Resource Description Language for Distributed RESTful Service Permission Management

Arbër Beshiri

South East European University, Faculty of Contemporary Sciences and Technologies, Department of Software Engineering and Telecommunication, Ilindenska no. 335, 1200 Tetovo, R. Macedonia

Abstract **- In this paper is defined a description language for REST services in order to enable fine-grained permission declaration for authorization purposes. These services are managed by a specific service in the distributed environment. Through a specific service provides permission and workflow management that are distributed in many REST services. The description enables data filtering of services based in parameters. The service permissions are based on HTTP operations and they are identified by unique URLs. The service is implemented separately, while the permission management is delegated in the central service which coordinates all other interacting services.**

Keywords **– Description Language, REST Services, HTTP Operations, Permission, URL**

1. Introduction

The description of services, authorization management through REST services [1] and a description language for REST services, in order to allow the declaration of more fine-grained permissions is discussed in this paper. These permissions will be managed in a separate service in the distributed environment. The goal is to provide the possibility of declaring permissions and workflows that may involve multiple REST services. At the same time, this approach allows having thinner service implementations since the permission management will be delegated to a central service by coordinating all other services.

In such systems, many services need to use common resources. Ideally, in such situations, one would avoid data duplication as much as possible. Service-Oriented Architecture (SOA) [2] facilitates the implementation of such systems by creating different services which are shared by a number of applications or other services. Generally, when we create a REST service for a particular data set, we export all the data through the services.

In situations that require a greater flexibility, such as when it involves privacy issues, we want to restrain the set of data that is accessible from an application/service. For instance, if a service returns the list of students, a learning management system should not have access to information related to all personal data of a student. For such purposes, in current REST approaches [1], one would implement a specific service endpoint, so a service might provide the necessary filtering.

2. REST services

REST [1] stays for Representational State Transfer which is an architectural style for designing of distributed applications [3]. It is architecture style for building of modern web software that in recent years has strength popularity and it is used as a way to building distributed services recognized as REST or RESTful services. It is an architecture style (collection of principles) for building web services, simple, heterogeneous and web-based format. REST presents a powerful and flexible technique on support of communication in distributed environments.

It has HTTP as a base which is transformed from one communication protocol to a protocol which is applied for the interoperability of services in various environments. The different data interacts among them through of web, while HTTP acts as a communication protocol and application by offering direct stateless access and operations. Resources are identified by unique URLs[1] and their state is addressed and transferred via HTTP for wide range

DOI: 10.18421/TEM54-19
https://dx.doi.org/10.18421/TEM54-19

Corresponding author: Arbër Beshiri,
South East European University, Faculty of Contemporary Sciences and Technologies, Tetovo, R. Macedonia
Email: ab18351@seeu.edu.mk

© 2016 Arbër Beshiri.
Published by UIKTEN.
This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 License.
The article is published with Open Access at www.temjournal.com

[1] http://www.w3.org/TR/url/

of services and clients. RESTful design, for simplifying purpose of using, developing and deploying on web restricts the web architecture.

This design uses the client-server architecture which separates interface from data storage and the biggest benefit is that components can be developed regardless from each other without influencing in their stability. The constraint state "requires" from the client to maintain application state. REST architecture is reversed to Simple Object Access Protocol (SOAP)[2] and exchanged messages are transmitted directly on HTTP protocol without encapsulations and using envelopes, which present a big advantage for REST. Such approach is in the application interest where interoperability is more important than formal relations between peers.

REST architecture contains five restrictions and they are defined on the basis of RESTful style:

- **The uniform interface**. Represents the interface between clients and servers. It simplifies architecture and provides each part to act and develop independently.
- **Stateless.** Means the state to handle a request as part of the URI, query-string parameters, body or headers.
- **Cacheable.** Clients can cache responses. Responses should be implicitly or explicitly defined as cacheable; to prevent clients reusing stale, inappropriate data in response, completely eliminates some client-server interactions by improving scalability and performance.
- **Client-server.** Means that customers "are not associated" with the data storage which are found in each server, so the portability of client code is improved. Servers stay separate from the user interface or the state of the users and thus servers can be more simple and scalable. Although there is separator between servers and clients, each part can be replaced and developed independently from each other.
- **Layered system.** There is no need to tell the client regularly if it is connected directly to the end server or with any intermediate server. Intermediate servers improve scalability by providing load balancing, cache distribution and strengthen security policies [4], [5].

Services that adhere to these restrictions are considered RESTful. These restrictions do not dictate technology in which the service is developed. By respecting these guidelines and the best practices, the service can be scalable, portable, reliable, and visible and enables best performance.

REST mostly operates in four different functions (operations):

1) **GET** – used to retrieve data from the server or return a list of resources. For example: GET /articles – returns a list of articles, GET /articles/234 – returns the article with id=234.
2) **POST** – creates a new resource (data) in server. For example: POST /article – create an article.
3) **PUT** – updates an existing resource (data). For example: PUT /article/134 – updates existing article with id=134.
4) **DELETE** – removes the data (resources). For example: DELETE /article/123 – removes an article with id=123.
5) **PATCH** – partial updates an existing resource [6]. For example: PATCH /article/12. It partial updates an existing article with id=12.

REST also has some particularities:

a) Resource identification.
b) Resource manipulation through representation.
c) Self-description messages.
d) Hypermedia as the engine of application state (HATEOAS).

These principles describe the system architecture and interaction of the web. Building blocks on the web present resources. This is marked as a target of the hypertext (file, script or resource collections). As a response of request, a client accepts representation of the resource. Manipulations with resources are made through messages. In the web, messages are HTTP methods. While with fourth principle understands that the state of any client-server interaction is keeping in hypermedia. Each state of information is passed between client and server through messages, thus keeping them both stateless.

According to Roy Fielding [1], in REST architecture, clients and servers exchange resources/data using standardized interface and protocol. The reason why REST is used is that it supports operations through HTTP verbs, facilitates state treatment and the variability depends of resource characteristics. Since REST relays in HTTP, authorization is done on the protocol level using interface such as HTTP and operations (GET, POST, PUT, PATCH and DELETE). If we get in consideration the flexibility, REST can work with all types of resources, which can be represented in XML [7] and JSON [8] formats [4].

[2] http://www.w3.org/TR/soap/

2.1. RESTful resources

The main part of REST considers resources. The resource is something that we can access or manipulate. Resources can be "videos", "profiles of users", "images", "tools" and "personas". They can connect with other resources. For example, in the e-commerce service, a client can order many products. In this case products' resources are related to resources that correspond with the order. Resources can be grouped even in collections. In case of the e-commerce service, "orders" is a collection of individual resource "order" [6].

2.2. URI templates

We should represent the structure of URI when we work with REST services and APIs. To elaborate this case, as example is taken a blog application with URI http://blog.example.com/2013/posts where we can retrieve all blog posts created in year 2013. In this model, it is necessary to know the URI structure for http://blog.example.com/year/posts which describes the range of URI. URI templates are defined by Internet Engineering Task Force (IETF) in RC6570[3] through which a standardized mechanism for describing of the URI structure is offered. For the above mentioned example as a standard for URI may be: http://blog.example.com/{year}/posts.

For this example, curly braces tell the field of year in the part of template, which is field of variable. Clients or services can get URI template as input by replacing variable year with their value and so retrieve blog posts of certain year. While in the server-side, URI template enables server code to parse and retrieve values of selected variables of URI in easier way as much as possible [6].

3. Composed RESTful service

Composed REST service uses existing REST services to offer new functionality in their building. It facilities the using of existing services and combines them entirely to serve different purposes. They participate in composed processes and central controls of processes by coordinating execution of services. Composed service in RESTful style distinguishes from composition technics of traditional web services; respectively this composition is made according to the operations perspective.

[3] http://tools.ietf.org/html/rfc6570

Composed RESTful service focuses in resources. It takes resources as a block that is building in a typical environment for development of REST, where new resources are exposed through a simple design and enable maximum decoupling. Usually the REST interface provides addressing, connectivity and uniform interface features by building a composed RESTful service which it gets in consideration calls of methods in partner services [9], [10].

Hereby, for a RESTful service scenario the framework architecture is given. Services are considered as a composed part of it and offer resources that can be accessed any time. Accessed services and new added services should be easy and entirely realized in accordance to RESTful principles. Simple services offer access in data from databases and support specific query for data, while composed services combine or widen the functionality of one or more other services as a composed part of their.

A service is accessed data through Service A. One or more services must be authorized to complete a specific task. In this case, the authorization is managed in coordinated way from the Authorization Management Service (AMS). Services act to AMS to prove their identity. These act to AMS to validate a request service identity and with possibility to control rights and permissions access of services for restriction resources (Figure 1.).

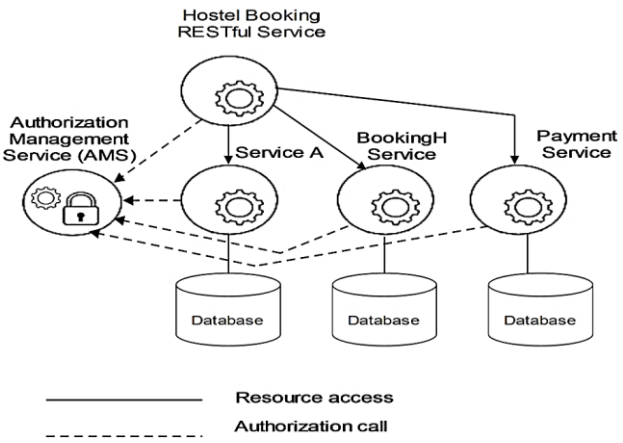

Figure 1. Composed HRBRS RESTful service framework architecture

4. Definition of a description language for description of RESTful service resources

As a language for description of REST service is used JavaScript Object Notation (JSON). Through this language are described all the elements of service by detailing each characteristic that includes the service. The service is composed and in itself includes Service A, BookingH Service and Payment

Service which are above emphasized. For elaboration of the description of service through JSON [8], as a case study is getting BookingH Service and it is described in details as follows. Such as this service can be described even other services of the REST composed service.

BookingH Service is described in JSON and consists from key/value pairs putting in double quotes where on one side is the key pair, and on the other side is its value presented as a string type:

```
{
 "name": "BookingService",
 "endpoint": "http://www.example.com"
…
}
```

In this case “name” and “endpoint” are keys, while “BookingH Service” and url “http://www.example.com” are values presented as strings.

Hereby is included even the list of operations which consist from some key/value pairs and arrays:

```
{
…
 "operations": [
     {
       "method": "GET",
       "url": "/bookings",
       "description": "Return the      list of bookings",
       "filters": [
         {
          "name": "date",
          "type": "date",
          "format": "dd-mm-yyyy"
         },
         {
          "name": "guestName",
          "type": "string",
          "minLength": 5,
         }
       ]
     }, {
       "method": "PUT",
       "url": "/bookings/{id}",
       "description": "Update an existing booking",
       "parameters": [
         {
            "name": "id",
            "type": "integer",
         }
       ],
     }, {
       "method": "POST",
       "url": "/bookings",
     }
  ]
…
```

In this part as key/value pairs are as follows:

```
…
 “method” : “GET”,
 “url” : “/bookings”,
 “description” : “Return the list of bookings”
…
```

The values of JSON code part for this service are string type. In the part of the above code is included even a list of filters identified as a key/values pairs of string, number and date types. This part is given as follows:

```
…
 "filters": [
         {
          "name": "date",
          "type": "date",
          "format": "dd-mm- yyyy",
          },
          {
           "name": "guestName",
           "type": "string",
           "minLength": 5,
           }
          ]
…
```

Also as parts of this code in JSON are even included two other lists with operations and parameters consisted by key/value pairs as follows:

```
…
{
 "method": "PUT",
 "url": "/bookings/{id}",
 "description": "Update an existing booking",
 "parameters": [
           {
            "name": "id",
            "type": "integer",
           }
          ],
          }, {
             "method": "POST",
             "url": "/bookings",
            }
…
```

In this part we encounter key (name)/value pairs:

```
…
 "method": "PUT",
 "url": "/bookings/{id}",
 "description": "Update an existing booking"
…
```

"method", "url" and "description" are keys (names), while "PUT", "/bookings/{id} and "Update an existing booking" are values of string type.

Also, in the above part is given a list of parameters in JSON. It contains "name": "id" and "type": "integer" as key/value pairs. "id" and "integer" values are number and string types.

In the end part of BookingH Service JSON code are included even key/values pairs as follows:

```
...
  {
   "method": "POST",
   "url": "/bookings"
  }
...
```

In this part keys are "method" and "url", while "POST" and "/bookings" are values of string type.

5. Authorization management

HTTP offers integration techniques for authentication of users, while authorization mechanism (coordinator) enables dependency underlying resources. By deciding of the authorization in functionality must be detailed characteristics in-depth for entities. Through them we can constrain functionality side for environments that are based on REST. In the coordinated authorization architecture protected stability, it increases portability and interoperability between services and reduces complexity. However, if we add resource-aware permission in the authorization mechanism, we can increase benefits that enable access to permission rules not only in resource mapping, but even in defining entity structure. This approach can be influenced in the authorization workflow in order to clarify any constraints in relation to dependency of different resources. Some issues should be respected in integration design and transparent authorization architecture by respecting REST with support of permissions in the structural level.

In our approach is presented an integrative system that satisfied constraint as follows:

- **It should respect different representations for underlying of resource.** Since there is variability of entities things, this point is crucial and should be respected conditionally without influence in reducing any functionality.
- **Different authorization permissions should be applied in resources.** Permissions should be referred in the corresponding URI, also in noticed operations from HTTP verbs which can be reflected by disjunct permission-sets. Additional data about resources and techniques for modifications should be registered in different permissions.
- **It must enable extensibility of REST.** If this property is respected, then we enable easy adaption of architecture depending of integration of new operations, new clients/services or resources, also in expanding optional levels of resources (content types). Each adaption that happens in the architecture may be constant and it should "obey" the coordinated architecture.

Authorization mechanism consists of a modular structure that acts as a layer; it handles and manages authorization requests. This is achieved by replacing the usual approach of resources with a mechanism for authorization. In the authorization mechanism registers resources, so there is the provider of the resource and clients/services model who include also a specific client/service. It enables the inclusion of links to forwarding of requests or directly save the content on its structure. Mechanism rules are enabled by defining the paths of resources.

This enables to cover the connection of resources with operations based on HTTP access rules. The number of combinations is not limited to connecting a specific group of rules for services. Each rule in the mechanism makes possible the support of "resource aware-filtering" depending of HTTP verb demand or entity characteristics.

The authorization mechanism provides content integration, filters and parameters which result in views, but also in finer-granular permissions of the requested resource. Filters must be notified and fulfilled with information in-depth for characteristics of entity and requests, also the responses should act on finer level as much as possible than the direct allocation of resources over an URI.

The authorization mechanism includes a structure for authorization and a large number of resources. In the structure for authorization is included through different rules mapping via HTTP functionalities, resources and resource-aware filters. Resources also are stored directly in the mechanism. We should note that each request is evaluated from the authorization structure before it interacts to specific resource(s). Since the rules mapped on a specific operation of HTTP relate to a unique URI, each authorization performs an operation on a defined resource, which is referenced only once in mechanism. The structure presents centralized mechanism regarding rules, parameters, and permissions and filtering.

Based on these, three issues provide four aspects for this structure:

1. Registered URI can be protected through many rules. Based on possibility operations (GET, PUT, PATCH and DELETE) of an URI, this aspect secures variability of different authorization groups.
2. Through a specific operation each rule is mapped. Operations are derived by an available HTTP verbs set. In this way is provided a unique REST awareness for any rule in the authorization mechanism.
3. Additional resource-aware filtering can be added and provided except authorization on URI and REST level. Also, if the filtering is independent from data that are retrieved there, it should be "aware" of the characteristics of data.
4. The resource can be referenced over links or stored in the authorization mechanism. In both cases, the content related to the resource regardless of mapping rules and permissions [4].

Based on the above given characteristics, in the figure 2 is given a framework that tells how the workflow of service can continue with permissions defined by operations (HTTP verbs) that consisted by filters and parameters. In this case a service for using resources of the other service must have permission and respect the putting rules of other service in order to access, utilize or manipulate the resources of a specific service. If the service is removed from this framework, then the operations together with filters, parameters and permission are invalid. Service consists of name and endpoint attributes that are string type. Operation contains from the method, URL and description attributes which are string types. Filter has name, type, format and minLength attributes where name and format are string type, while type is date type and minLength is integer type. Parameter contains name and type attributes, where name is string type and type is integer. Services should have operations, which include one or more filters and parameters. Operation executes one or more permissions, where through these enables the service to use and manipulate the resources of other service.

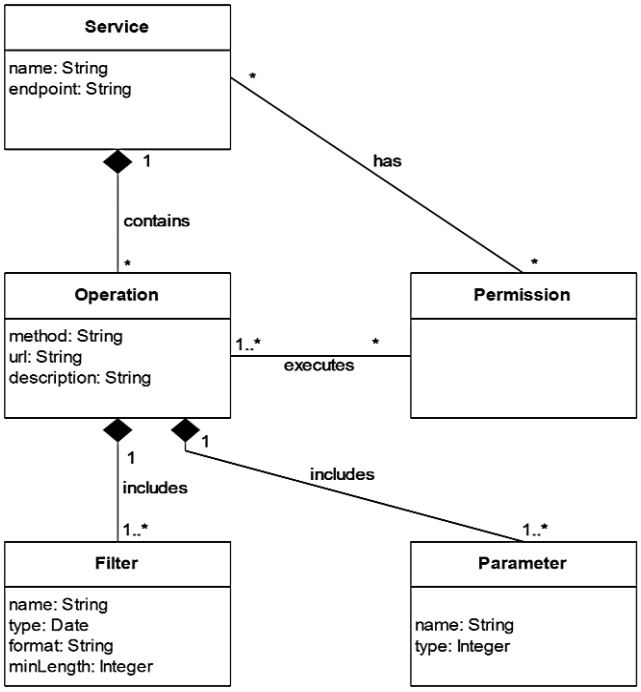

Figure 2. Service authorization framework based on permission

6. Case study – Hostel Rooms Booking REST service

6.1. Resource identification

The resource identification is made through names. On the top of this model are services which enable booking and interaction to HRBR service. Services are identified with their names and then they are classified as resources. Name also enables identification of a set of resources (e.g., bookings, rooms, payments).

6.2. Resource presentation

REST service supports many formats for presentation of resources such as HTML, XML and JSON. Choosing formats depends from the REST audience. When a RESTful service is used within a company, it can support JSON format, while a public REST API can support XML and JSON formats [6]. In our scenario is used JSON format for presentation of resources and operations of REST service.

6.3. REST interfaces

REST architecture is defined from four properties as follows.

6.3.1.Addressing

Each part of data (information) which is connected to service should be indicated as a resource, while resource(s) should be addressed through URI(s) [9]. A GET method for a collection of defined resources returns a list of resources that it contains, if any. Also GET method for a specific resource returns a resource if it exists. According to REST style is requested that all the defined resources are to be addressed [6]. In our scenario we are getting in consideration addressing and each defined resource is achieved from specific service(s) by at least a path of navigation of one or more links.

6.3.2. Connectivity

Through this feature is required that a resource to have links with other resources as interconnections [9]. Links between resources connect defined resources and provide connection between resources in our case study. These connections are presented as URLs. For services that are fully in-appropriate to the RESTful architectural style, the resource presentation must contain a list of links that can be accessed further in resources of REST service [6].

HRBR service enables booking, payment and cancellation of rooms. If the booking is not paid within twelve hours, it is cancelled from the system. If the third part of the service is not responded to information for payment within two hours, our service will cancel the payment. The confirmed booking, without being paid can be cancelled and the cancelled booking is deleted from the system after one month. The booking resource can be retrieved with GET, created with POST, updated with PUT and deleted with DELETE. The cancelled resource can be updated partially with PATCH. The same occurs even with operations of service for booking payment. Payment can be accepted if "interconnection" between room(s) and booking(s) exists.

6.3.3. Uniform interface

Resources in REST services are manipulated using HTTP standard methods (GET, POST, PUT, PATCH and DELETE) which they use to return/retrieve data from a resource or to update/change their state [9].

6.3.4. Statelessness

Each request that comes from service should contain all data that are required in process, i.e. server is not responsible to contain all details (information). From this we understand that each request is handled independently. REST services with these features can cooperate better with the existing devices and infrastructure of the system.

It should not hide session or state information (data), regardless to the effects of the GET, POST, PUT, PATCH and DELETE operations [9].

6.4. HTTP methods

For accessing the service resources are used standard HTTP methods (GET, POST, PUT and DELETE) [6], [9] where through their properties a uniform interface is enabled. For example, GET method can be called in any resource to return its specific state or information.

6.4.1. GET method

Through GET method [6] are returned/retrieved resource(s) of our RESTful service, for example: GET/payments/{id}/bookings is called for returning a payment of booking. Whenever GET method is called in a resource, it returns a response. In practice, the resource access for our scenario can be restricted through authorization.

6.4.2. POST and PUT methods

These methods are called based in defined criteria for REST. When a POST method is called in an existing URI, the resource is created, while if PUT is called in an existing URI, then it modifies an existing resource. PUT creates a new resource only when it is called in a new URI. POST method in generally is used for creating subordinate resources, so there are resources in relation to other parent resource. This understands that call of POST method always will create a new resource with a new address and same properties [9].

In the scenario, POST method is used to create a new resource. For example, with POST we create a new resource at the collection of bookings, so we make a booking. URI of new booking resource returns as a part of response. PUT method is idempotent, so it has a same effect even when it is called one or more time. If the PUT is called more times in a resource, it creates a new resource for the first time and can update the same resource with the same address in the consequent calls [6], [9].

For example, PUT is called to update a resource as PUT/bookings/{id}. If the service (in the client role) should modify that, it calls PUT again for the booking a resource with same URI but with different parameter values.

6.4.3. DELETE method

DELETE method enables removing of the resource. This method also is idempotent, so by calling of DELETE method in a resource will have a same effect even if it is called many times [6], [9]. We can delete a booking resource by calling DELETE: DELETE /bookings/{bid}. In our scenario, the cancelled booking can be deleted from system if it is cancelled before one month.

6.5. Endpoint identification

REST resources are identified by using URI endpoints. REST services well-designed should have endpoints that are understanding and easy to use. In the case study of our REST service are used best practices and conventions for designing of endpoints. As a first convention is considered uses a URI base for our REST service. URI base allows entry point to access in the REST service. REST API offers using of subdomains such as http://api.domain.com or http://dev.domain.com as a URI base. By creating separated subdomains, we protect every possibility of names entanglement of service. This also enables to strength the security policies [6]. For our scenario as a URI base is using http://www.example.com.

The second convention [6] is that endpoints' names of resources are used in the plural. For example, http://www.example.com/bookings is used to access a collection of booking resources. Individual resources for booking, we can access using URI such as http://www.example.com/bookings/1234. We can generalize the booking resources access using URI template http://www.example.com/bookings/{id}.

Third convention [6] is using of URI hierarchy for presentation of resources that are connecting with each other. In HRBR service, each resource of the payment is connecting to the booking resource. For this case we can do payments for booking according to a hierarchy of endpoints as follows http://www.example.com/payments/{id}/bookings, which is recommended to manipulate with all payments that are associated to certain booking of a room. In the same way the endpoint http://www.example.com/rooms/{id}/bookings enables to manipulate with specific room booking.

6.6. Requests identification

HTTP methods (verbs) allow services to interact and access resources by using endpoints. In our service we can complete one or more CRUD (Create Read Update Delete) operations [6] in resources like booking or payment of the hostel room(s). Through the GET operation, in a collection of resources returns/retrieves all the possible bookings. With POST operations is created a new room booking or a payment in the hostel.

PUT operation is used to update and modify a specific booking, before pay for a specific room which we want to book. Through PATCH operation we can make partial update of a booking resource. DELETE operation removes a room booking if it is not confirmed within twelve months after the registration is done, or after a month when it is considered "as an old registration" and the system also can remove it automatically.

Table 1. Requests and their description

HTTP method	URL	Description
GET	/bookings	It returns all possible bookings.
POST	/bookings	It creates a new booking.
PUT	/bookings/{id}	It updates a booking.
PATCH	/bookings/{id}	It partial updates a booking.
DELETE	/bookings/{id}	It deletes a booking.
GET	/rooms	It returns all possible rooms.
POST	/rooms/{id}/bookings	It creates a room booking.
PUT	/rooms/{id}/bookings	It updates a room booking.
DELETE	/rooms/{id}/bookings	It deletes a room booking.
GET	/payments	It returns bookings' payments.
POST	/payments/{id}/bookings	It makes a booking payment.
PUT	/payments/{id}/bookings	It updates a booking payment.

6.7. HTTP requests and responses

HTTP requests are called in for the different resources of REST services. When a service calls a HTTP method in other service, it is returned as a HTTP response that holds the state code and other information (if any). When a HTTP request is making in a resource, its response is sent and presented as a resource in that state as it was sent (if its content exists). Resource presentation consists from attributes and hyperlinks of resource which can be navigated further. Request parameters with HTTP request creates HTTP methods' parameters. HTTP responses contain the response code and call presentation of resource including attributes that are defined in the resource [9]. For elaboration of this case, some examples are getting for POST, PUT, PATCH and DELETE requests for a resource booking. POST request is called in the booking resource with request parameter guestName. HTTP response contains JSON presentation for created resource and information for links resource(s). Each resource can be addressed independently and associated to roles' names. The role names are given for navigation path of resources. So, each resource has an URI address and in this way offers features for resources of the service.

Table 2. Some examples with HTTP requests and responses for HRB REST service

Request	Response
POST/bookings HTTP/1.1 Host: www.example.com Content-type: json {"guestName": "Agim", "roomType": "double"}	HTTP/1.1 201 Created Location: http://www.example.com/bookings/11 Content-type: json {"id": 11, "date": "12-08-2015", "guestName": "Agim", "roomType": "double", "confirmation": false}
PUT/bookings/11 HTTP/1.1 Host: www.example.com Content-type: json {"id": 11, "date": "12-08-2015", "guestName": "Artan"}	HTTP/1.1 200 OK Location: http://www.example.com/bookings/11 Content-type: json {"id": 11, "date": "12-08-2015", "guestName": "Artan", "confirmation": false }
PATCH/bookings/11/payment HTTP/1.1 Host: www.example.com Content-type: json {"confirmation": true}	HTTP/1.1 200 OK Location: http://www.example.com/bookings/11/payment Content-type: json {"id": 11, "date": "12-08-2015", "guestName": "Artan", "confirmation": true }
PATCH/bookings/11 HTTP/1.1 Host: www.example.com Content-type: json {"note": "not traveling", "canceled": true}	HTTP/1.1 200 OK Location: http://www.example.com/bookings/11 Content-type: json {"id": 11, "date": "15-08-2015", "guestName": "Artan", "canceled": true, "note": "not traveling"}
DELETE/bookings/11 HTTP/1.1 Host: www.example.com Content-type: json	HTTP/1.1 200 OK Content-type: json {"success": true}

6.8. Description language and implementation of HRB RESTful service scenario using JSON

HRB REST service is composed and as above mentioned it contains three services: Service A, BookingH Service and Payment Service. The purpose of this part is to describe each of the emphasized services with their constituent elements in JSON.

BookingH Service is defined and described in JSON. Initially it is given name of service and endpoint followed by operations. For returning a certain resource, in this case a list of bookings, GET method with its URL and description is used. Data in this part are filtered by date and name of the service. All of these are done through key (name)/value pairs, where each name is followed by its value. Also to create a new booking in this service a POST method with respective URL and description is used. Even in this part each key is followed by its value. Although every created resource sometimes should be updated, for this aim is used PUT method in the BookingH service. It is followed by URL of resource that will

update and its description. Inside of this are specified parameters in JSON code.

Sometimes it is needed to do partial update of resource, respectively for booking in the BookingH service. For this case is used PATCH method followed by URL and its description. There are included parameters with which resource should be updated partially. Even this part is shown in below JSON code. The resource, concretely booking if it is not confirmed by specific service should be deleted. Other case for removing a resource is its seniority. For this case is used the DELETE method followed by URL and description. All above mentioned cases are given in below JSON code.

```
{
 “name”:”BookingService”,
 “endpoint”: "http://www.example.com",
 “operations”: [
{
 “method”: “GET”,
 “url”: “/bookings”,
 “description”: “Return the list of
bookings”,
“filters”: [
                {
          “name”: “date”,
          “type”: “date”,
          “format: "dd-mm-yyyy",
          },{
          “name”: “guestName”,
          “type”: “string”,
          “minLength”: 10
         }
        ]
},{
  “method”: “POST”,
  “url”: /bookings”
  “description”: Create the booking”
},{
  “method”: “PUT”,
  “url”: “/bookings/{id}”,
  “description”: “Update an existing  booking”,
  “parameters”:[
             {
              “name”: “id”,
              “type”: “integer”
             }
            ],
{
 “method”: “PATCH”,
 “url”: “/bookings/{id}”,
 “description”: “Partial update an existing booking”,
 “parameters”:
  [{
    “note”: “not traveling”,
    “canceled”: true
   },{
     “method”: “DELETE”
     “url”: “/bookings/{id}”,
     “description”: “Delete an existing  booking”
     }
  ]
}
```

In JSON is defined and described also the REST Service A in below code. Initially it is named, and then is given its endpoint followed by operations. For returning a list of rooms through this service GET method followed by its URL and description is used. Rooms that are required for booking are filtered by date, room type and guest name. Each part is given through key/value pairs where each key is followed by its value e.g. “name”: “guestName”.

To create a new resource, respectively new room booking is used POST method followed by its URL and description. Even in this part each key is followed by its value.

Sometimes each created resource should be updated. Through Service A, this is enabled by PUT method followed by URL and description. Resource is updated according to the parameters given in following code in JSON.

In the case of not confirmation of room booking in the HRB RESTful service has cases when resource, respectively the room booking should be removed. For this case is used DELETE method followed by its URL and description. All cases are given in following.

```
{
 “name”: “ServiceA”,
 "endpoint": "http://www.example.com",
 “operations”: [
 “method”: “GET”,
 “url”: “/rooms”,
 “description”: “Return the list of     rooms”
 “filters”:[
         {
          “name”: “date”,
          “type”:”date”,
          “format”: “dd-mm-yyyy”
          },{
            “name”: “roomType”,
            “type”: “enum”,
            “values”:            [”single”,”double”]
            },{
              “name”: “guestName”
              “type”: “string”
              “minLength”: 15
              },
{
 “method”: “POST”,
  “url”:/rooms/{id}/bookings
  “description”: “Create a new room  booking”
},{
  “method”: “PUT”,
```

```
  "url": "/rooms/{id}/bookings",
  "description": "Update an existing room booking"
  "parameters":[
            {
             "name": "id",
             "type": "integer"
             },
   {
    "method": "DELETE",
    "url": /rooms/{id}/bookings",
    "description": "Delete an existing    room booking"
   }
 ]
}
```

In this composed REST service are found even the REST Payment Service. Through JSON is defined and described this service with methods, filters and parameters. Initially there is given its name, endpoint and operations. For returning a list of payments by service the GET method is used. Payments are filtered by date and guest name. In this part each key is followed by its value.

For making the booking payment in this service is used the POST method followed by URL and description. Even in this part each key is followed by its value.

There are cases when the payment should be updated. For this case is used the PUT method with below defined parameters.

In cases when the payment is not made, then follows remove of it for specific booking. This is done by DELETE method. All cases are given in below JSON code.

```
{
 "name": "PaymentService",
 "endpoint": "http://www.example.com",
 "operations": [
{
 "method": "GET",
 "url": /payments",
 "description": "Return the list of  payments",
 "filters":[
         {
          "name": "date"
          "type": "date",
          "format": "dd-mm-yyyy"
         },
         {
          "name": "guestName",
          "type": "string",
          "minLength": 15
         }
        ]
},{
  "method": "POST",
  "url": "/payments",
  "description": "Make a booking  payment",
},{
  "method": "PUT",
  "url": "/payments/{id}",
  "description": "Update an existing payment booking",
  "parameters":[
            {
             "name": "id",
             "type": "integer",
            }
           ],
},{
  "method": "DELETE",
  "url": "/payments/{id}",
  "description": "Delete an existing room booking"
 }
 ]
}
```

7. Results and discussion

The results of this research are summarized in these points:

- **Declarative language.** Hereby is defined declarative language for describing interface of our REST service in the scenario (case study) emphasized in the point 4. There is clear explanation defining of resources of REST services using JSON and their implementation based on this language by respecting REST best practices.
- **Services are described as a possible list of HTTP requests.** This part is fulfilled and explained entirely, which results with description of HTTP requests using HTTP methods (operations) mostly used like GET, POST, PUT, PATCH and DELETE. Through these operations are enabled requests creation and they are sent in the specific endpoints for REST service, respectively for above mentioned scenario of this service that elaborates exactly this part.
- **Filtering of resources using parameters.** This part is considered as one of main parts of this paper. It is realized through JSON using different parameters which can be enabled by filtering of data in the authorization mechanism. This has resulted in the access restriction of resources for non-authorization services, respectively for these services that do not fulfill required parameters of service which they want to access and use its resources.
- **The defined language for description of the services' operations.** There are more languages for description of the service

operations, where through of these can describe even REST services. In this aspect as appropriate is considered JSON, therefore it is used along the whole paper for description of REST service resources.

- **Creating of permissions based in operations and their parameters.** As a key part are considered even permissions. They are based in operations and their parameters. Specific services can communicate between them only by respecting permissions that are decided in the authorization mechanism (the specific service) in order to access, use and manipulate to resources of each other.
- **Different permissions are applied in a resource (service).** By combining operations and their parameters are created different permissions which are applied in services. These permissions are referred to corresponding URIs, while operations are marked from HTTP verbs and they can reflect in disjunct permissions (permission-sets).
- **Authorization mechanism.** This part is presented as a modular structure which acts as a layer and handles authorization requests. This is achieved by replacing the usual approach of resources to a mechanism for authorization.

8. Conclusion

The application of REST services in institutions and enterprises services improves implementation and maintenance of services (applications). The interaction of services should be coordinated. Consequently the coordination model with existing services reduces dependence of services and dismantles authorization. The main goal of this paper was to define and describe usage of REST service resources for authorization purposes - to effectively manage and secure permission and authorization of services; description of REST operations with their parameters and filters for authorization purposes, creating of permissions, permission authorization and control. By integrating all services in one in-house offers stable and flexible services. Meanwhile, by delegating permissions we can achieve better reusability of services, reduce administration time, simplicity service creation and the management can be centralized.

Certainly, this research work allows space for further advancement. The whole aim of this work has been to define and describe permissions for access to and manipulation of service resources by enabling restriction in their access through the authorization mechanism (the authorization service), which is based on defined permissions and authorizes the service to access and manipulate resources of the specific services. Concretely, it must have permission and should respect the defined rules of other service that utilize and manipulate with resources of service. In this aspect, this work can be further continued in the client-service communication, respectively in the part of permission managing through the authorization mechanism.

References

[1]. Fielding, T. R. (2000). *Architectural Styles and Design Network-Based Software Architectures*. University of California, USA. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[2]. Lewis, G. (2010). *Getting Started with Service-Oriented Architecture (SOA) Terminology.* Software Engineering Institute.

[3]. Anderson, R. (2001). *A Guide to Building Dependable Distributed Systems*. Wiely.

[4]. Beshiri, A. (2015). *Analyzing of REST services: Integration Services and Authorization Management.* Sixth International Conference for Information Systems and Technology Innovation: Inducing Modern Business Solutions. Tirana, Albania.

[5]. Memeti, A., Selimi, B., Besimi, A., & Çiço, B. (2015). *A Framework for Flexible REST Services: Decoupling Authorization for Reduced Service Dependency.* Forth Mediterranean Conference on Embedded Computing (MECO). Budva, Montenegro.

[6]. Varanasi, B., & Belida, S. (2015). *Spring REST-REST and Web Services Development using Spring.* Apress.

[7]. Extensible Markup Language (XML). http://www.w3.org/XML

[8]. JSON. http://www.json.org/

[9]. Rauf, I. (2014). Design and Validation of Stateful Composition RESTful Web Services. Abo Akademi University. http://tucs.fi/publications/attachment.php?fname=phdRauf_Irum14a.full.pdf. Finland.

[10]. Brachman, E., Dittman, G., & Schubert, K.D. (2011). Simplified Authentication and Authorization for RESTful Services in Trusted Environments. IBM Systems and Technology Group.