The *-PaaS API specification

Version 1.5.1

Telecom SudParis, Computer Science Department

Table of contents

A. Introduction

This document provides a description of the *-PaaS API based on REST/XML. This API is designed to provide an abstraction layer and a middleware for existing PaaS solutions to manage applications and environments in a generic fashion (Figure 1). To define a new connection between a novel PaaS and developer /application, one has simply to add its specific implementation.

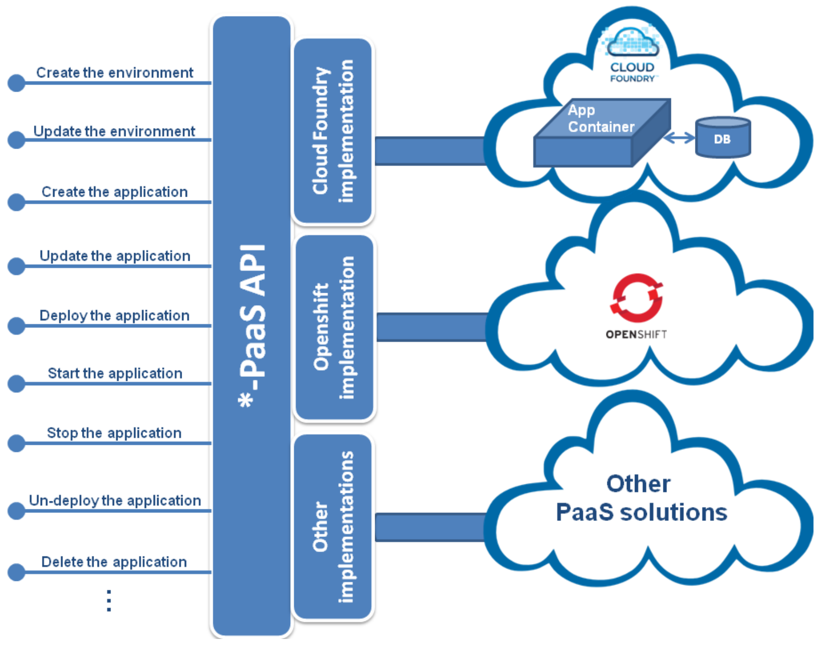

Figure 1 : Overview

Two *-PaaS API implementations are available (V 0.1): a Cloud Foundry implementation (CF-PaaS) and an OpenShift implementation (OS-PaaS). Both implementations are available at: http://gitorious.ow2.org/ow2-compatibleone/coaps/ and a user guide for CF-PaaS is available at: http://www-inf.it-sudparis.eu/~sellami/starPaaS/PaaSAPI-UserGuide.pdf.[1]

B. Overview on the *-PaaS API

*-PaaS API exposes two main resources: *RestApplicationManager and RestEnvironmentManager*. *RestApplicationManager* offers PaaS application management methods (See Figure 2). By "application" we mean any computer software or program that can be deployed over a PaaS. Application source archives should be provided by the developer in a bundled format (*i.e.* war, ear, zip, etc.) or extracted format (*i.e.* a local folder with the different files and dependencies, distant URL, etc.).

[1] Both implementations and the user guide use an old version of the specification and some inconsistencies with the current specification can be found.

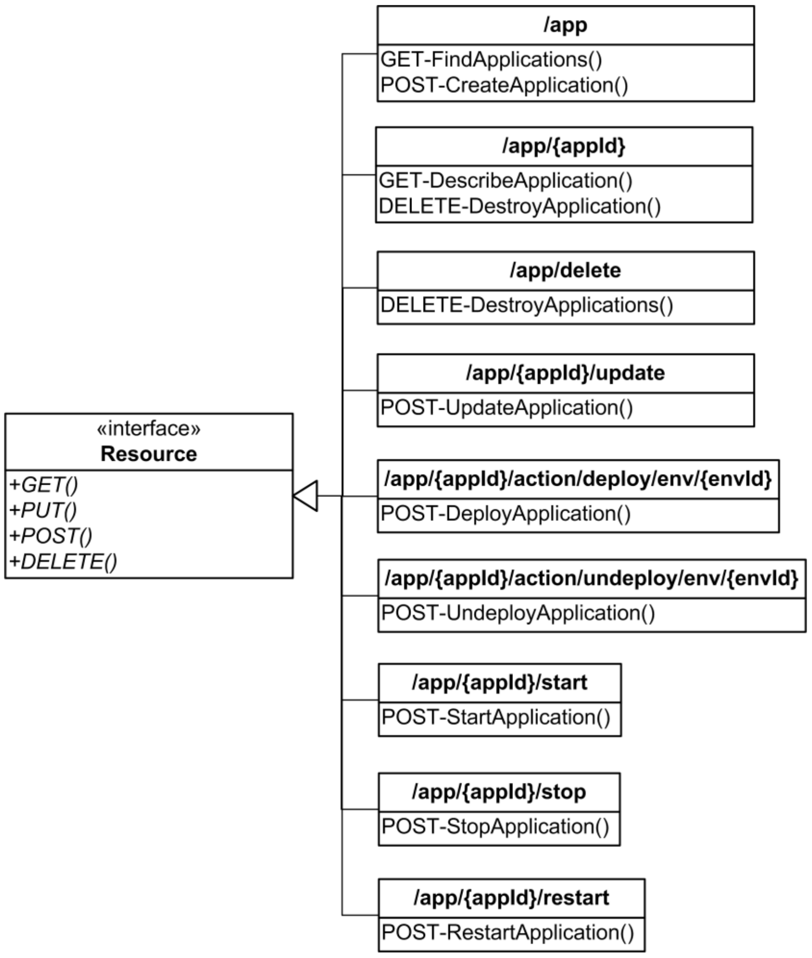

Figure 2 RestApplicationManager

RestEnvironmentManager offers PaaS environment management methods (See Figure 3). A PaaS environment represents a set of ‘’settings’’ needed to host and run an application in this PaaS: *i.e.* the needed runtime (java 7, java 6, ruby, etc.), the needed frameworks/containers (spring, tomcat, ruby, etc.) and eventually needed services (databases, messaging, etc.).

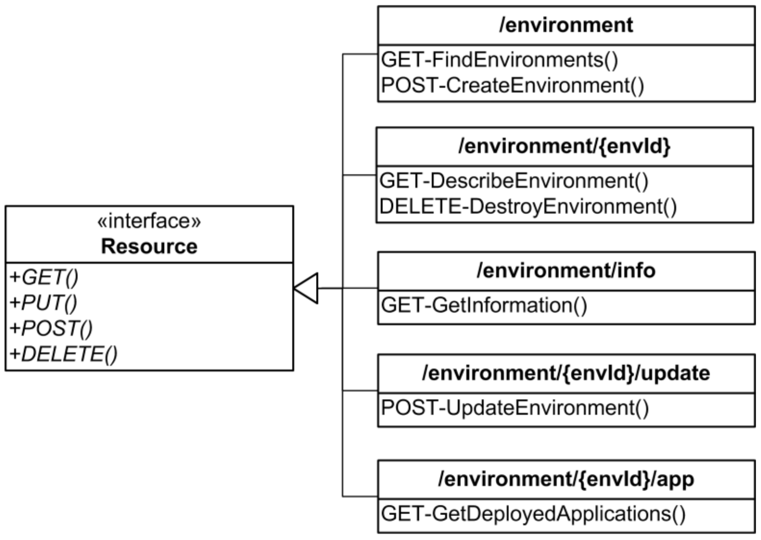

Figure 3 RestEnvironmentManager

To deploy an application and run it through *-PaaS API, one should follow the basic usage scenario illustrated in Figure 4.

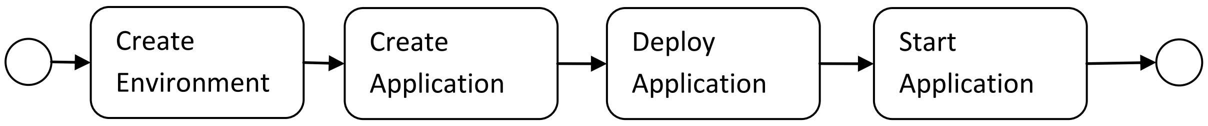

Figure 4 an application deployment scenario

C. The Environment Manager Resource

In this section, we introduce the environment manager resource, its different child resources and their associated methods. We start this section by providing examples of an environment manifest (required as input by some of the REST methods of the environment manager resource) and of an environment description (returned as response by some of the REST methods of the environment manager resource).

Environment manifest

The *createEnvironment* and *updateEnvironment* operations require as input an environment manifest. This manifest allows developers to specify the different characteristics of the application's environment using an environment template (see Figure 5). Each environment template is composed by a set of PaaS resource nodes and PaaS relations to link these nodes. PaaS nodes can be *container* type, *database* type or *router* type while relations between them can be a binding between a container node and a database node or between two containers node through a router node for example.

> ***Note***: *the environment manifest used in this document is given as an example. While implementing our API you can specify your own manifests.*

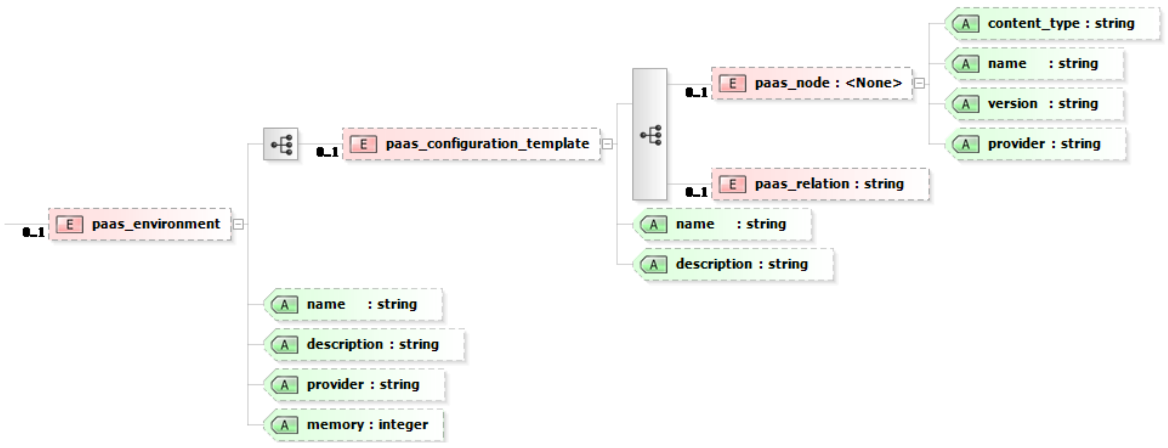

Figure 5 schema of a possible environment manifest

Environment description

Some of the environment management operations return as response an XML environment descriptor. The XML format of this descriptor is specific for the *-PaaS API implementation. In the following, we provide as an example the XML schema for the environment descriptor specific to a CloudFoundry implementation of the *-PaaS API (see Figure 6).

> ***Note***: *the environment description used in this document is given as an example. While implementing our API you can specify your proper environment description.*

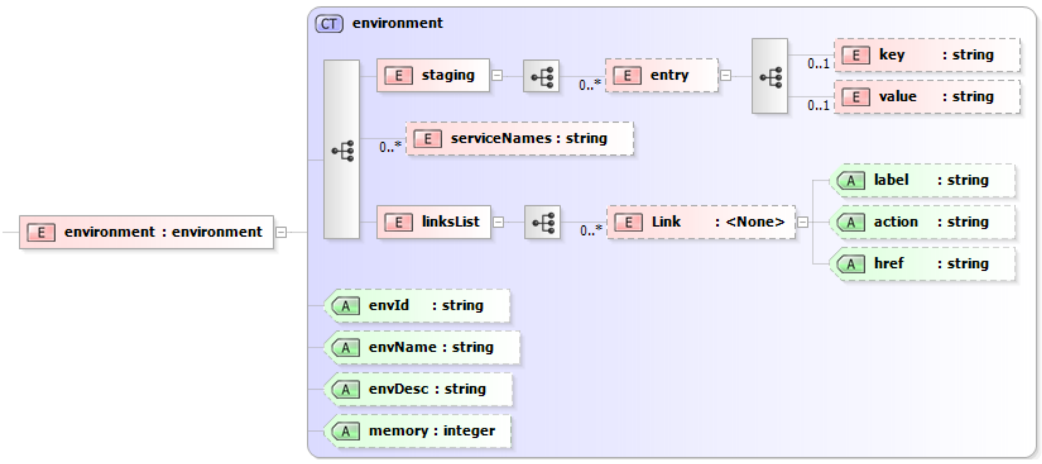

Figure 6 XML schema of a possible environment description

In Table 1, we provide the semantics of the different elements in the environment descriptor presented in Figure 6 and provide the corresponding element in the environment manifest presented in Figure 5.

Element of the environment descriptor	**Description**	**Corresponding element in the environment manifest**
E staging	This element describes the frameworks, environments and eventual commands offered by the environment.	The <paas_node> elements with *container* as content_type value.
E serviceNames	This element defines the services (database, messaging pool …) associated to the environment.	The <paas_node> elements with *database* as content_type for persistent values, *container* for hosting applications or *router* for formatted messages between Paas nodes
E linksList	The set of links associated to the environment (see Section F). These links are automatically generated.	--
A envId	An automatically generated identifier for the environment.	--
A envName	The environment's name.	A name defined in <paas_environment>
A envDesc	An optional textual description of the environment.	A description defined in <paas_environment>
A memory	The physical memory that is allocated to the application expressed in Megabytes.	A memory defined in <paas_environment>

Table 1 elements and attributes of the environment description

Environment management methods

- ***Create Environment***

This method creates a new environment using an environment descriptor. An environment specifies the needed frameworks, runtimes containers and/or services required by a given application.

REST method	POST
Resource identifier	/environment
Input parameter	An XML environment manifest (in the body of the request)
Response	An XML environment descriptor. This descriptor contains, among other information, the created environment's ID.
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

Example using CURL

Request

curl –X Post –d@EnvironmentDescriptor.xml[2] –H "Content-Type: application/xml" http://hostname:port/CF-api/environment[3]

Response

Return code: 200 OK

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<environment envId="1" envName="JavaWebEnv" envDesc="JavaWebApplicationsEnv">
 <staging>
  <entry>
   <key>command</key>
   <value>no</value>
  </entry>
  <entry>
   <key>runtime</key>
   <value>java</value>
  </entry>
  <entry>
   <key>framework</key>
   <value>java_web</value>
  </entry>
 </staging>
 <serviceNames>mysql</serviceNames>
 <linksList>
  <link label="destroyEnvironment()" action="DELETE" href="http://localhost:8080/CF-api/rest/environment/1"/>
  <link label="getEnvironment()" action="GET" href="http://localhost:8080/CF-api/rest/environment/1"/>
 </linksList>
</environment>
```

- ***Update Environment***

This method updates an existing environment. The environment ID must be provided (i.e. envId) and the updates has to be specified in the input parameter (i.e. as an environment Manifest)

REST method	POST
Resource identifier	/environment/{envId}/update
Input parameter	An XML environment manifest (in the body of the request)

[2] EnvironmentDescriptor.xml refers to the path of an XML manifest describing the environment to create.
[3] The hostname and the port number where the API is deployed. CF-api is the application path.

Response	The new XML environment descriptor.
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

Example using CURL

Request

```
   curl –X Post –d@EnvironmentDescriptor.xml –H “Content-Type: application/xml”
http://hostname:port/CF-api/environment/1/update
```

Response

```
   Return code:      200 OK

   Response:         The XML environment descriptor
```

- ***Destroy Environment***

This method destroys an environment given its ID.

REST method	DELETE
Resource identifier	/environment/{envId}
Input parameter	--
Response	The destroy discharge
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

- ***Find Environments***

This method lists the available environments.

REST method	GET
Resource identifier	/environment
Input parameter	--
Response	A list of XML environment descriptors of the existing environments
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

- ***Describe Environment***

This method returns the XML environment description of an environment given its ID.

REST method	GET
Resource identifier	/environment/{envId}
Input parameter	--
Response	The XML environment descriptor
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

- ***Get Deployed Applications***

This method lists the deployed application in an environment given its ID

REST method	GET
Resource identifier	/environment/{envId}/app
Input parameter	--
Response	A list of XML application descriptors
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

- ***Get Information***

This method lists the runtimes, frameworks and services supported by the PaaS.

REST method	GET
Resource identifier	/ environment/info
Input parameter	--
Response	The list of supported runtimes, frameworks and services
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

D. The Application Manager Resource

In this section, we introduce the application manager resource, its different child resources and their associated methods. We start this section by providing examples of an application manifest (required as input by some of the REST methods of the application manager resource) and of an application description (returned as response by some of the REST methods of the application manager resource).

Application manifest

createApplication and *updateApplication* operations requires as input an application manifest. This manifest allows developer providing information needed by the PaaS to manage its deployment and execution. It allows, among others, specifying the application name, its different versions with specific properties of each of them and a set of operational instances. It also allows specifying the type and the location of the source archives needed by the API at deployment time. An XML schema describing the various descriptive elements of the application manifest and their hierarchy is illustrated in Figure 7.

> ***Note**: the application manifest used in this document is given as an example. While implementing our API you can specify your own manifests.*

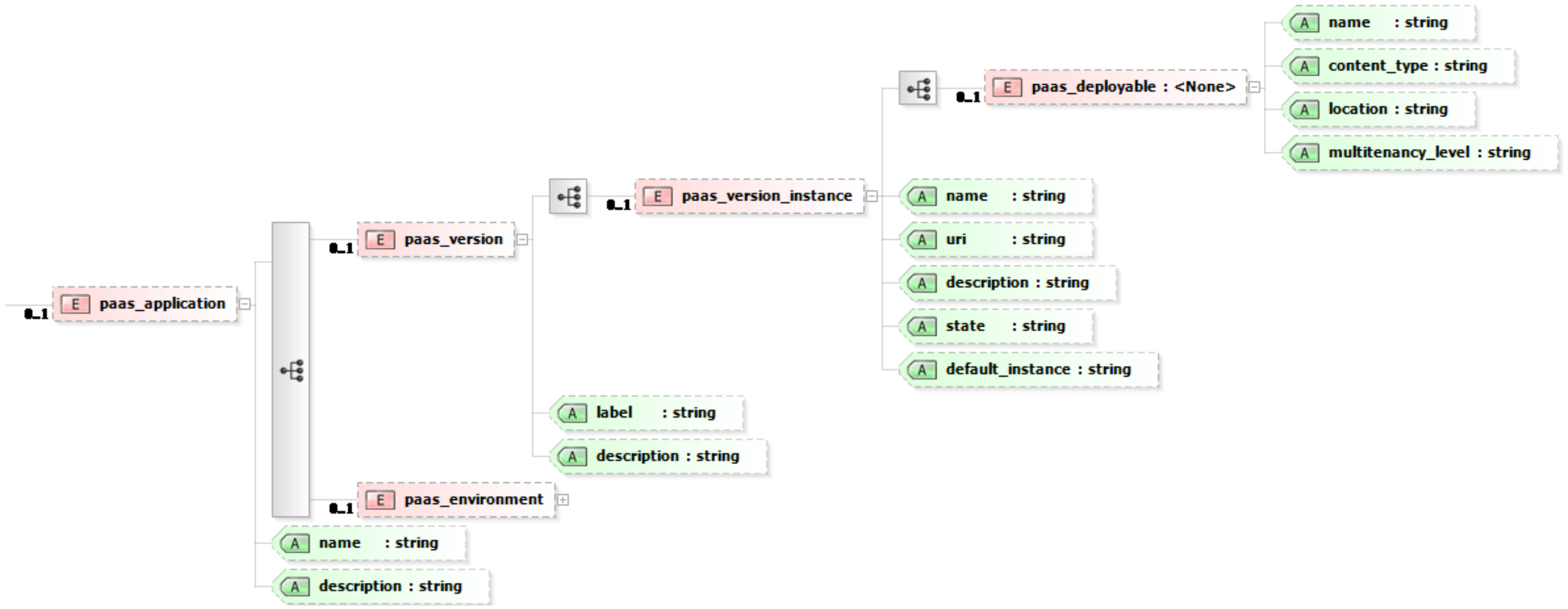

Figure 7 XML schema of an application Manifest

Application description

Some of the application management operations return as output an XML application descriptor. The XML format of this descriptor is specific and depends on the *-PaaS API implementation. In the following, we provide the XML schema for the application descriptor corresponding to the CloudFoundry API implementation (see Figure 8).

> ***Note**: the application description used in this document is given as an example. While implementing our API you can specify your proper application description.*

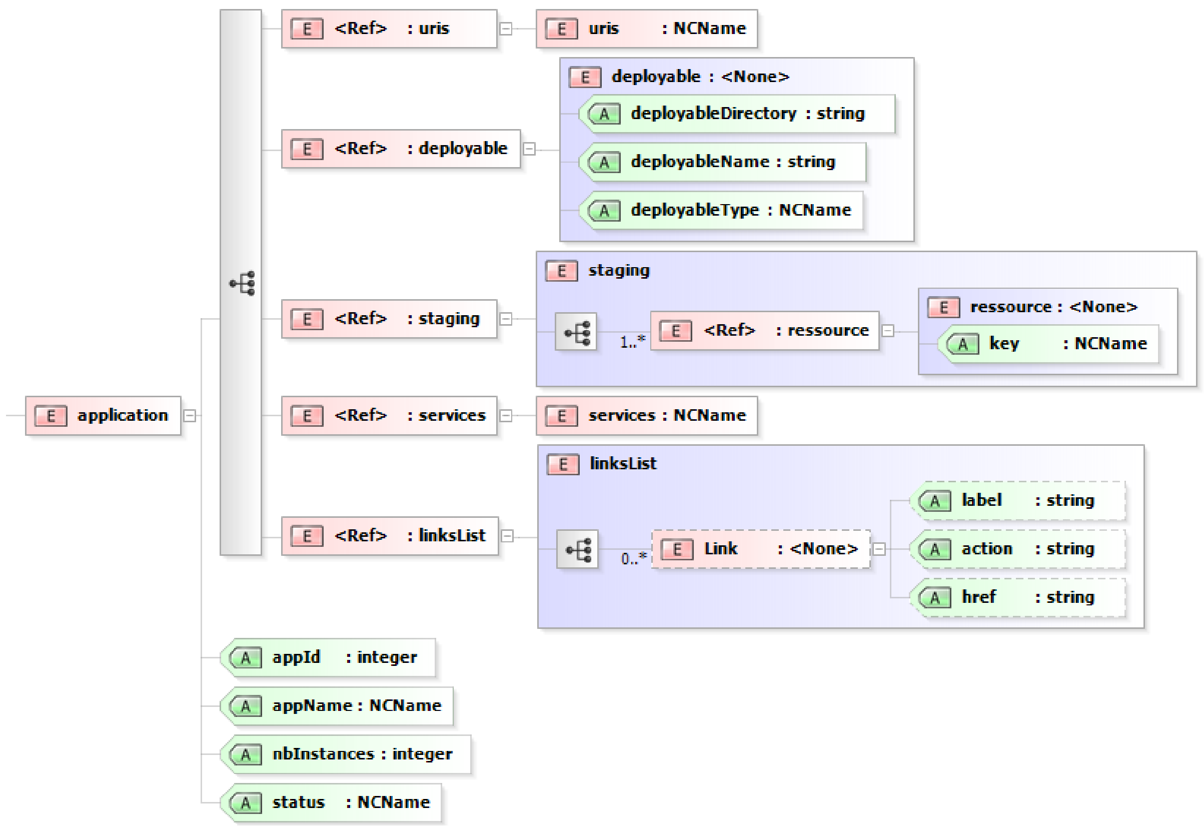

Figure 8 XML schema of a possible application description

In Table 2, we provide the semantics of the different elements (E) and attributes (A) in the application descriptor and provide the corresponding element in the application Manifest (See Figure 7 and Figure 8).

Element of the application descriptor	Description	Corresponding element in the application manifest
E uris	The URI of the deployed application. This URI is automatically generated using the provided application name and the PaaS URI.	--
E deployable	This element describes the application deployable (e.g. artifact, source files …).	E <paas_deployable>
E Staging	This element describes the runtime, framework and commands required by the application. This information is retrieved from the <paas_node> element in the environment manifest.	--

E services	This element describes the services (e.g. messaging, databases …) used by the application. This information is retrieved from the <paas_node> element in the environment manifest.	--
E linksList	The set of links associated to the application (see Section F). These links are automatically generated.	--
A appId	An automatically generated identifier for the application.	--
A appName	The application's name.	A name defined in <paas_application>
A nbInstances	The number of the application instances.	The number of <paas_version_instance> elements.
A status	This attribute indicates the status (STARTED/STOPPED) of the application. When an application is created, the default value is STOPPED.	--

Table 2 elements and attributes of the application description

Application management methods

- ***Create Application***

This method creates a new application using an application descriptor.

REST method	POST
Resource identifier	/app
Input parameter	An XML application manifest (in the body of the request)
Response	An XML application descriptor. This descriptor contains, among other information, the created application's ID.
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

Example using CURL

Request

```
    curl –X Post –d@ApplicationDescriptor.xml[4] –H "Content-Type: application/xml"
http://hostname:port/CF-api/app
```

Response

Return code: 200 OK

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns:ns2="ressources" appName="SampleApplication" appId="2" status="STOPPED" memory="512"
checkExists="true" nbInstances="1">
 <uris>SampleApplication.cloudfoundry.com</uris>
 <deployable deployableName="SampleServlet.war" deployableType="artifact"
deployableDirectory="APPLICATION_PATH"/>
 <versionInstances instanceName="Instance1"/>

 <linksList>
  <link label="describeApplication()"  action=" GET " href="http://localhost:8080/CF-api/rest/app/2"/>
```

[4] ApplicationDescriptor.xml refers to the path of an XML manifest describing the application to create.

```
  <link label="destroyApplication()"  action="DELETE" href="http://localhost:8080/CF-api/rest/ app/2/delete"/>
 </linksList>
</application>
```

- ***Update Application***

This method updates an existing application. The application ID must be provided (*i.e.* appId) and the updates has to be specified in the input parameter (*i.e.* as an application Manifest).

REST method	POST
Resource identifier	/app/{appId}/update
Input parameter	An XML application manifest (in the body of the request)
Response	The new XML application descriptor
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

Example using CURL

Request

```
   curl –X Post –d@ApplicationDescriptor.xml –H “Content-Type: application/xml”
http://hostname:port/CF-api/app/2/update
```

Response

```
   Return code:     200 OK

   Response:        The XML application descriptor
```

- ***Find Applications***

This method lists the available applications.

REST method	GET
Resource identifier	/app
Input parameter	--
Response	A list of XML application descriptors of the existing applications
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

Example using CURL

Request

```
   curl –X Get  http://hostname:port/CF-api/app
```

Response

```
   Return code:     200 OK

   Response:        The list of XML application descriptors
```

- ***Start Application***

This method starts a deployed application.

REST method	POST
Resource identifier	/app/{appId}/start
Input parameter	--
Response	The XML application descriptor with the value of the status attribute set to **STARTED**
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

- ***Stop Application***

This method stops a started application.

REST method	POST
Resource identifier	/app/{appId}/stop
Input parameter	--
Response	The XML application descriptor with the value of the status attribute set to S**TOPPED**
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

- ***Restart Application***

This method restarts a deployed application.

REST method	POST
Resource identifier	/app/{appId}/restart
Input parameter	--
Response	The XML application descriptor with the value of the status attribute set to **STARTED**
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

- ***Describe Application***

This method returns the XML application description for an application given its ID.

REST method	GET
Resource identifier	/app/{appId}
Input parameter	--
Response	The XML application descriptor
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

- ***Destroy Application***

This method deletes an application given its ID.

REST method	DELETE
Resource identifier	/app/{appId}
Input parameter	--
Response	The destroy discharge
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

Example using CURL

Request

```
curl –X Delete  http://hostname:port/CF-api/app/3
```

Response

Return code: 200 OK

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<operationResponse value="The application with ID 3 was successfully destroyed"/>
```

- ***Destroy Applications***

This method deletes all existing applications.

REST method	DELETE
Resource identifier	/app/delete
Input parameter	--
Response	The destroy discharge
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

- ***Deploy Application***

This method deploys an application identified by its ID (i.e. appId) on an existing environment also identified by its ID (i.e. envId). The application artifact to deploy must also be included.

REST method	POST
Resource identifier	/app/{appId}/action/deploy/env/{envId}
Input parameter	The application artifacts (as a file)
Response	An XML application descriptor
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

Example using CURL

Request

curl –X Post -F file[5]=@SampleApp.war http://hostname:port/CF-api/app/1/action/deploy/env/2

Response

Return code: 200 OK

Response: The application 2 was successfully deployed on the environment 1

- ***Undeploy Application***

This method un-deploys an application identified by its ID (i.e. appId) already deployed on an existing environment also identified by its ID (i.e. envId).

[5] "file" references the name of the file field name used by the deploy operation of the CF-PaaS API implementation.

REST method	POST
Resource identifier	/app/{appId}/action/undeploy/env/{envId}
Input parameter	--
Response	The un-deployment discharge
Status code	200 if OK the error code otherwise (see Section E for possible error codes)

E. Common errors

This section lists common errors, based on the HTTP status codes[6], which can be returned by the management operations.

	Error	**Description**	**HTTP Status Code**
Client errors	Bad Request	The request has a syntax error (invalid action, missing parameter …).	400
	Resource Not Found	The requested resource (environment or application) was not found.	404
	Method Not Allowed	The used REST action (i.e. GET, POST …) is not allowed on that resource.	405
Server errors	Internal Failure	The internal processing has failed due to some unexpected errors.	500
	Service Unavailable	The request has failed due to a temporary failure on the server	503
	Timeout exception	The server took long time to respond.	504

F. Link Elements

The *-PaaS API uses link elements to connect: (1) application objects to environment objects and (2) different management methods to application and environment objects. The aim of links is to ease the retrieval, by a human or software agent, of the information associated to an environment or an application object.

The structure of the link element is described in Figure 9.

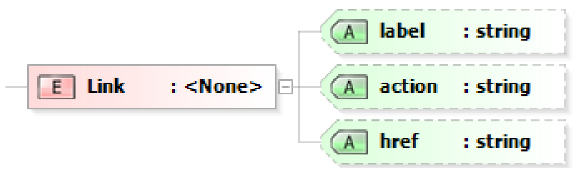

Figure 9 XML schema of the link element

In Table 3, we provide the semantics of the different attributes (A) of the link element.

6 Fielding, et al. HTTP/1.1, Internet RFC 2616, available at: http://www.ietf.org/rfc/rfc2616.txt.

Attribute	**Description**
Ⓐ label	The name of the management method (e.g. describeApplication(), destroyEnvironment(), etc.)
Ⓐ action	The associated REST action (e.g. GET, POST, etc.)
Ⓐ href	The URI, including the resource identifier, of the management method

Table 3 attributes of the link element

Annex A: Application and Environment Management Operations

The following Table provides a summary of the different Application and environment management operations and their associated REST method and resource identifiers.

Application management operations	
Operation	**Command**
Create Application	POST /app
Update Application	POST /app/{appId}/update
Find Applications	GET /app
Start Application	POST /app/{appId}/start
Stop Application	POST /app/{appId}/stop
Restart Application	POST /app/{appId}/restart
Describe Application	GET /app/{appId}
Destroy Application	DELETE /app/{appId}
Destroy Applications	DELETE /app/delete
Deploy Application	POST /app/{appId}/action/deploy/env/{envId}
Undeploy Application	POST /app/{appId}/action/undeploy/env/{envId}
Environment management operations	
Operation	**Command**
Create Environment	POST /environment
Update Environment	POST /environment/{envId}/update
Destroy Environment	DELETE /environment/{envId}
Find Environments	GET /environment
Describe Environment	GET /environment/{envId}
Get Deployed Applications	GET /environment/{envId}/app
Get information	GET /environment/info