        "@type": class:image
                        type: string
                        format: binary
         precondition: truee
         postcondition: ASK{
           ?x a class:Goal .
           ?x prop:acceptImageFormat ?y .
           ?x prop:minImageSize ?mins .
           ?x prop:maxImageSize ?maxs .
           FILTER( ?mins= 240 && ?maxs=360 ?y IN (res:png , res:jpg)      ).
           }"
```

Figure 14 :OpenAPI with semantic description of Cloud Mobile API.

Definition 4 : (Adaptation Subgoal). It is defined by RDF predicates that makes related precondition query expression true. Example :

```
PREFIX prop:<http://example.org/property/>
PREFIX res:<http://example.org/resource/>
res:goal prop:acceptImageFormatres res:png
res:goal prop:acceptImageFormatres res:bmp
res:goal prop:acceptImageFormatres res:tiff
res:goal prop:minImageSize 144
res:goal prop:maxImageSize 720
```

Figure 15 :Semantic description of expected service output in RDF/RDFS.

Definition 5 : (Adaptation Goal). It is resulted from the conversion of expected output profile to a set of RDF predicates.

```
PREFIX prop:<http://example.org/property/>
PREFIX res:<http://example.org/resource/>
res:goal prop:acceptImageFormatres res:jpg
res:goal prop:acceptImageFormatres res:png
res:goal prop:minImageSize 360
res:goal prop:maxImageSize 360
```

Figure 16 :Semantic description of expected service output: Goal in RDF/RDFs.

2.2.2) The Algorithm

The proposed adaptation API mashup algorithm is a goal-driven reasoning algorithm that generates on the fly API mashup plan and mashup graph for other mashup possibilities . It starts after identifying the expected output profile and transforms it to a goal by adding their RDF predicates to instance ontology. (Expected output profile uses the same RDF schema used de describe the goal).

This profile is transformed to a semantic goal represented by RDF triple, as shown in Figure 16. The graph of API mashup is generated on the fly according to the goal while trying to generate the sought mashup plan. The algorithm is executed by mashup execution engine described in Figure 13. It can be described as follow:

```
MainGoal=Load the main goal to ontology
Goal = MainGoal
for each AApi in ApiList {
    for each AApi1 in ApiList – [AApi] {
       OApiD1 ← get OpenApi description from AApi
       OApiD2 ← get OpenApi description from AApi1
       validateMg ← Execute precondition ASk Query of OApiD1 with Goal
       add AApi to graph
       if ( not validateMg){
            goal ← transform OApiD1 postcondition to goal
            AApig ← Load the goal    to ontology
            AApivalidate = Execute precondition Ask Query of OApiD2 with AApig
            AApicomp = compare @type from OApiD1 parameters with @type from OApiD2 responses
            if (AApicomp == True and AApivalidate == True) {
                 connect AApi with AApi1 in the graph
                 Break
            }

        }
        else{
           add AApi to validateList
           if( the graph is empty){
                pathesList= AApi
                draw the graph with just one node (AApi)
           }
        }
for each node in the graph{
   for each element in validateList{
      add the path from the start node in the graph to the element node to pathesList
   }
}
```

In a for loop, it tests if the main goal validates with the first adaptation API. by:

a. adding goal RDF triples to instances' ontology

b. verifying precondition veracity by the execution of the ASK query of the precondition. It also verifies inputs/output matching

If not validated, it will be added to the graph, then It will validates the current adaptation API with the next adaptation API

If validated, the current adaptation API will be connected with the next one

If not validated, it will exit from the loop and move from the current adaptation API to the next adaptation API.

When it gets to the state where it finds adaptation API that matches the main goal, it adds it to a list and so on . In the end, it will link the first adaptation API with the items in the list (adaptation API that validates with the main goal) to give us mashup plans that link the goal to the available cloud service, if it exists.

3) Implementation

We implemented the proposed algorithm (Adaptation service composition planning algorithm) that we proposed. We used PyCharm as an IDE for Python programming language with rdflib library to manipulate RDF description of goals/subgoals and executing ASK SPARQL queries corresponding to (sub)goal/postcondition validation. To annotate APIs, (sub)goals, and expressing precondition and postcondition predicates, we define a domain ontology schema in RDFS. Instances of APIs and goals/subgoals are defined in an RDF instance ontology. All APIs are described syntactically in OpenAPI YAML format with semantic annotation of request and response parameters and ASK SPARQL query to express precondition/postcondition predicates. When adding a new API to the registry, description schema is browsed recursively to extract parameter annotations and precondition-postcondition statements.

3.1) Required Tools

3.1.1) Programming languages and Libraries

a) Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code

reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed [28].

b) RDF (Resource Description Framework)

RDF is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed [29].

c) RDFS (Resource Description Framework Schema)

RDFS is a general-purpose language for representing simple RDF vocabularies on the Web. Other vocabulary definition technologies, like OWL or SKOS, build on RDFS and provide language for defining structured, Web-based ontologies which enable richer integration and interoperability of data among descriptive communities [30] .

d) SPARQL (SPARQL Protocol and RDF Query Language)

SPARQL enables users to query information from databases or any data source that can be mapped to RDF. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions [31].

e) Rdflib

RDFLib is a pure Python package for working with RDF. RDFLib contains useful APIs for working with RDF, including:

- **Parsers & Serializers**
  - for RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, RDFa and Microdata
  - JSON-LD, via a plugin module
- **Store implementations**
  - for in-memory and persistent RDF storage - Berkeley DB
- **Graph interface**
  - to a single graph
  - or a conjunctive graph (multiple Named Graphs)
  - or a dataset of graphs

- **SPARQL 1.1 implementation**
  - supporting both Queries and Updates [32].

f) NetworkX

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks [33].

g) YAML

YAML It is a human friendly data serialization standard for all programming languages.[34]

3.2) Source Code Overview

Step 01: Transform ask for a goal:

It's done by transforming the ASK format to triple.

1) find_Ask_values(): it takes list of ASK lines and return the variables(ex:? X)

```
115  def find_ASK_values(spltASK):
116      for i in spltASK:
117          if i.find("?") != -1:
118              valList.append(i)
119          else:
120              if i.find("FILTER")!=-1:
121                  break
122      return valList
```

Figure 18 :find_Ask_values Source Code.

Input:

```
postcondition: ASK{
  goal:goal1 schm:minImageSize ?goalmin ;
  schm:maxImageSize ?goalmax  .
  FILTER( xsd:decimal(?goalmin)=144  && xsd:decimal(?goalmax)>360 )
  }
```

Figure 19:find_Ask_values input.

Output:

List = ['?goalmin', '?goalmax']

2) **find_Filter_value():** It return list containing the variables values (ex:144,360…)

```
124  def find_FILTER_values(spltFltr):
125      spltFlt=list()
126      for i in spltFltr:
127          if " IN " in i:
128              befIn=i.rpartition(" IN ")[0]
129              aftIn=i.rpartition(" IN ")[2]
130              spltbefIn = befIn.split()
131              Yvalues=find_between(aftIn,"(",")")
132              yValues=Yvalues.split(",")
133              for t in range(0,len(yValues)):
134                  y1=spltbefIn[-1]+"="+yValues[t]
135                  spltFlt.append(spltbefIn[0])
136                  spltFlt.append(y1)
137          else:
138              spltFlt.append(i)
139      return spltFlt
```

Figure 20 : find_Filter_value Source Code.

Input:

```
parameters:
  - name: imageID
    in: path
    required: true
    schema:
      type: string
      "@type": class:imageID
responses:
  200:
    - description: returns binary image
      content:
        application/octet-stream:
          schema:
            "@type": class:imageID
            type: string
            format: binary
 precondition: truee
PREFIX goal: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#>
PREFIX schm: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
postcondition: ASK{
  goal:goal1 schm:minImageSize ?goalmin ;
  schm:maxImageSize ?goalmax  .
  FILTER( xsd:decimal(?goalmin)=144  && xsd:decimal(?goalmax)>360 )
  }
```

Figure 21 :find_Filter_value input.

Output:

```
PREFIX goal:<https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#>
PREFIX schm:<https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
ASK{
goal:goal1 schm:minImageSize ?goalmin ;
schm:maxImageSize ?goalmax  .
FILTER( xsd:decimal(?goalmin)=144  && xsd:decimal(?goalmax)>360 )
 }
```

Figure 22 :find_Filter_value output.

3) find_Classes(): It gives us the values of the subject (ex: ?x a class goal)

```
145  def find_Classes (ASKlines):
146      valueList = list()
147      xCls = list()
148      for i in ASKlines:
149          if i.find(" a ")!=-1:
150              #find the equal Class of ?x
151              xCls.append(i.rpartition(' a ')[0])
152              xCls.append(i.rpartition(' a ')[2])
153          for j in range(0,len(xCls)):
154              # replace ?x with class:Goal
155              valueList.append(i.replace(xCls[0],xCls[1]))
156      return valueList
```

Figure 23 : find_Classes Source Code.

Input:

```
?x a class:Goal .
?x prop:acceptImageFormat ?y .
?x prop:minImageSize ?mins .
?x prop:maxImageSize ?maxs .
```

Figure 24 :find_Classes input.

Output:

```
class:Goal prop:acceptImageFormat ?y .
class:Goal prop:minImageSize ?mins .
class:Goal prop:maxImageSize ?maxs .
```

Figure 25 :find_Classes output.

4) fltr_value_in_Ask(): It substitutes the variables in Ask to its equivalent in Filter

```
def fltr_values_in_ASK(ASKList,FilterValues,valList):
    simpleL=['=','>',"<"]
    complexL=['!=','>=','<=']
    finalList = list()
    clist=list()
    for t in ASKList:
        for i in FilterValues:
            for j in valList:
                if j in i:
                    j1 = "?" + j
                    if j1 in t:
                        for c in complexL:
                            ic=i.find(c)
                            if ic != -1:
                                clist.append(c)
                                finalList.append(t.replace(j1,i.rpartition(c)[2]))
                        if not clist:
                            for s in range(0, len(simpleL)):
                                if simpleL[s] in i:
                                    finalList.append(t.replace(j1, i.rpartition(simpleL[s])[2]))
                        else:
                            for cl in range(0,len(clist)):
                                for s in range(0, len(simpleL)):
                                    iS = i.find(simpleL[s])
                                    if iS != -1:
                                        if simpleL[s] not in clist[cl] and clist[cl] not in i:
                                            finalList.append(t.replace(j1, i.rpartition(simpleL[s])[2]))
    finalList = list(dict.fromkeys(finalList))
    return finalList
```

Figure 26 :fltr_value_in_Ask source code.

Input:

1. List1 = ['goal:goal schm:minImageSize ?goalmin ;schm:maxImageSize?goalmax ']
2. List2 = [' xsd:decimal(?goalmin)= 144 ', ' xsd:decimal(?goalmax)=360']
3. List3 = ['goalmin', 'goalmax']

Output:

Output List = ['goal:goal1 schm:minImageSize 144 ; schm:maxImageSize ?goalmax ',

'goal:goal1 schm:minImageSize ?goalmin ; schm:maxImageSize 360 ']

5) add_Class_after_semicolon_values(): In the case of a semicolon(;) in Ask, it puts the subject at the beginning of the line until it reaches the point (.)

```
198  def add_Class_after_semicolon_values(finalList,valList):
199      finalList1=list()
200      for i in range (0,len(finalList)):
201          if ";" in finalList[i] :
202              spI=finalList[i].split(";")
203              if i ==0:
204                  finalList1.append(spI[i])
205              else:
206                  fstPart=spI[0].split(" ")
207                  finalList1.append(fstPart[0]+spI[i])
208      for j in valList:
209          for i in finalList1:
210              if j in i:
211                  finalList1.remove(i)
212      return finalList1
```

Figure 27 :add_Class_after_semicolon_values source code.

Input:

1. List 1 = ['goal:goal1 schm:minImageSize 144 ; schm:maxImageSize ?goalmax ', 'goal:goal1 schm:minImageSize ?goalmin ; schm:maxImageSize 360 ']
2. List 2= ['goalmin', 'goalmax']

Output:

Output List = ['goal:goal1 schm:minImageSize 144 ', 'goal:goal1 schm:maxImageSize 360 ']

6) find_PREFIX(): It gives a list of the prefix values

```
86      def find_PREFIX(imagedesc_getmethod):
87          PrefixList=list()
88          for key, value in imagedesc_getmethod.items():
89                  if key.find("PREFIX")!=-1:
90                      PrefixList.append(key+":"+value)
91          return PrefixList
```

Figure 28 :find_PREFIX source code.

Input :

```
parameters:
  - name: imageID
    in: path
    required: true
    schema:
      type: string
      "@type": class:imageID
responses:
  200:
    - description: returns binary image
      content:
        application/octet-stream:
          schema:
            "@type": class:imageID
            type: string
            format: binary
 precondition: truee
PREFIX goal: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#>
PREFIX schm: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
postcondition: ASK{
  goal:goal1 schm:minImageSize ?goalmin ;
  schm:maxImageSize ?goalmax  .
  FILTER( xsd:decimal(?goalmin)=144  && xsd:decimal(?goalmax)>360 )
  }
```

Figure 29 :find_PREFIX input

Output:

```
PREFIX goal: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#>
PREFIX schm: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

Figure 30 :find_PREFIX output .

7) **add_URLs():** It substitutes the prefix to its equivalent in Ask(ex: goal: It substitutes by the URL of prefix goal)

```
214    def add_URLs(finalList, prefix):
215        for i in range(0, len(finalList)):
216            for j in range(0, len(prefix)):
217                spltI = prefix[j].replace("PREFIX ", "")
218
219                spltI = spltI.rpartition('#')[0] + spltI.rpartition('#')[1]
220
221                if spltI.rpartition('<')[0] in finalList[i]:
222                    finalList[i] = finalList[i].replace(spltI.rpartition('<')[0], spltI.rpartition('<')[2])
223        finalList=list(dict.fromkeys(finalList))
224        Goal = listToString(finalList)
225        print(Goal)
226        return finalList
```

Figure 31 :add_URLs source code.

Input:

1. List = ['goal:goal1 schm:minImageSize 144 ', 'goal:goal1schm:maxImageSize360 ']

2.

```
PREFIX goal: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#>
PREFIX schm: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

Figure 32 : add_URLs input 2

Output:

1. Print Goal

```
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#goal1
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#minImageSize 144
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#goal1
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#maxImageSize 360
```

Figure 33 :add_URLs output 1.

2. Return list

```
['https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#goal1
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#minImageSize 144 ',
'https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#goal1
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#maxImageSize 360 ']
```

Figure 34 : add_URLs output 2.

8) get_Goal(): It converts ask into a goal using previous functions from 1 to 7.

```
286  def get_Goal(imagedesc_getmethod):
287      finalList=list()
288      ASK=imagedesc_getmethod["postcondition"]
289      ASKlines=find_between(ASK,"ASK{ ","FILTER").split(".")
290      spltASK=imagedesc_getmethod["postcondition"].split(" ")
291      Filter=find_between(imagedesc_getmethod["postcondition"],"FILTER(" )")
292      spltFltr=Filter.split("&&")
293      valList=find_ASK_values(spltASK)
294      spltFlt=find_FILTER_values(spltFltr)
295      valueList=find_Classes_(ASKlines)
296      for i in range(0,len(valList)):
297          valList[i]=valList[i].replace("?","")
298      for i in ASKlines:
299          if ";" not in i:
300              for i in fltr_values_in_ASK(valueList, spltFlt,valList):
301                  finalList.append(i)
302          else:
303              finalList1 = fltr_values_in_ASK(ASKlines, spltFlt, valList)
304              for i in add_Class_after_semicolon_values(finalList1, valList):
305                  finalList.append(i)
306      prefix=find_PREFIX(imagedesc_getmethod)
307      finalList=add_URLs(finalList,prefix)
308      return finalList
```

Figure 35 :Find_Goal source code.

Input:

```
parameters:
  - name: imageID
    in: path
    required: true
    schema:
      type: string
      "@type": class:imageID
responses:
  200:
    - description: returns binary image
      content:
        application/octet-stream:
          schema:
            "@type": class:imageID
            type: string
            format: binary
 precondition: truee
PREFIX goal: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#>
PREFIX schm: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
postcondition: ASK{
  goal:goal1 schm:minImageSize ?goalmin ;
  schm:maxImageSize ?goalmax  .
  FILTER( xsd:decimal(?goalmin)=144  && xsd:decimal(?goalmax)>360 )
  }
```

Figure 36 :Find_Goal input

Output:

```
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#goal1
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#minImageSize 144
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#goal1
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#maxImageSize 360
```

Figure 37 :Find_Goal output.

Step 02: Loading the goal to the ontology

This is done by converting the triple format to an RDF file(ex: goal.rdf).

9) Goal_to_RDF(): Loading the goal in ontology(rdf file)

```
def goal_to_RDF_file(finalList):

    schm = Namespace('https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#')
    goal = URIRef("https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf")
    g = rdflib.Graph()
    g.namespace_manager = NamespaceManager(Graph())
    g.namespace_manager.bind('schm', schm)
    g.add((schm['JPGFormat'], RDF.type, schm["JPGFormat"]))
    for i in finalList:
        splt_i1 = i.split(" ")
        splt_i=[e for e in splt_i1 if len(e.strip())!=0]
        for j in range(0, len(splt_i)):
            if splt_i[j] == splt_i[0] :
                Class = URIRef(splt_i[j])
                goal = URIRef("https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#goal1")
                g.add((goal, RDF.type, Class))
            else:
                colone2 = rdflib.term.URIRef(splt_i[1])
                forResource = colone2.find("acceptImageFormat")
                if forResource != -1:
                    imgFormatlist.append(colone2)
                    for i in imgFormatlist:
                        g.add((goal, schm['acceptImageFormat'], schm['JPGFormat']))
                else:
                    g.add((goal, colone2, Literal(splt_i[2])))
    define_file = open("Goal_file.rdf", "w+")
    define_file.write(g.serialize(format="pretty-xml").decode("utf-8"))
    define_file.close()
    return g
```

Figure 38 :Goal_to_RDF source code .

Input:

```
['https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#goal1
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#minImageSize 144 ',
'https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#goal1
https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#maxImageSize 360 ']
```

Figure 39 :Goal_to_RDF input .

Output:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF

xmlns:schm="https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#"

xmlns:ns1="https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <schm:JPGFormat
rdf:about="https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#JPGFormat"/>
  <ns1:goal1
rdf:about="https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#goal1">
    <schm:maxImageSize>360</schm:maxImageSize>
    <schm:minImageSize>144</schm:minImageSize>
  </ns1:goal1>
</rdf:RDF>
```

Figure 40 :Goal_to_RDF output.

Step 03: Execute ASK Query:

Execute an ASK query using the ontology to validate the ASK with the goal.

10) get_AskQuery(): return AskQuery of adaptation Api

```
278    def get_AskQuery(imagedesc_getmethod):
279        per=""
280        for i in find_PREFIX(imagedesc_getmethod):
281            per=per+i+"\n"
282
283        AskQuery=per+imagedesc_getmethod['postcondition']#postconditio
284        return AskQuery
```

Figure 41 :get_AskQuery source code.

Input:

```
parameters:
  - name: imageID
    in: path
    required: true
    schema:
      type: string
      "@type": class:imageID
responses:
  200:
    - description: returns binary image
      content:
        application/octet-stream:
          schema:
            "@type": class:imageID
            type: string
            format: binary
 precondition: truee
PREFIX goal: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#>
PREFIX schm: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
postcondition: ASK{
  goal:goal1 schm:minImageSize ?goalmin ;
  schm:maxImageSize ?goalmax  .
  FILTER( xsd:decimal(?goalmin)=144  && xsd:decimal(?goalmax)>360 )
  }
```

Figure 42 : get_AskQuery input

Output:

```
PREFIX goal:<https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#>
PREFIX schm:<https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
ASK{
goal:goal1 schm:minImageSize ?goalmin ;
schm:maxImageSize ?goalmax  .
FILTER( xsd:decimal(?goalmin)=144  && xsd:decimal(?goalmax)>360 )
 }
```

Figure 43 : get_AskQuery output

11) **ExecuteQuery():** It gives us a Boolean values (true or false)(ex: If the goal matches the Ask Query, it returns true)

```
110  def ExecuteQuery(graph,imagedesc_getmethod):
111      AskQuery = get_AskQuery(imagedesc_getmethod)
112      for row in graph.query(AskQuery):
113          return row
```

Figure 44 : ExecuteQuery source code.

Input:

1.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF

xmlns:schm="https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#"

xmlns:ns1="https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <schm:JPGFormat
rdf:about="https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#JPGFormat"/>
  <ns1:goal1
rdf:about="https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#goal1">
    <schm:maxImageSize>360</schm:maxImageSize>
    <schm:minImageSize>144</schm:minImageSize>
  </ns1:goal1>
</rdf:RDF>
```

Figure 45 : ExcuteQuery input 1.

2.

```
    parameters:
      - name: imageID
        in: path
        required: true
        schema:
          type: string
          "@type": class:imageID
    responses:
      200:
        - description: returns binary image
          content:
            application/octet-stream:
              schema:
                "@type": class:imageID
                type: string
                format: binary
   precondition: truee
   PREFIX goal: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/instanceonto.rdf#>
   PREFIX schm: <https://raw.githubusercontent.com/mbeggas/Mobile_Cloud_Adaptation/master/ontologies/schemaonto.rdfs#>
   PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
   postcondition: ASK{
    goal:goal1 schm:minImageSize ?goalmin ;
    schm:maxImageSize ?goalmax  .
    FILTER( xsd:decimal(?goalmin)=144  && xsd:decimal(?goalmax)>360 )
    }
```

Figure 46 :ExcuteQuery input 2.

Output:

Boolean value (True /False)

Step 04: Taking Input/Output description

Extract the @type of goal with @type of adaptation API.

12) ResType(): It return list of @type found in response.

```
def ResType(dic):
    resType = list()
    dictt=dic.get(200)
    if type(dictt)== list:
        for i in dictt:
            class DObj(object):
                pass
            dobj = DObj()
            dobj.__dict__ = i
            i=dobj.content
            for key,value in i.items():
                    if key == "application/octet-stream":
                        val=value
                        for key, value in val.items():
                            if key == "schema":
                                val = value
                                resType.append(val.get('@type'))
    else:
        class DObj(object):
            pass
        dobj = DObj()
        dobj.__dict__ = dictt
        dictt= dobj.content
        for key, value in dictt.items():
            if key == "application/octet-stream":
                val = value
                for key, value in val.items():
                    if key == "schema":
                        val = value
                        resType.append(val.get('@type'))
    return resType
```

Figure 47 :ResType source code .

Input:

```
200:
 - description: returns binary image
   content:
     application/octet-stream:
       schema:
         "@type": class:imageID
         type: string
         format: binary
 - description: returns binary image
   content:
     application/octet-stream:
       schema:
         "@type": class:imageID
         type: string
         format: binary
```

Figure 48 :ResType input.

Output:

List = ['class:imageID', 'class:imageId']

13) ParType(): It return list of @type found in parameters.

```
14
15  def ParType(dic):
16      dictlistPar = list()
17      parType = list()
18      lst = list()
19      for i in dic:
20          for key, value in i.items():
21              temp = [key, value]
22              dictlistPar.append(temp)
23
24      for i in dictlistPar:
25          for key in i:
26              if key == "schema":
27                  for j in i:
28                      if j !="schema":
29                          lst.append(j)
30                          #print(lst)
31
32      for l in lst:
33          parType.append(l.get('@type'))
34      return parType
35
```

Figure 49 :ParType source code .

Input:

```
parameters:
  - name: imageID
    in: path
    required: true
    schema:
      type: string
      "@type": class:imageID
  - name: imageType
    in: path
    required: true
    schema:
      type: string
      "@type": class:imageID
```

Figure 50 :ParType input

Output:

List = ['class:imageID', 'class:imageId']

Step 05: Compare inputs with outputs:

Where we compare @type of goal with @type of adaptation API.

14) **Comparison():** It compares ResType() output with ParType() output and gives us a Boolean value (true or false)

```
def Comparison(parameters ,responses):
    typeResList = ResType(responses)
    typeParList = ParType(parameters)
    if len(typeResList )== len(typeParList):
        for i in range(0,len(typeResList)):
            for j in range(0,len(typeParList)):
                if typeParList[j]!=typeResList[i]:
                    return False
                    quit() # if there is one false then it's false
                else:

                    if i==(len(typeResList)-1) & j==(len(typeParList)-1):
                        return True
                        quit() # if all true then it's true
    else:
        return False
```

Figure 51 :Comparison source code .

Input: 1.

```
parameters:
  - name: imageID
    in: path
    required: true
    schema:
      type: string
      "@type": class:imageID
  - name: imageType
    in: path
    required: true
    schema:
      type: string
      "@type": class:imageID
```

Figure 52 :Comparison input 1.

2.

```
200:
 - description: returns binary image
   content:
     application/octet-stream:
       schema:
         "@type": class:imageID
         type: string
         format: binary
 - description: returns binary image
   content:
     application/octet-stream:
       schema:
         "@type": class:imageID
         type: string
         format: binary
```

Figure 53 :Comparison input 2.

Output :

Boolean value (True /False) .

3.3) Results

During the implement we used five files yaml, one of them considered the main Goal and called it **retrieveimage1-api**.YAML and the remaining four files ("**retrieveimage2-api.YAML**" ,"**retrieveimage3-api.YAML**","**retrieveimage4-api.YAML**", "**retrieveimage5-api.YAM**L") we considered them as adaptation API, where the difference between these five files was the **maxImageSize** as follows:

- retrieveimage1-api.YAML : maxImageSize = 360
- retrieveimage2-api.YAML : maxImageSize =720
- retrieveimage3-api.YAML : maxImageSize >=360
- retrieveimage4-api.YAML : maxImageSize >360
- retrieveimage5-api.YAML : maxImageSize >=360

We started the comparison from retrieveimage2-api.YAML to retrieveimage5-api.YAML, and by looking at the suggested values, we see that retrieveimage3-api.YAML and

retrieveimage5-api.YAML correspond to retrieveimage1-api.YAML and this is what we found as a result of our work shown bellow:

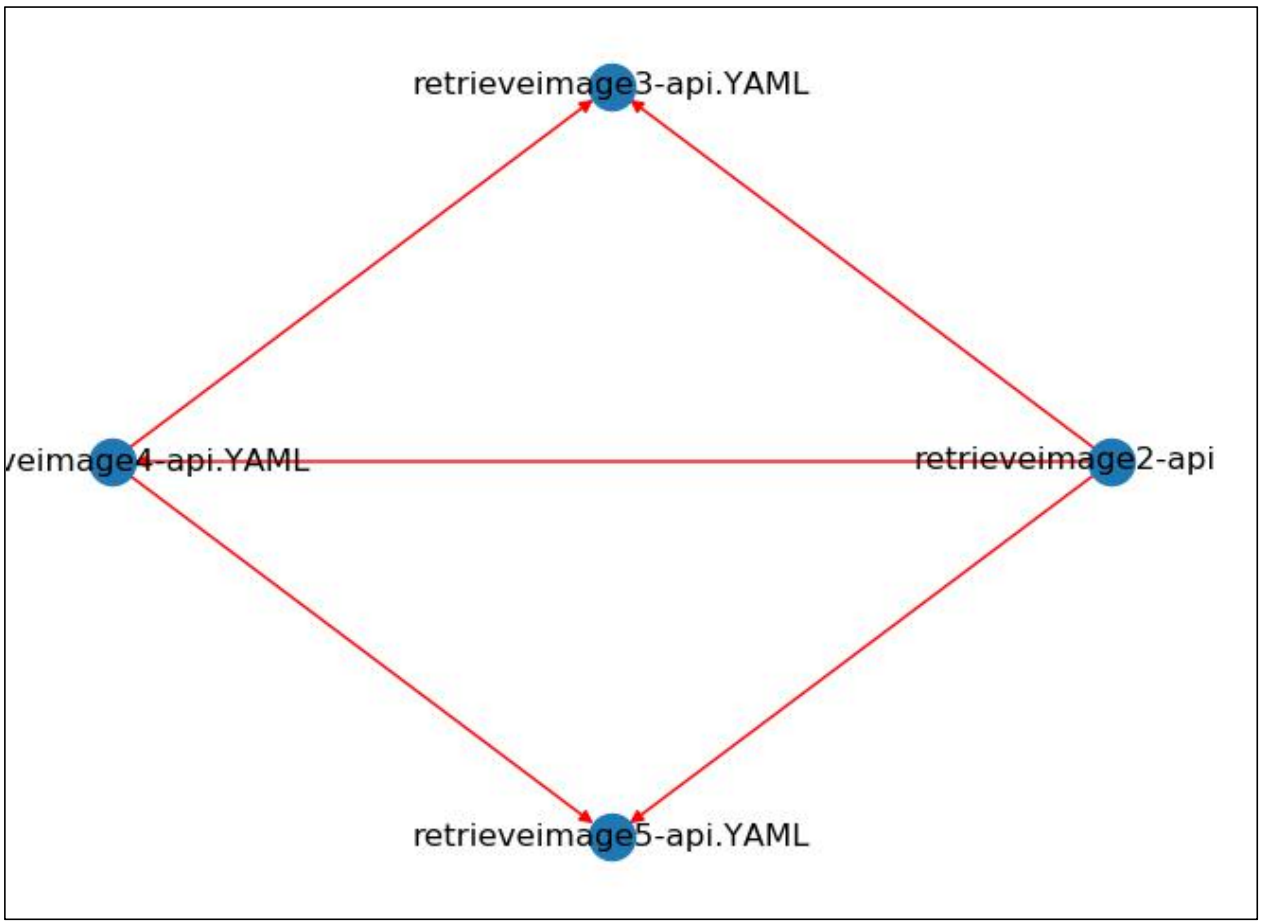

Figure 54 :Interconnection of adaptation API graph.

```
[('retrieveimage2-api.YAML', 'retrieveimage3-api.YAML'),
('retrieveimage2-api.YAML', 'retrieveimage4-api.YAML','retrieveimage3-api.YAML'),
('retrieveimage2-api.YAML', 'retrieveimage4-api.YAML', 'retrieveimage5-api.YAML'),
('retrieveimage2-api.YAML', 'retrieveimage5-api.YAML')]
```

Figure 55 :pathes list .

Then we did another experiment with a simple change:

- retrieveimage4-api.YAML : maxImageSize =240

it gives us the following result:

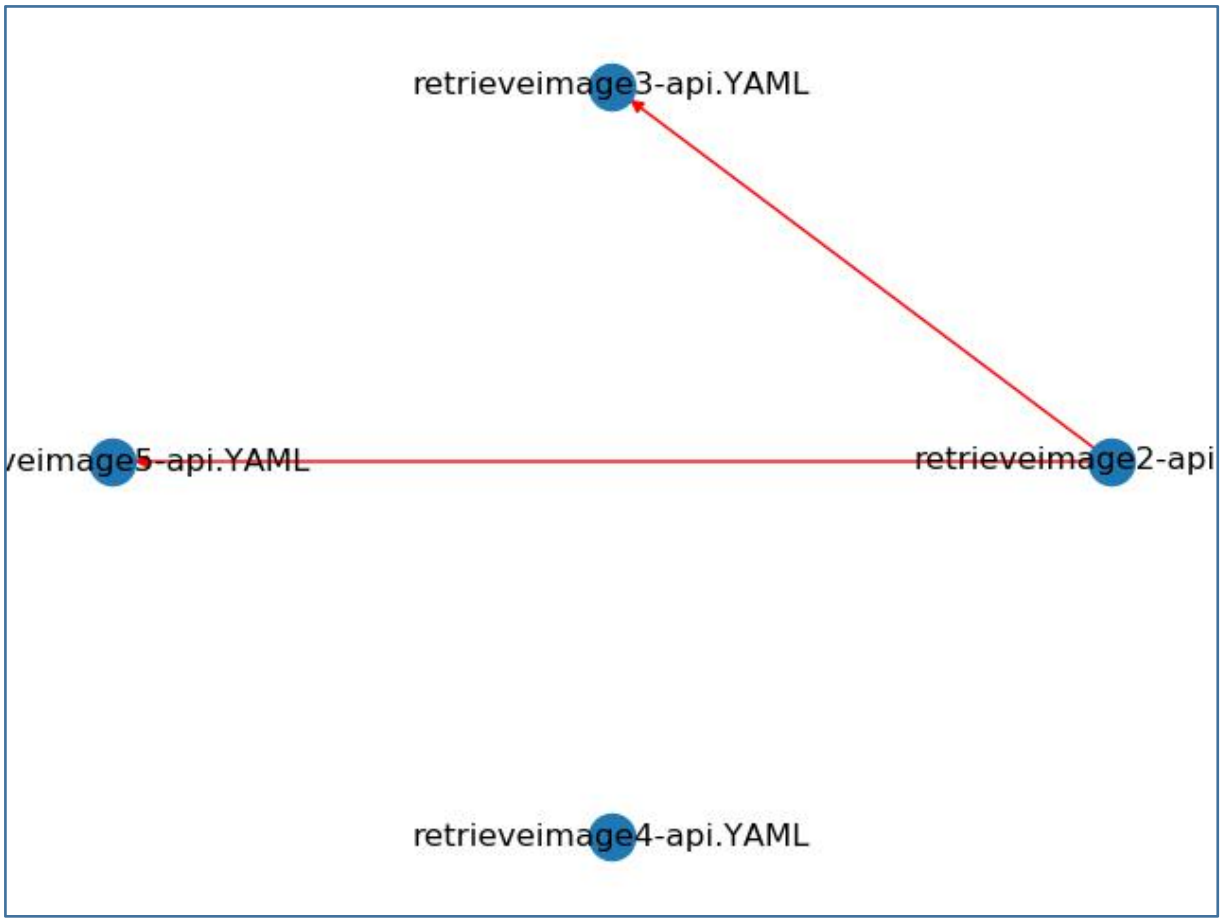

Figure 56: Interconnection of adaptation API graph1.

```
[('retrieveimage2-api.YAML', 'retrieveimage3-api.YAML'),
('retrieveimage2-api.YAML', 'retrieveimage5-api.YAML')]
```

Figure 57 :pathes list1.

in the end, we try the case when the first service is the one that matches with the main goal:

- retrieveimage2-api.YAML : maxImageSize =360

it gives us the following result:

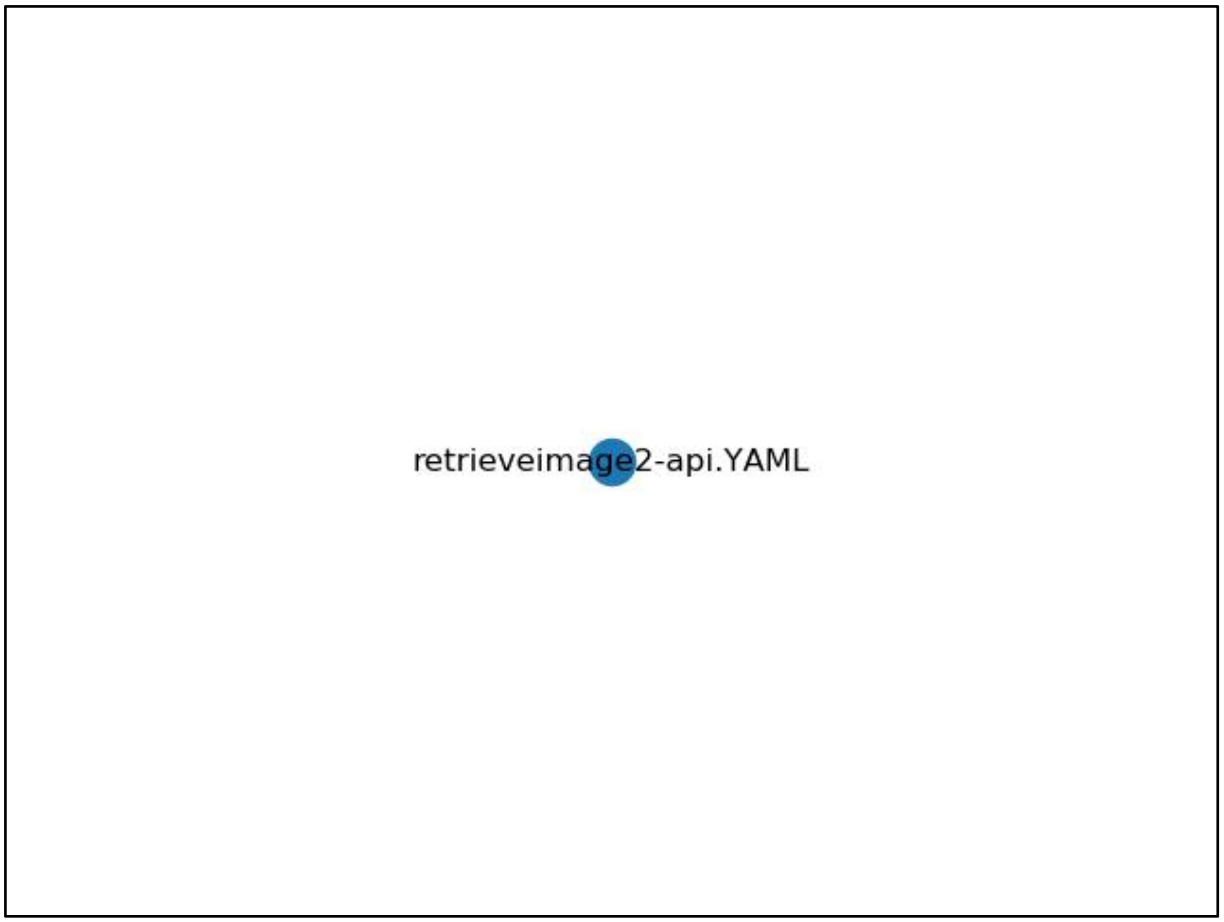

Figure 58 :Interconnection of adaptation API graph2

```
['retrieveimage2-api.YAML']
```

Figure 59 :pathes list2.

4) Conclusion

In this chapter, we proposed a cloud service adaptation process that deduces, according to mobile context, the relevant API mashup plan. The plan is defined by an orchestration of data-driven APIs to transform the primary output to context-relevant ones. The automatic adaptation process is carried out by a gateway in which each cloud service is represented by a proxy API. The adaptation is performed by an automatic data-driven mashup process that orchestrates adaptation APIs to transform cloud service output to corresponding one to expect output profile. API mashup is based on semantic annotation of input/output and precondition/postcondition. We used an algorithm used for service composition: backward chaining AI planning (based on postcondition/precondition validation) and graph algorithm (based on input/output matching). Back chaining AI planning that starts from the expected output (the goal) and matches and validates the is matching by input-output matching to create the mashup plan with the cloud mobile service. For REST API description we used OpenAPI and extend it by inserting semantic annotations (i.e., links to ontology classes and ontology properties) through the use of the Yaml.

GENERAL CONCLUSION

In this work, we adopted and extended the work of [37] to propose a cloud service adaptation process that deduces, according to mobile context. The API mashup plan is defined by an orchestration of data-driven APIs to transform primary output to context / relevant one. The automatic adaptation process is carried out by a gateway in which each cloud service is represented by a proxy API.When mobile application calls cloud mobile service API proxy, the adaptation is performed by automatic data-driven mash up process that orchestrates adaptation APIs so as to transform cloud service output to the corresponding one to expect output profile.

In this master thesis, we proposed an adaptation process of cloud service to mobile context by a mashup of adaptation APIs in. The adaptation process is generated by mashing up the cloud service with adaptation APIs to transform the cloud service output to the appropriate one in the current mobile context. To define the appropriate output, we proposed the concept of the expected profile (adaptation goal) that is deducted from the context situation. An algorithm is proposed for adaptation API mashup based on goal-postcondition validation and input/output matching.The adaptation of cloud service to mobile context is carried out by generating and executing the adaptation API mashup plan in such a way as to satisfy the expected output profile.

Mobile cloud computing could be described as the availability of cloud computing services . world wide distributed storage system, exceed the traditional mobile device capabilities, and offload processing, storage, and security.

In the future, We can generalize the system by allowing the user to introduce the adaptation model which will be transformed into an adaptation API or even define by itself his adaptation API. We can also add the QoS level which will be used to select the best composition plan whenever several possible mashup plans exist. We’ll be also ooking at ways of using the proposed semantic API description on the internet of things to automate the discovery and ingration of sensors and actuators.

BIBLIOGRAPHY

[1] .Mohamed Wiem Mkaouer, Web service API recommendation for automated mashup creation using multi-objective evolutionary search.October 2019

[4] .Roy Thomas Fielding, “REST API Must Be Hypertext Driven,” 28 October, 2008.

[8] .Y. J. Lee and J. H. Kim, “Semantically Enabled Data Mashups using Ontology Learning Method for Web APIs,” Proceedings of the 2012 Computing, Communications and Applications Conference, 2012

[9] .A. H. Ngu, M. P. Carlson, Q. Z. Sheng, and H. Y. Paik, “SemanticBased Mashup of Composite Applications,” IEEE Transactions on Services Computing, Vol. 3, No. 1, pp. 2-15, 2010

[10].A. H. Ngu, M. P. Carlson, Q. Z. Sheng, and H. Y. Paik, “Semantic-based mashup of composite applications,” IEEE Transactions on Services Computing, Vol. 3, No. 1, 2010, pp. 2-15.

[11].Y. J. Lee, “Semantic-Based Data Mashups Using Hierarchical Clustering and Pattern Analysis Methods,” Journal of Information Science and Engineering, Vol. 30, No. 5, 2014, pp. 1601-1618.

[12] .D. Bianchini, V. D. Antonellis, and M. Melchiori, “Semantic-driven mashup design,” in Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services, 2010, pp. 247-254,

[13] .“Y. J. Lee, “Semantic-Based Data Mashups Using Hierarchical Clustering and Pattern Analysis Methods,” Journal of Information Science and Engineering, Vol. 30, No. 5, 2014, pp. 1601-1618"’

[14].Y. J. Lee, “Semantic-Based Data Mashups Using Hierarchical Clustering and Pattern Analysis Methods,” Journal of Information Science and Engineering, Vol. 30, No. 5, 2014, pp. 1601-1618'

[15].“D. Bianchini, V. D. Antonellis, and M. Melchiori, “Semantic-driven mashup design,” in Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services, 2010, pp. 247-254,”

[17].M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, “Semantic matching of web services capabilities,” in Proceedings of the International Semantic Web Conference, 2002, pp. 333-347.

[20]. Baldauf, M., Dustdar, S., Rosenberg, F.. A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing 2007;2(4):263–277.

[21].Gabriel Orsini, Dirk Bade, Winfried Lamersdorf ,.Generic Context Adaptation for Mobile Cloud Computing Environments. International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016) ;(2016) 17 – 24

[22]. Orsini, G., Bade, D., Lamersdorf, W.. Computing at the mobile edge: Designing elastic android applications for computation offloading. In: 8th Joint IFIP Wireless and Mobile Networking Conference (WMNC). IEEE Explore Washington/DC, USA; 2015, p. 8.

[23] .Yang, S.J.H, Lan, B.C.W., and Chung, J.Y., “A New Approach for Context Aware SOA,” In Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and eService (EEE 2005), pp. 438-443, 2005.

[24] .Maamar, Z., Mostefaoui, S.K., and Mahmoud, Q.H., “Context for Personalized Web Services,” In Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS 2005), pp. 66- 73, 2005.

[25] .Sheng, Q.Z., Benatallah, B., and Maamar, Z., “User-Centric Services Provisioning in Wireless Environments,” Communications of the ACM, Vol. 51, No. 11, pp. 130-135, 2008.

[26] .Choi, O.J. and Yoon, Y.I., “A Meta Data Model of Context Information for Dynamic Service Adaptation,” In Proceedings of 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE 2007), pp. 108-113, 2007.

[27] .Chen, I. Y.L, Yang, S.J.H, and Zhang, J., “Ubiquitous Provision of Context Aware Web Service,” In Proceedings of the 2006 IEEE International Conference on Services Computing (SCC 2006).

[36]. Dinh, H.T., Lee, C., Niyato, D., Wang, P.. A survey of mobile cloud computing: architecture, applications, and approaches. Wireless Comm and Mobile Computing 2011;13(18):1587–1611.

[37].Mounir Beggas, Mouadh Bali, Samir Othmani and Messaoud Abbas. Adding Adaptivity to Mobile Cloud Service by a Semantic Data-Driven Web API Mashup ACM The 8th International Conference on Software Engineering and New Technologies "ICSENT'2019" Hammamet, Tunisia.

[2] .https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, Accessed 10/2020.

[3]. https://www.ibm.com/cloud/learn/rest-apis, Accessed 10/2020.

[5] .https://developers.evrythng.com/docs/openapi-description, Accessed 10/2020.

[6] .https://swagger.io/docs/specification/about/, Accessed 10/2020.

[7] .https://swagger.io/specification/, Accessed 10/2020.

[16].https://swagger.io/docs/specification/basic-structure/, Accessed 10/2020.

[18] .http://www.mobilecloudcomputingforum.com/, Accessed 10/2020.

[19].https://onlinelibrary.wiley.com/doi/full/10.1002/wcm.1203., Accessed 10/2020.

[28].https://www.python.org/doc/essays/blurb/ , Accessed 10/2020.

[29]https://www.w3.org/RDF/ , Accessed 10/2020.

[30] https://www.w3.org/2001/sw/wiki/RDFS , Accessed 10/2020.

[31].https://www.w3.org/TR/rdf-sparql-query/ , Accessed 10/2020.

[32].https://rdflib.readthedocs.io/en/stable/ , Accessed 10/2020.

[33].https://networkx.github.io/ , Accessed 10/2020.

[34].https://yaml.org/, Accessed 10/2020.

[35].https://www.zscaler.com/resources/security-terms-glossary/what-is-cloud-proxy, Accessed 10/2020.