OpenAPE

Stephan Unfried

February 26, 2017

Contents

1 Introduction

OpenAPE is a settings-based personalization framework to cover three important scenarios:

The bread and butter scenario is to **infer settings** for an application a user might not have met yet, for example using an Android phone for the first time with only using Windows PCs so far. In this use case, OpenAPE will try to find other users who have similar settings, but also have settings for the new target device or application. To do so, OpenAPE deploys various machine learning techniques, such as clustering existing users and creating virtual user profiles for these clusters. The fundamental assumption here is that users will adapt their device to their needs or preferences and thus settings applied by a users can be considered verified.

Quite similar to settings-adaptation scenario is **inferring content**. When coupled with a content server, OpenAPE can deliver content for webpages, such as images, that match a users settings, for example delivering a high-contrast image if respective settings are present for screen readers or magnifiers. There are, however, some subtle differences: it is hard to assess whether the content delivered was actually a good choice in the current situation. Further, content nowadays can range from simple images up to whole web apps, which in turn will invoke their own adaptation queries.

Finally, as a mix of the above use cases, OpenAPE could be used to **transfer rules and scripts to a user**, based on their preferences. Consider, for example, a smart home where the configuration for dimming lights in the evening or altering the lighting colour are complex, even time-dependent scripts. These scripts could be shipped to a user similar to simple content, but they have many configuration dimensions themselves, such as scaling with preferred light intensity etc.

2 project structure

2.1 Modules

The project consists of tree parts, represented in tree maven modules. For details of the associations see figure 1.

2.2 API

API contains classes and class structure used by the application server and clients who want to communicate with it. The classes contained in the API module represent the objects stored in the application. For clients it is important to uphold the class structure for the server to recognize those objects. For the structure of the module see figure 2.

2.3 Server

The Server module is the main module of the application. It contains the web application used to up- and download contents. It provides an rest interface for access and uses a MongoDB and the file system to store data. The *MongoDB* access can be configured in `tomcat‣webapps‣<current application name on the server>‣classes‣config‣mongo.properties`. For using the rest paths see the *ISO-IEC_CD_24752-8* Standard. For the module structure see figure 3.

2.4 Client

The Client is a sand alone module that uses the rest interface of the server to communicate with it. It can be replaced by any application capable of generating rest requests and processing the result.

3 Get started with the project

Remark This section shows how to set up a development environment to work with the project. To see how to deploy the application for use please skip forward to section Set up application.

3.1 Git repository

To use the repository used for this project you need a *Github* account and an application capable of running git commands.

3.1.1 Create Github account

To create an account visit github.com.

3.1.2 Install Git application

On *Linux* systems use your package installer to install a current git package. You can use the normal shell to run git commands. On *Windows* you can use the *Github* application from github.com and the build in git shell to run git commands.

3.1.3 Clone Git repository

To clone the repository use your console, which is able to use git commands, and navigate to the location where it should be cloned to. Then use `git clone https://github.com/REMEXLabs/OpenAPE`. You will be asked so sign in with your *Github* credentials.

3.2 Eclipse

The project is developed in *Eclipse* and so here is described how to use it. But any IDE can be used that provides *Maven* support.

3.2.1 Install Eclipse

The current version of *Eclipse* can be found on eclipse.org/downloads/.

3.2.2 Install Java

To develop this project with *Eclipse* you need a current *Java Development Kit*.
Use the download button under *JDK* oracle.com/technetwork/java/javase/downloads/index.html to get the current version.

3.2.3 Import project

It is recommended not to use the project folder as workspace for *Eclipse* but to import the project. To do so start *Eclipse* and navigate $File \rightarrow Import... \rightarrow Maven \rightarrow Existing\ Maven\ Projects \rightarrow Next \rightarrow Browse... \rightarrow$ browse the folder OpenAPE/openAPE $\rightarrow -> Finish$. Now you should have the main project plus the modules *API client* and *server* in your navigation.

3.2.4 Ensure the right Java version is used

The project will not work correctly if *Eclipse* doesn't use the previously installed *Java JDK* but a *Java RE*. To change the used *Java version* mark a module and navigate $Project \rightarrow Properties \rightarrow$ emphJava Build Path $\rightarrow Libraries \rightarrow$ mark the used system library $\rightarrow Edit... \rightarrow$ emphExecution environment $\rightarrow$ Choose a Java 1.8 JDK or newer fi $Search... \rightarrow$ Choose the installation path of the previously installed Java JDK. It should now be available in t Repeat this process for each module containing source code.

3.3 Maven

Maven is a build automation tool used to compile the project and create the deployable *war file*. Next is the order in which *Maven* commands are used to create the *war*. After importing the project this should be done once to update the projects *build path* and get rid of errors regarding it.

3.3.1 Maven deployment commands

Right click on the main project $\rightarrow Maven \rightarrow Update\ Project...$

Right click on the main project $\rightarrow Run\ As \rightarrow Maven\ clean$

Right click on the main project $\rightarrow Run\ As \rightarrow Maven\ install$

After each command wait for the IDE to finish. After the last command there should be a deployable *war* file in `server‣target`.

4 Set up application

The application needs an installed and running *MongoDB* to work. So first it is described how to set up that. The application itself can run on any web server capable of deploying a *war* file. Here is described how to deploy it onto a Tomcat server.

4.1 Set up MonoDB

4.1.1 Create setup.js

You will need a *setup.js* file containing the following lines.

```
var connection = new Mongo("localhost:27017");
var database = connection.getDB("openAPE");
database.createUser({
      user: "openAPE",
      pwd: "<userPassword>",
      roles: [ "readWrite", "dbAdmin" ]});
database.createCollection("environment_contexts");
database.createCollection("equipment_contexts");
database.createCollection("resource_descriptions");
database.createCollection("listings");
database.createCollection("task_contexts");
database.createCollection("user_contexts");
database.createCollection("test");
database.test.insert( { x: 1 } );
```

With your choice for the `<userPassword>`. That password must also be set in the *MongoDB* configuration file mentioned in Server. In the same way other parameters of the *monoDB* can be adjusted.

4.1.2 Install MongoDB

The used version of *MongoDB Server* can be found here docs.mongodb.com/v3.4/installation/. If you are working on *Windows*, after installing you may have to create the folder `C:▸data▸db`. This folder is used by *MongoDB* to run its databases in.

4.1.3 Start MognoDB

To start *MongoDB* open a console, navigate into its installation directory and use the command `mongod`.

4.1.4 Set up Database

To set up the database you have to run the previously created *setup.js* file. To do so you have to use the *mogno shell*. Copy the *setup.js* file into the installation directory of *MongoDB*. Open a console, navigate into its installation directory

and use the command `mongo`. Then use the command `load("setup.js")`. Now your database should be running, you can close the console containing the *mongo shell*. Closing the other console containing the *Mongo* application will stop you database. To start it again simply restart the *Mongo* Application. You don't have to run the *setup.js* again.

4.1.5 Robomongo

If you want to monitor your database you can use the tool Robomongo from robomongo.org.

4.2 Set up Tomcat server

4.2.1 Install Tomcat

You can get the latest version on hrefhttps://tomcat.apache.org/tomcat.apache.org. When installing on *Windows* I recommend to use the *32-bit/64-bit Windows Service Installer* and chose `C:‣web‣tomcat` as installation directory. It is also recommended to create an administration account for the web server while using the installer.

You can start the server by simply starting the execution file in `tomcat‣bin`. To stop it simply close the console that opens itself.

4.2.2 Deploy war file

To deploy the *war* file onto the server simply place it into `tomcat‣webapps`. After restarting *Tomcat* will create a Folder in there which is named after the war file and holds the currently executed application.

To check if it is working properly open a web browser and go to localhost:8080/war file name/hello. You should receive Hello World.

4.2.3 Changing the application name on the server

The application is usually named after the name of the war file. To change it to *openape* place your war file outside of the tomcat server. Create a file named `openape.xml` in `tomcat‣conf‣Catalina‣localhost`, with the following contents.

```
<Context
  docBase="<path/Application.war>"
  path="/openape"
  reloadable="true"
/>
```

4.2.4 Changing the tomcat IP-Address

Tomcat usually start itself on localhost:8080 and works only on the local system. To change that open `tomcat‣config‣server.xml`. Look for the `<Connector/>` element that has `protocol="HTTP/1.1"` as attribute. It should look something like that.

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

Add the attribute `address="<ipaddress>"` and restart *Tomcat*. You can change the *port* at the same point.

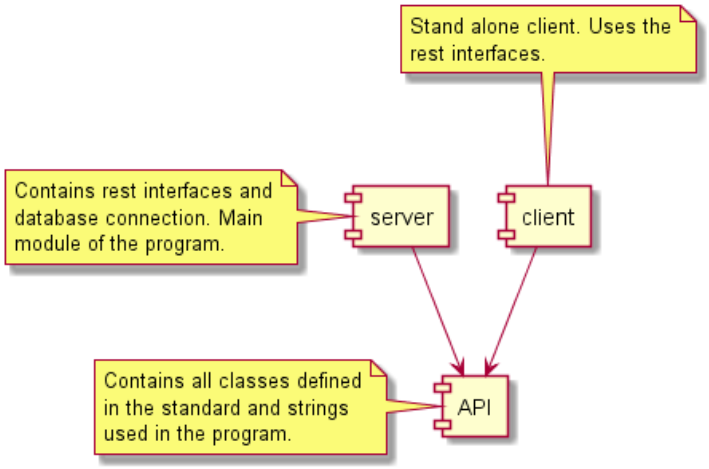

Figure 1: module diagram.

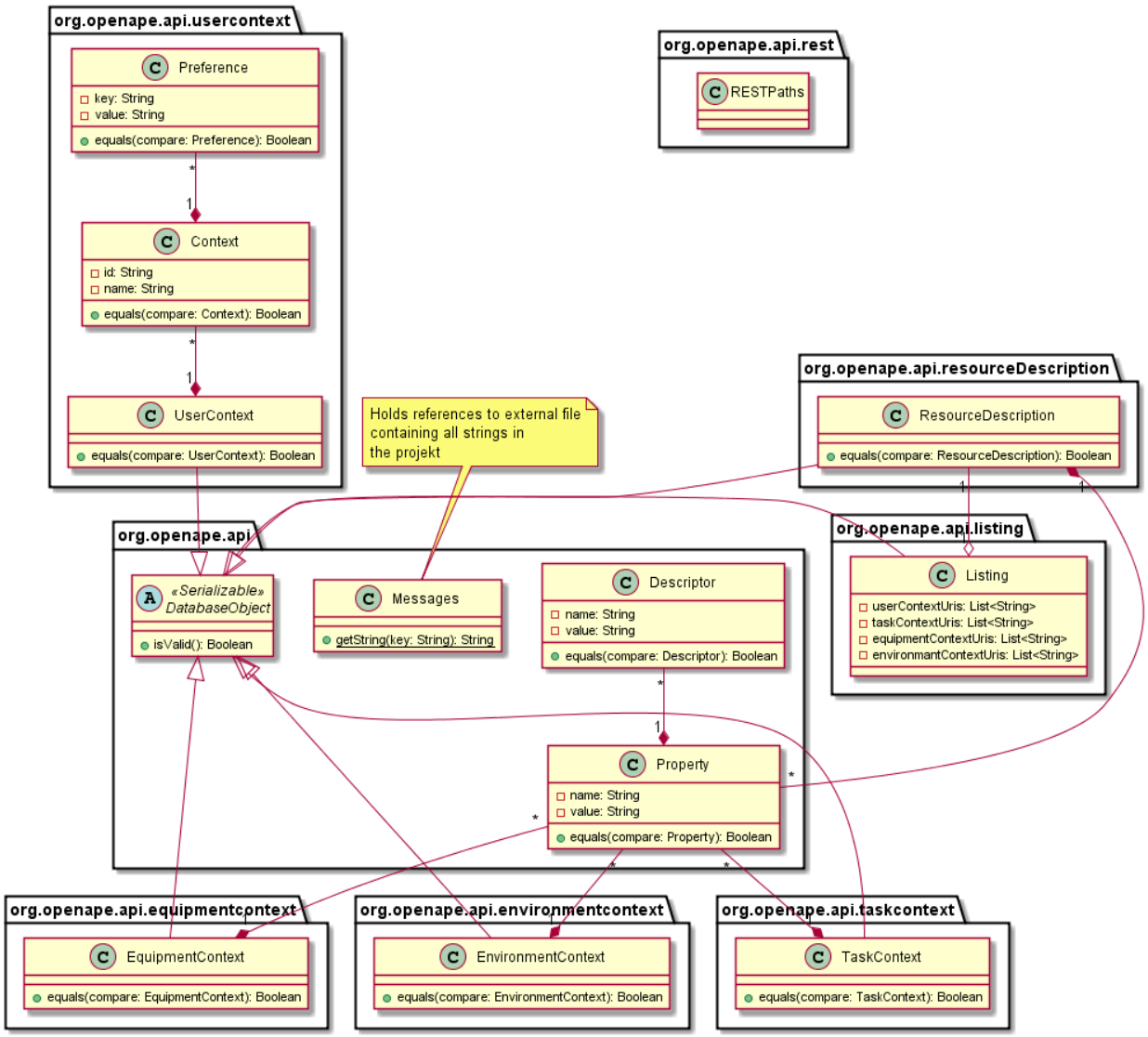

Figure 2: API module structure diagram.

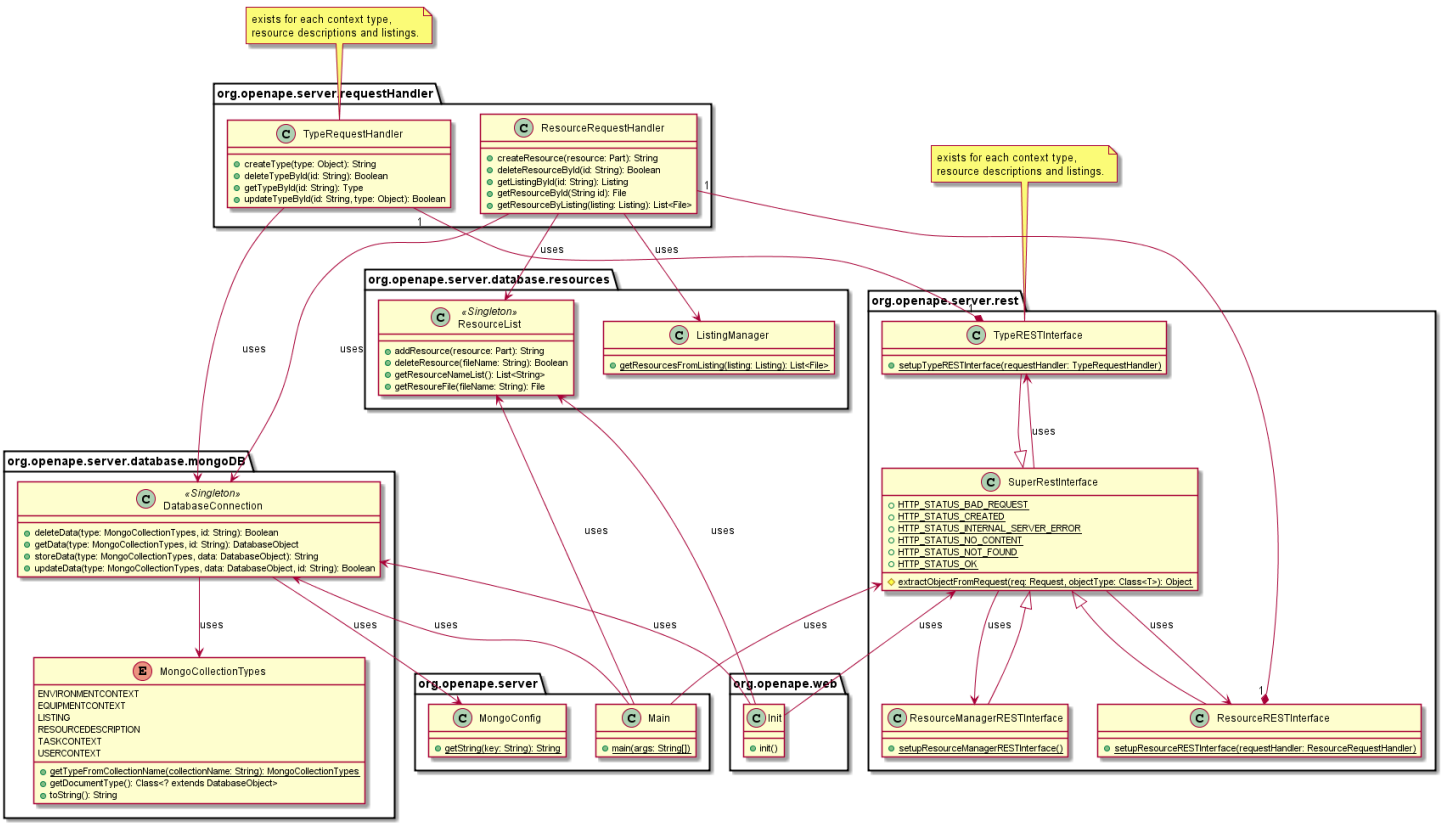

Figure 3: Server module structure diagram.