Implementation of the new REST API for open source LBS-platform Geo2Tag

Mark Zaslavskiy
ITMO University, Fruct Ltd.
Saint-Petersburg, Russia
mark.zaslavskiy@fruct.org

Dmitry Mouromtsev
ITMO University
Saint-Petersburg, Russia
mouromtsev@mail.ifmo.ru

***Abstract*—The article describes current state of Geo2Tag LBS platform project and new API version implementation. The platform was improved by following challenges: data visualization, extended datetime processing, social network integration and background calculations support. These challenges were justified by review of most important tendencies for geocontext applications and LBS platforms. Recommendations were fully implemented in API. Also the article contains description of new version implementation. As an example Open Data import API and specific plugin for Open Karelia system was implemented. This extension allowed performing geocontext markup of complex spatiotemporal data inside the platform.**

I. Introduction

Nowadays location-based services (LBS) and technologies became one of the most important trends at the mobile software development. Forecast [1] shows than total revenue of such services industry will grow by more than twice to the level of 2014 at 2019. The key factors according to authors are context-awareness and the ability to provide more relevant data to application users by taking into account their location. At the same time market of LBS instruments also will rise. Global LBS Platform Market 2015-2019 report [2] shows that compound annual growth rate for LBS platforms will increase at 2015-2019 by more than 22 percent. These two trends demonstrate that development of LBS technologies and tools is important task.

Geo2Tag is an Open Source LBS platform solution. The platform allows creating mobile, desktop and web-based geo and geocontext applications using already implemented backend with data storage and processing functions. Geo2Tag was recommended by IEEE Internet of Things technical community as a candidate platform for prototyping IoT solutions in City tagging scenarios [3]. These scenarios includes geocontext markup of the urban environment.

Goal of the proposed article is to define, justify and describe Geo2Tag API changes needed to match IoT solutions in City tagging scenarios. These changes include implementation of complex background user-defined data processing mechanisms, date intervals support, custom maps interfaces and social network integration.

II. Use-case

Particular use-cases of Geo2Tag LBS platform services were described in details at [15-16], [6]. The platform use-case at [6] is described as a common backend for services includes solution of following tasks:

- spatiotemporal data storage;
- spatiotemporal data processing;
- data isolation inside separated services;
- user management.

Modification of the API described in this article also affects platform use-case. The most important changes are related with background data processing interfaces. Due to end user direct involvement into this mechanism it should be accessible by users (for creation and control on background computation tasks) and also by server administrators (for configuration and maintenance. The social network integration also changes the use-case. Much of modern web-services provide OAuth2-based registration using existing accounts in email services and social networks. This approach allows web-services to avoid complex user management routine such as user registration with email confirmation and therefore user management in the Geo2Tag use-case can be replaced to authorization. Also custom maps support extends available user actions by data visualizations. New use-case is displayed at Fig.1.

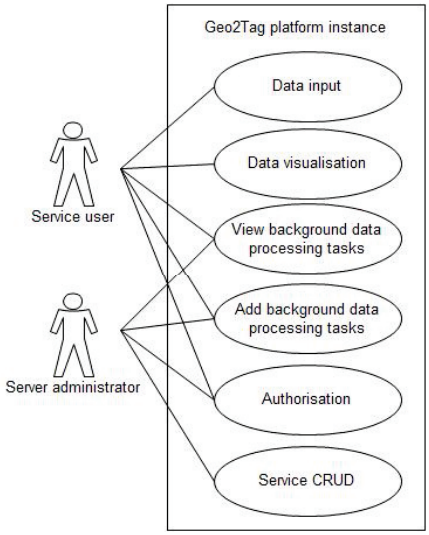

Fig.1. Use case diagram for platform

ISBN 978-952-68397-0-7 (paperback), ISBN 978-952-68397-1-4 (PDF)

III. LBS PLATFORMS AND GEOCONTEXT APPLICATIONS REVIEW

For justifying requirements for new Geo2tag API the review of existing LBS platform and geocontext solutions should be done. Before the comparison "geocontext" term should be defined. Definition will be based on the definition of "context" term from work [4] and ideas from [5]: geocontext is any information about location of an entity, which can describe current state.

Review of geocontext tendencies was done in the previous paper [6]. This article shown that one of the key trends is support of spatiotemporal operations on a level more sophisticated than scalar dates processing. Also the work [6] demonstrated that creation of standalone geocontext applications is inefficient in mobile and web cases. Instead, applications which use separated backend can achieve more power-saving and attract more users. The third important conclusion covers the structure of geocontext markup. The most effective way is to perform computations in two steps. On the first step simple statistic processing (average, variance etc) of the whole dataset should be done. Special geocontext calculation may be done at the second step. Such architecture of computations is impossible in case of frequent requests and constantly updating data without running statistical computations in background of backend.

For understanding existing trends in LBS platforms functionality the review must be done. Following different solutions were selected:

- Search and map service providers APIs:
  - GoogleMaps [7],
  - YandexMaps [8],
  - HEREMaps [9].
- Cloud LBS platforms for geo and geocontext application development:
  - ojoo [10] ,
  - shoutem [11],
  - geoloqi [12].

For the comparison following criterions were chosen:

- custom web-based maps
- processed data visualisation on maps
- social network integration.

TABLE I. COMPARISON OF BACKENDS FOR GEO APPLICATIONS

	Custom web-based maps	Processed data visualization on map	Social network integration
GoogleMaps	+	+	-
YandexMaps	+	+	-
HEREMaps	+	+	-
ojoo	+	+	+
shoutem	+	+	+
geoloqi	+	+	+

The comparison results can be seen at the Table I. It shows that the most important user features for LBS platform are a support of custom maps and data visualization, because they are supported by the almost all solutions and became a standard de facto for LBS platform. Social network integration is widely supported by specialized platforms instead of search and map service providers and can be a competitive advantage.

As a conclusion requirements for Geo2Tag API can be found sufficient and corresponding to modern LBS platform tendencies.

IV. GEO2TAG ARCHITECTURE

A. Data model

Before describing Geo2Tag architecture details its basic conceptions should be described. The data model of the platform is very simple - it contains only four entities. Point is an annotated media content related to specific geographical coordinates (latitude, longitude and altitude) and a datetime object (exact date or a time period). Channel is a named and annotated set of points. Service is an annotated, named and isolated set of channels which can be treated as backend for a single mobile or web geo application. Platform instance is a single installation of the Geo2Tag platform containing combination of a set of services and service-user relations data. User is an application or a human, who performs requests to the platform. Users can be divided into two categories: service users (service administrators and geo application end users) and platform instance users

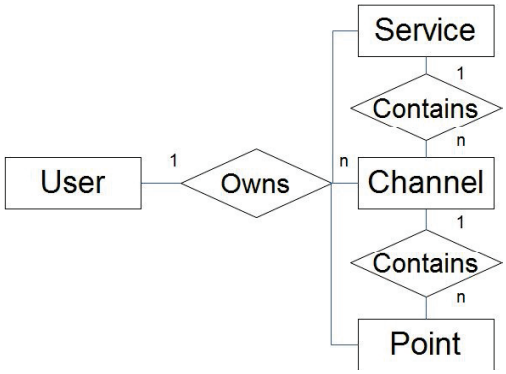

Fig.2. ER diagram for platform entities

Most important relations between entities are showed at Fig.2.

B. Internal architecture

According to use-case for end users the platform should be able to process big number of simultaneous different requests to different services from the external network. That's why service-oriented architecture was chosen.

The architecture is illustrated at Fig. 3. A dashed rectangle shows virtualization environment, which wraps transparently the platform. Such structure is needed for a seamless integration with cloud computing providers - Vagrant images of Docker containers can be deployed easily in such environment. Geo2Tag consists only of two basic parts: Query Engine and the Core. Query Engine encapsulates REST API queries processing and forwarding them to Core APIs. These

APIs stands for fundamental actions - control and management of data access rights, interaction with database server, and execution of third party plugins. Such division is needed for abstraction from specific technologies. For example, query processors, which are interacting with database do not do it directly, instead they use core database API, hiding database-specific realization.

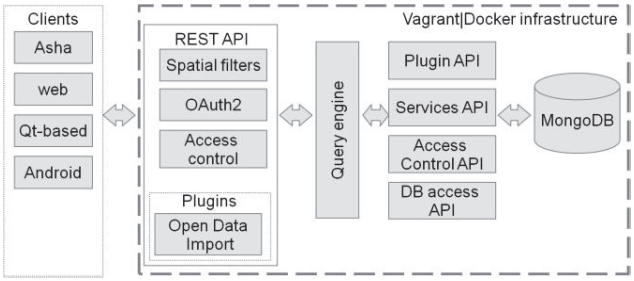

Fig.3. Platform architecture schema

C. Databases

Because main goal of Geo2Tag platform development was a creation of a simple and powerful backend for different types of geo application Platform as a Service (PaaS) service model was chosen as a foundation. It was implemented as a conception of separated services, working simultaneously and storing their data in separated databases. These databases do not have links between each other, but the metadata about all of them are stored at the special master database. Fig. 3. shows relations between two database types.

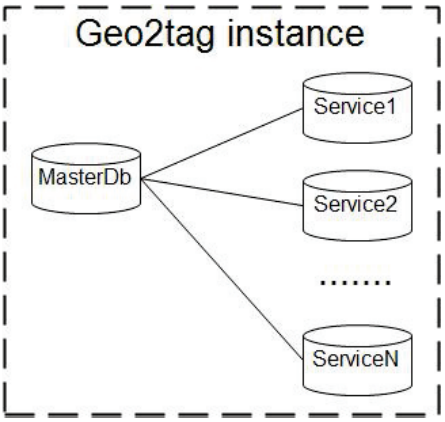

Fig.4. Platform database relation diagram

Service databases store following service-specific objects - service logs, points and channels. Master database contains of users, services and logs collections.

Logs are stored at the service and master databases due to MongoDb ability to organize collection as a cycled buffer and because of high-performance search interfaces.

D. REST API

The API for platform users is implemented in a form of REST API working over the HTTP. All REST resources (Webs) have mandatory prefix, which is needed for simultaneous work of several Geo2Tag instances at one domain name.

CRUD operations for REST resources were mapped to the HTTP methods in the following way:

- create - POST, /resource_Web/;
- read - GET, /resource_Web/specific_resource_id or /resource_Web/ (several resources);
- update - PUT, /resource_Web/specific_resource_id;
- delete - DELETE, /resource_Web/specific_resource_id.

API can be divided for two parts - service API and instance API. Service API includes resources (Webs), which allow operating with service data (points, channels) and maintaining service work.

TABLE II. SERVICE REST API

Resource	Operations
/<instance_prefix>/service/<service_name> /point	create, read, update, delete
/<instance_prefix>/service/<service_name> /channel	create, read, update, delete
/<instance_prefix>/service/<service_name>/log	read

Instance API implements services and plugins management, instance administration.

TABLE III. INSTANCE REST API

Resource	Operations
/<instance_prefix>/service	create, read, update, delete
/<instance_prefix>/logout	read
/<instance_prefix>/plugins	read
/<instance_prefix>/manage_plugins	read

E. Authorization

User authorization is based on OAuth2 [13] protocol. Login procedure implements following use-case:

- User visits /<instance_prefix>/login - web-page with buttons leading to OAuth2 providers.
- User press to button of some provider.
- Browser redirects user to provider authorization page.
- User gives his agreement at the OAuth provider web-page.
- OAuth provider redirects user back to Geo2Tag web-page with user identifier.
- Platform check is user already registered.
    - If user is not registered, then platform add new document into master DB users collection.

- Backend stores received user identifier into cookies as a symmetrically encrypted string.

Logout procedure implements other use case:

- User performs GET request to /<instance_prefix>/logout.
- Backend cleans user identifier from cookies.

For the current moment there is only one GooglePlus connected as a OAuth2 provider, but interfaces allow to add support of any other authorization source based on social network.

Approach described below have several significant pros. At first it simplifies registration process for new users, allowing using existing accounts. At second, this approach does not require usage of complex user verifying schemas, as email or SMS verification and thou approach simplifies platform architecture and maintenance.

F. Data visualization

Modern geo application cannot be interesting to users without proper visualization. Therefore geo data visualization API was implemented inside Geo2Tag. Because the most popular form of geo data view is a map, the following architecture was chosen.

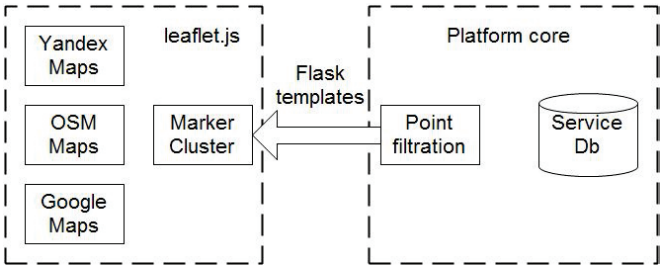

Fig. 5 Visualization web-service architecture

The visualization API was implemented in a form of a web-service available at the /<instance_prefix>/map resource. The service receives the same arguments as /<instance_prefix>/service/<service_name>/point and returns web-page, where the /<instance_prefix>/service/<service_name>/point request results are displayed. Visualization contains of custom map widget, based on leaflet.js library and several map backgrounds, which can be changed in real time by user. Displayed points are grouped into clusters by their proximity using MarkerCluster plugin for Leaflet.js. Number of displayed points and clusters is adjusted according to the map scale. Such approach was chosen because displaying big amounts of geo data without reduction of number of visible objects is very memory-intensive.

Each point displayed on a map of the web-server includes pop-up window with short description of related content. Fig.6 illustrates how geo objects are displayed at the map.

G. Platform deployment

Because main target platform of Geo2Tag will be different cloud services, platform deployment process should be unified to different operation systems. Due to diversity of used Linux distributives and their package manager's maintenance of all existing package types is difficult task. Therefore installation process should not depend from the target system and thus virtualization was selected as a deployment instrument. Vagrant configurations were created for platform installations on end users and developers servers. Docker files were written for automated testing of Geo2Tag during development.

H. Plugin system architecture

Main goal of Geo2Tag platform is to provide very simple and effective API for building applications. But at the same time geo services are development is affected with constantly changing preferences of users. In such situation standard API may not satisfy application requirements. For this purpose the plugin system was implemented inside the platform.

Fig.6. Platform database relation diagram

In term of the Geo2Tag platform plugin is a python package, which extends REST API with new resources. Due to security issues all new requests are isolated inside /<instance_prefix>/plugins/ resource. Plugins are installed by copying their sources in separate directories to the plugins/ dir at the platform path. After this procedure plugins can be managed with following REST requests:

- /<instance_prefix>/plugins - receive list of installed plugins and their statuses (enabled/disabled),
- /<instance_prefix>/manage_plugins?<plugin_name>=<status> - change plugin status.

I. Open Data import plugins

For making Geo2Tag an instrument for processing of Open Data the API for building import plugins was developed. Let us define basic requirements for such plugin. Because lots of Open Data sources can be related to the Big Data plugin should allow importing single data elements from external sources into specific service database with keeping Web to the origin and without copying all original content. Import extension must provide procedure for translation of data

elements format to Geo2Tag point's format with extraction of location and time information because it will allow processing imported data with standard platform API. Due to versioning of imported points plugin also should keep information about import date of a data element.

The API provides general import algorithm, abstract classes for implementing source-specific solutions and common REST interfaces for controlling import procedure as a building blocks for import plugin.

Import algorithm use following input data: channel where points will be saved later, link to the Open Data source and name of a service. The algorithm includes four stages:

1) Data acquiring. At this stage steps for data downloading are performed including authorization and connection setup to the Open Data source.
2) Data array splitting into single elements. This stage includes initial parsing of received data array and conversion it to the python list of single serialized elements.
3) Translation of each element into point. On translation stage elements are parsed and converted into points. All mandatory fields as location, altitude, date and channel identifier. This stage also includes saving the import information (Open Data Web and datetime, when import was performed) inside point.
4) Points saving to service database.

Classes included into the API contain logic for each step of algorithm and templates for REST interfaces implementation:

1) Job - basic abstract class with common interfaces for import jobs running in parallel;
2) JobListResourceFactory - factory method for generating job list resource request processors;
3) JobManager - class which controls execution of jobs and provides statistics;
4) JobResource - job resource request processor;
5) OpenDataToPointsLoader - class for saving points list into service's channel;
6) OpenDataToPointTranslator - basic class for translation between Open Data element format and Geo2Tag point format;
7) OpenDataObjectsLoader - basic abstract class for loading data elements array from Open Data source;
8) OpenDataObjectsParser - basic abstract class for splitting Open Data into separate elements.

Plugins developed with the API have standardized REST interfaces for import job management:

- /<instance_prefix>/plugin/<plugin_name>/service/<service_name>/job - allows to add new job by POST requests and view status of existing ones by GET;
- /<instance_prefix>/plugin/<plugin_name>/service/<service_name>/job/<job_id> - allows viewing status of specific job by GET request and stopping it by DELETE.

V. Evaluation

As an implementation of Open Data import API Open Karelia [14] import plugin for Geo2Tag platform was created. Open Karelia was chosen for implementation because it stores spatiotemporal data, supports bc dates and interval dates. The plugin was developed to match format of Get Nearest Objects web-service at the mobile frontend. This web-service collects all Open Karelia objects with valid information about location and which are not far from given geographical coordinates more than one kilometer.

The plugin extended basic API in two ways. At first, all objects imported from Open Karelia in a format of Geo2Tag points store information about source object name, brief description, image and data interval. These attributes are stored at json field of the Geo2Tag point. At second, import jobs were implemented using python threads. This solution has low performance due to Global Interpreter Lock (GIL) restrictions in Python, but it is acceptable for proof of concept implementation.

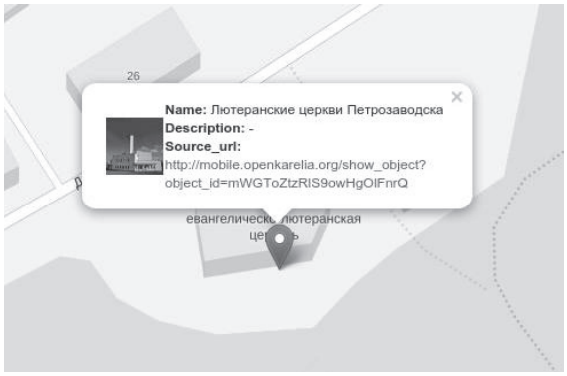

Fig.7. Imported object view

OpenKarelia plugin was included into the platform source code as an example of import plugin and deployed to the demoserver (example on Fig. 7) where it can be viewed using the map web-service. The page shows all available points, including ones which were imported with the plugin. Each point is displayed as a marker with a popup, where information is shown.

VI. Conclusion

In this paper reasons for creating a new version of Geo2tag REST API are revealed and the changes in architecture were described. These reasons are related to challenges connected with prototyping of IoT solutions in City tagging scenarios: complex background processing of geo data, support of the different types of datetime data, support custom data visualization and social network integration.

New version of the Geo2tag REST API contains operation on all basic entities, which were partly renamed, and several new requests. The plugin system inside the platform allowed performing user computations on the service's data in a background. Also, the API for Open Data import was implemented as a part of the system. For evaluation purpose of such extension were developed for Open Karelia system

Data visualization were implemented as a web-service using leaflet.js and MarkerCluster libraries combined with maps of most popular map providers – Google, Yandex and OSM.

The authorization were reimplemented using OAuth2 protocol allowing usage of existing accounts from social networks or other authorization providers.

ACKNOWLEDGMENT

Authors would like to thank Finnish-Russian University Cooperation in Telecommunications Program (FRUCT, fruct.org) for provided support. This work was partially financially supported by Government of Russian Federation, Grant 074-U01.

REFERENCES

[1] Location based services market to reach $43.3bn by 2019, driven by context aware mobile services. Web: http://www.juniperresearch.com/press-release/context-and-location-based-services-pr2.

[2] Global LBS Platform Market 2015-2019 - Increased Demand for Location-based Services. Web: http://www.prnewswire.com/news-releases/global-lbs-platform-market-2015-2019---increased-demand-for-location-based-services-300142322.html.

[3] IEEE IoT. IoT Scenario & Use Cases: City Tagging. WEB: http://iot.ieee.org/images/files/pdf/scenarios/IEEE_IoT_Service_Use Cases_CityTagging_clean.pdf.

[4] A. K. Dey, "Understanding and Using Context", *Personal and Ubiquitous Computing*, v.5 n.1, February 2001, pp.4-7.

[5] A. Schmidt, M. Beigl, H.W. Gellersen, "There is more to context than location", *Computers & Graphics,* v.23 n.6, 1999, pp 893-901.

[6] M. Zaslavskiy, D. Mouromtsev, "Geocontext extraction methods analysis for determining the new approach to automatic semantic places recognition", *in Proceedings of the 14th Conference of FRUCT Association*, 2014, pp. 137-142.

[7] Google Maps API, Google Developers. Web: https://developers.google.com/maps/?hl=ru.

[8] Yandex maps API, Yandex. Web: https://tech.yandex.ru/maps/.

[9] Build applications with HERE Maps API and SDK Platform, HERE developer. Web: https://developer.here.com/.

[10] Create and play location-based apps for iOS & Android, Ojoo. Web: http://ojoo.com/.

[11] Create a Location Based App - Daily Deal Apps, ShoutEm. Web: http://www.shoutem.com/local.

[12] A powerful location-based API for application development // Geoloqi Developers. Web: https://developers.geoloqi.com.

[13] OAuth2.0. WEB: http://oauth.net/2/.

[14] Open Karelia. WEB: http://openkarelia.org/about.

[15] E. Balandina, S. Balandin, Y. Koucheryavy, D. Mouromtsev, "IoT Use Cases in Healthcare and Tourism", *in Proc. of the 17th IEEE Conference on Business Informatics (CBI 2015)*, Lisbon, Portugal, July 13-16, 2015.

[16] E. Balandina, S. Balandin, Y. Koucheryavy, and D. Mouromtsev, "Innovative e-Tourism Services on top of Geo2Tag LBS Platform", *submitted to the 11th International Conference on Signal Image Technology & Internet Systems (SITIS 2015)*, Bangkok, Thailand, Nov. 23-27, 2015.