Updated Platform Release - Y2

MARCH 2016

DELIVERABLE

Project Acronym: **SDI4Apps**
Grant Agreement number: **621129**
Project Full Title: **Uptake of Open Geographic Information Through Innovative Services Based on Linked Data**

D3.4.1 UPDATED PLATFORM RELEASE - Y2

Revision no. 03

Authors: Karel Chavat (CCSS)
Premysl Vohnout (BOSC)
Marin Kuba (MU)
Jan Jezek (UWB)
Michal Kepka (UWB)
Dmitrij Kozuch (HSRS)
Runar Bergheim (Avinet)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	X
C	Confidential, only for members of the consortium and the Commission Services	

REVISION HISTORY

Revision	Date	Author	Organisation	Description
01	30/12/2015	Karel Charvat	CCSS	Initial draft
02	20/03/201	Karel Charvat, Premysl Vohnout, Marin Kuba, Jan Jezek, Michal Kepka, Dmitrij Kozuch, Runar Bergheim,	CCSS, BOSC, MU, UWB, Avinet	Contribution
03	31/03/2016	Karel Charvat		Final Version

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Disclaimer:

Views expressed in this document are those of the individuals, partners or the consortium and do not represent the opinion of the Community.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

EXECUTIVE SUMMARY

The report describes the second version of SDI4Apps portal and its inhalation as Virtual server. Current version including all components is now available to be install in any cloud environment.

1 INTRODUCTION

During the second year, the enablers were integrated into one operational platform. This platform is now available on Masaryk University cloud (portal.sdi4apps.eu). There is now also prepared virtual server, which could be easily deployed to any cloud. This opens possibilities for different users to run their own platform.

2 BASIC CLOUD ENABLERS UPDATE

The main update to the basic cloud enablers is the ability to deploy to multiple Infrastructure-as-a-Service clouds in a uniform way. The previous version was deployable only to the research cloud operated by **Masaryk University**, the current version can be deployed also to the main commercial clouds: **Amazon Elastic Compute Cloud (EC2), Google Computing Engine (CE)** and **Microsoft Azure**.
The current version of the generic cloud enablers is based on the **Ubuntu 14.04 LTS** version of Linux, which is provided by all the 4 cloud providers in the form of a disk image with pre-installed base operating system. The pre-installed operating system includes a software package named **cloud-init**, which modifies the base operating system after boot following instructions provided by user and thus called **user-data**.
The SDI4Apps platform is now represented as the user-data instructions for modifying the base Ubuntu 14.04 LTS system to a system with all needed SDI specific enablers installed and running.
The user-data instructions are maintained in the GitHub repository https://github.com/SDI4Apps/cloud-platform in the file **user-data.yaml**, which needs to be provided when a new virtual machine (VM) with an instance of the SDI4Apps platform is to be started in a cloud.
The way in which the user-data.yaml should be provided when launching a new VM differs from cloud to cloud. In the Masaryk University's cloud, a special VM template was created for this purpose. In the Amazon EC2, it can be specified through its web interface in the Step 3 of its Launching wizard. In the Google CE and Microsoft Azure clouds, the new VM must be launched using their command-line tools and the user-data.yaml file needs to be specified as a command parameter. Detailed instructions are provided in the GitHub repository.

The user-data.yaml gives instructions for the following changes:

- **adds software package repositories** for PostgreSQL and Oracle Java in addition to the official Ubuntu package repositories
- **installs enablers** that are available as software packages from the package repositories (PostgreSQL, PostGIS, Oracle Java, Apache, PHP, Python, Mapserver, etc.)
- **creates a user account** named ubuntu accessible by the person that launched the VM
- downloads and runs the script install_sdi4apps.sh that installs and configures all the remaining enablers that could not be installed from the package repositories

In the end, when the cloud-init finishes, the following enablers are installed and configured:

- Apache 2.4 + PHP
- Geoserver 2.8.2
- HSProxy
- LayMan
- Liferay 6.2 GA6 with geo portlets
    - MapViewer - portlet with full functionality. Specialized on visualising broad amount of spatial formats and creating map outputs
    - MapViewer Lite - portlet with limited functionality. Mainly used for consuming predefined spatial content
    - Layman - portlet with webui for LayMan application which provides functionality for publishing spatial data
- MapServer 6.4.2
- MICKA
- Oracle Java 7
- pgRouting
- phpPgAdmin

- PostgreSQL 9.5
- PostGIS
-
- Virtuoso 7.2
- JavaScript libraries
    - ExtJS 4.2.1
    - jQuery 1.12.0
    - Proj4js
    - HSLayers NG
        - Proxy4ows
        - Statusmanager
    - WebGLayer

The freshly started VM provides a web server that can be visited with a web browser, and an ssh (secure shell) server that can be used for command line access.

This uniform way of deployment of the SDI4Apps platform to any cloud (that provides a Ubuntu 14.04 LTS image with cloud-init) allows easy creation of new instances of the platform. Also, as the enablers are always freshly installed, it enables easy updates of the platform by simply launching a new instance of the platform.

For future, we plan to use Docker containers to better separate the enablers from each other and from the underlying operating system.

3 SPECIFIC ENABLERS FOR SDI DOMAIN UPDATE

3.1 HSLayers-NG

HSlayers NG (https://ng.hslayers.org/) is a web mapping library written in Javascript. It extends OpenLayers 3 functionality and takes basic ideas from the previous HSlayers library, but uses modern JS frameworks instead of ExtJS 3 at the frontend and provides better adaptability. That's why the NG ("Next Generation") is added to its name. It is still under development and provided as open source. HSLayers is built in a modular way which enables the modules to be freely attached and removed as far as the dependencies for each of them are satisfied. The dependency checking is done automatically.

The core of the framework is developed using AngularJS, requireJS and Bootstrap. This combination of frameworks was chosen mainly for providing fast and scalable development and for providing modern responsive layout of application.

The most important modules are:

- Map: The map functionality is provided by OpenLayers3 and extended by some controls like navigation bar, scale line, attribution dialog, GPS and compass tracking etc. It supports multi-touch gestures, but the performance is highly dependent on the browser and mobile device hardware.
- Layer manager and legend: Layer manager is used for listing all the map layers, displaying or hiding them and setting the transparency. The user can view layers metadata and attribution by clicking on it. A legend is fetched from the server and displayed in a separate panel for all the wms layers on the map. Grouping of layers in containers is also provided which enables a more user friendly and organized representation of layers for
- OGC Web Services parser: This is used for GetCapabilities requests to different map servers and parsing the response. It can then be used for automatic or user initiated generation of map layers only by knowing the URL to the specific OGC standardized map service.
- Linked Open Data explorer: Eurostat explorer is a demo application (module) which queries Semantic Web data sources via SPARQL endpoints. It demonstrates the feasibility of automatic query building for Eurostat report data and displaying it on a map of NUTS2 regions (specified in GeoJSON file) according to the calculated transparency ratios. On the server side it uses a Virtuoso Universal Server which is a middleware and database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, web application server and file server functionality in a single system.

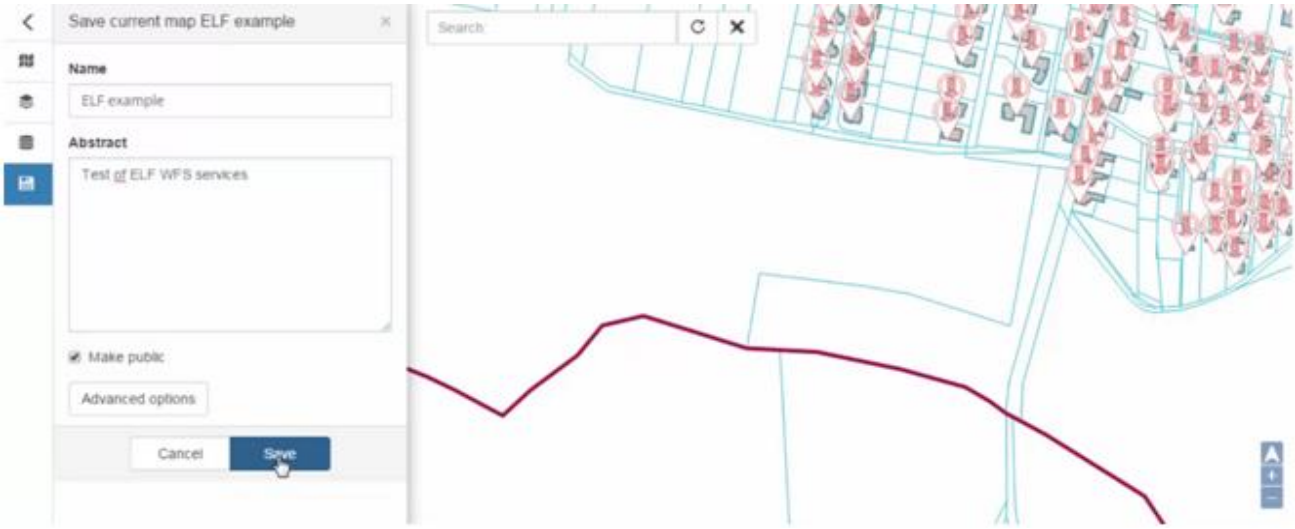

Figure 1: The figure shows interface for creating map outputs

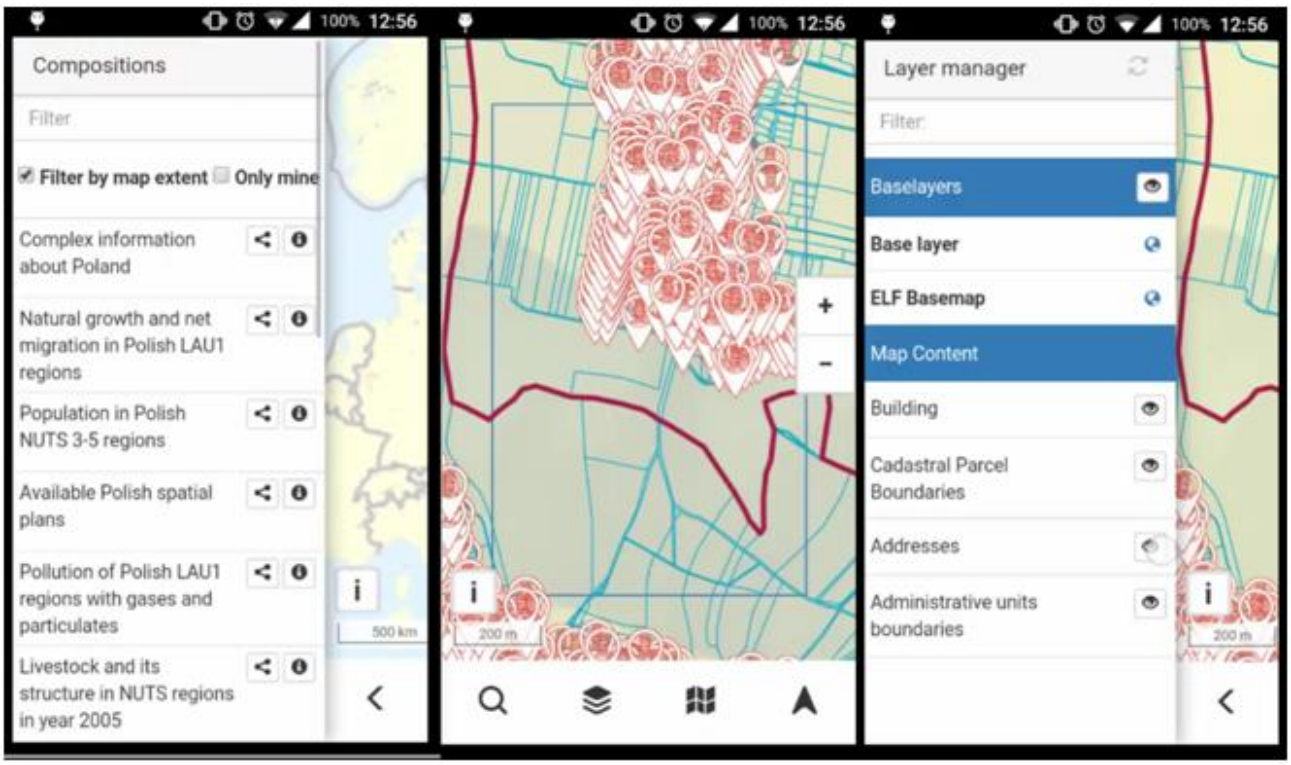

Figure 2: The figure shows mobile version of HSLayers-NG.

3.2 Layman

When geodata goes public, several steps are needed: Upload the data to the server, import the data into the database, publish the data through some kind of map server, and, if needed, configure the access rights so only the users with the proper privileges can display the data. LayMan – the Layer Manager – sorts it out for you.

LayMan offers a single entry point into the filesystem, PostGIS database and GeoServer:

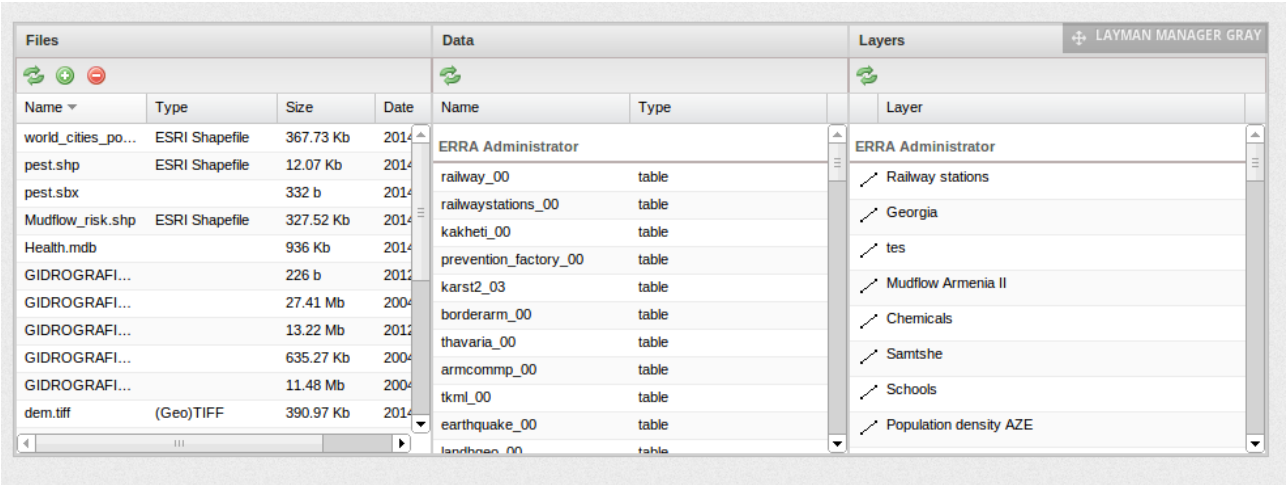

Figure 3: LayMan Web GUI

On the left side, files in the user directory are shown. In the middle, there are tables and views with data that has been already imported into the database. On the right side are the layers that have been already published with GeoServer.

The files belong solely to the user that is logged-in. Data and layers are common for the whole group they have been published to and can be manipulated by any member of the group. The user sees the data and layers of every group he/she is a member of.

Data can be published either from the uploaded files, or from the tables or views already present in the database. Various parameters of a layer can be set, with access control being of a special interest:

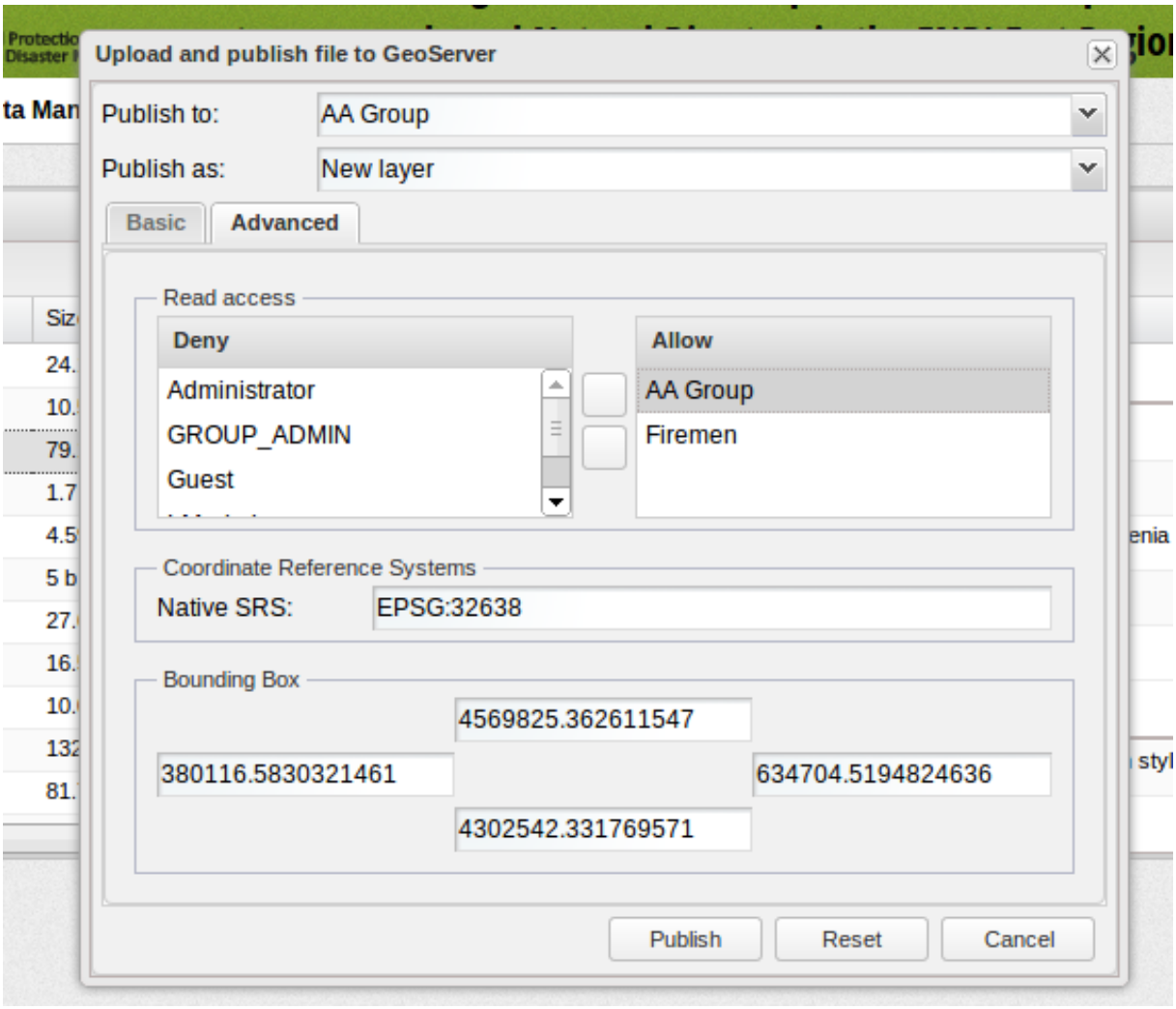

Figure 4: Publishing with Layman

While manipulation with the published layer (write access) is limited to the members of the group that layer is published to, the read access (showing the layer in a map) can be granted to any other group. Users and groups are managed within the Liferay portal which encapsulates the whole system.

Once published, the layers can be styled with OpenGeo Styler:

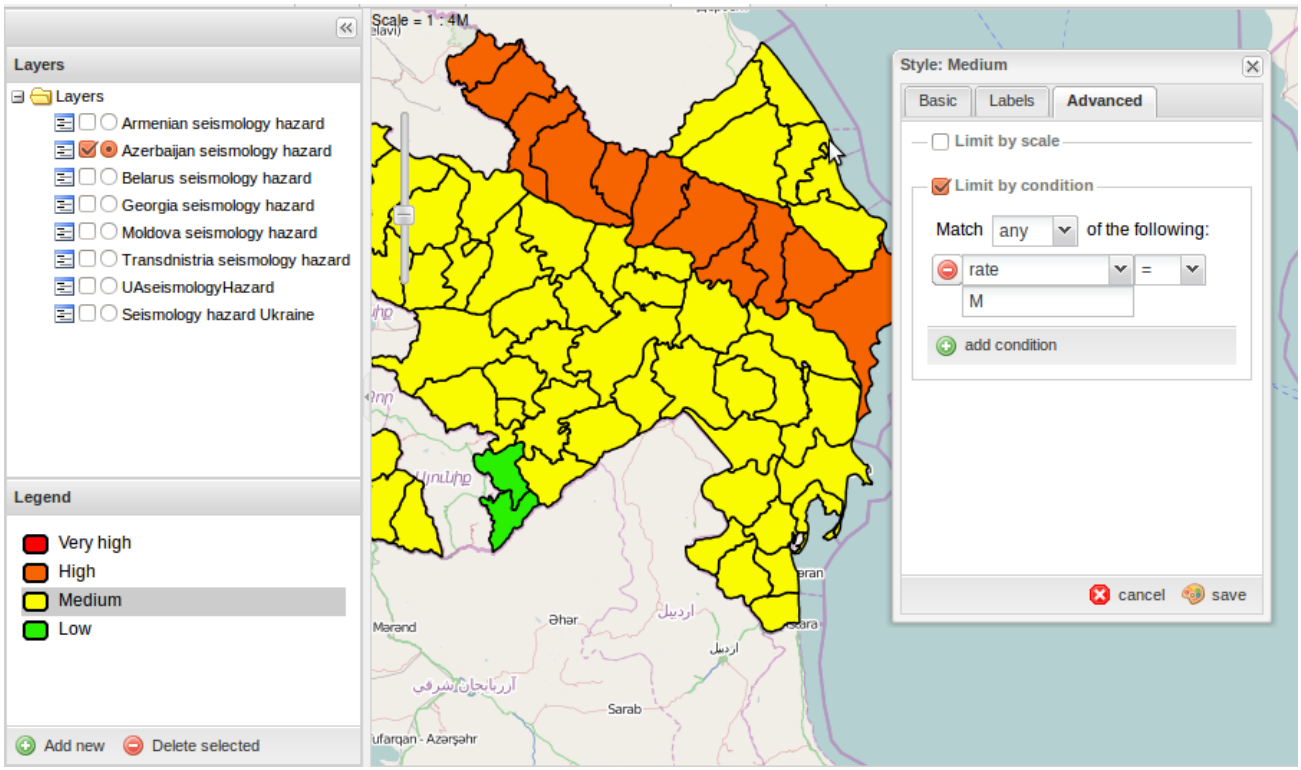

Figure 5: Styler

And of course the layers can be shown on the map:

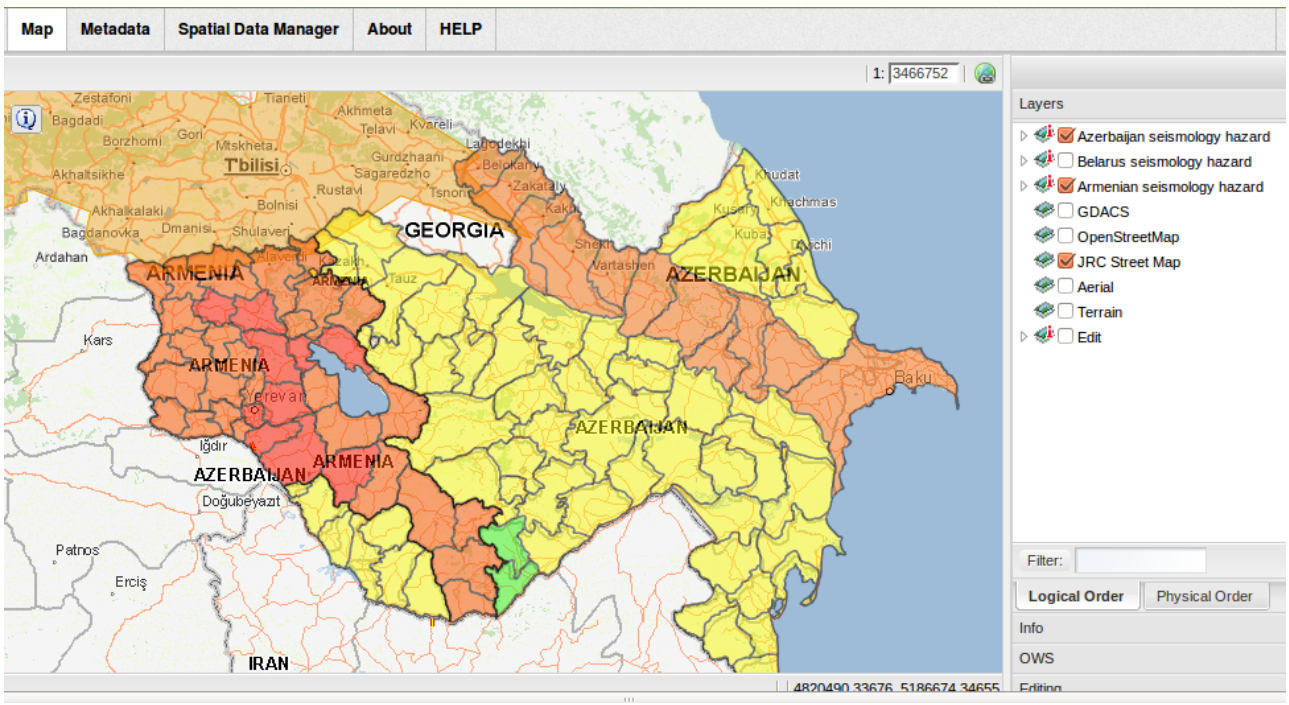

Figure 6: Map

3.3 Virtuoso

Virtuoso Universal Server[1] is a middleware and database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text content management & full-text indexing, linked data server, web application server and file server functionality in a single system. So, instead of providing dedicated servers for each of these functionality realms, Virtuoso enables a single multithreaded server process that implements multiple protocols (see architecture diagram in Figure 2215). It is designed to take advantage of operating system threading support and multiple CPUs.

Virtuoso database engine includes physical file and in memory storage and operating system processes that interact with the storage, provides dynamic locking from row to page level, support transactions as well as entity and referential integrity.

Moreover, in addition to the functionality realms mentioned above, Virtuoso implements several industry standard Web & Internet protocols, including among others: HTTP, WebDAV, UDDI, SOAP, WSDL, SPARQL and SPARUL. It also implements a variety of industry standard data access APIs for the database (e.g., ODBC, JDBC, XMLA), and supports different standards for content syndication and interchange format (e.g., Atom, RSS, FOAF). Virtuoso also supports several query languages, including SQL, SPARQL, XQuery, XPath and XSLT.

Regarding Virtuoso RDF, key features include: an RDF triple store, SPARQL query language support, SPARQL protocol support, inline SPARQL integration within SQL, use of bitmap indices for optimizing storage and management of RDF triples, implementation of the HTTP-based Semantic Bank API[2] that enables client applications to post to its RDF Triple Store, and several RDF insert methods, including http PUT and POST.

Virtuoso SPARQL can use an inference context for inferring triples that are not physically stored. Such an inference context can be built from one or more graphs containing RDF Schema triples. The supported RDF Schema or OWL constraints are imported from these graphs and are grouped together into rule bases. A rule base is a persistent entity that can be referenced by a SPARQL query or end point.

Virtuoso's reasoning includes support for owl:sameAs, rdfs:subClassOf, rdfs:subPropertyOf, owl:equivalentClass, owl:equivalentProperty, owl:InverseFunctionalProperty, owl:TransitiveProperty, owl:SymmetricalProperty, and owl:inverseOf.

The latest release v7.1.0 includes improvements in the Engine (SQL Relational Tables and RDF Property/Predicate Graphs), SPARQL compiler, Jena and Sesame provider performance and GeoSpatial support, among others.

Virtuoso comes in two editions: Open-source and Commercial. The difference between these two is that the Open Source Edition does not include the Virtual Database Engine and Data Replication Functionality of the Commercial Edition.

---

[1] http://virtuoso.openlinksw.com/
[2] http://simile.mit.edu/wiki/Semantic_Bank

3.4 MIcKA

MICKA is a complex metadata system developed by HSRS (a member of WirelessInfo). It includes spatial metadata editor, metadata catalogue and metadata harvester. It supports the CSW 2.0.2 / INSPIRE discovery service. In addition to the CSW 2.0.2 / ISO-AP 1.0 profile it supports many output formats including HTML, PDF, JSON, GeoRSS, Atom, KML, OAI_PMH and OAI_MARC21.

MICKA as part of the REB platform is responsible for storing meta-information about available data and supporting discovery of existing geospatial data and services. Its role is very important for the platform. MICKA is a bridge to the current INSPIRE geoportal and services. MICKA supports internal management of geospatial data. It helps to keep certain consistency of data. Since the metadata profile is compliant with international standards, it is much easier to search data. In addition, if metadata are stored as RDF then it is possible to query datasets from many sources (for example "show all datasets that were published by Eurostat", or more complicated query "show all datasets that were published in 2013 by Eurostat and are related to demographics").

Functions:

- Spatial data metadata (ISO 19115)
- Spatial services metadata (ISO 19119)
- Dublin Core metadata (ISO 15836)
- Feature catalogue support (ISO 19110)
- OGC CSW 2.0.2 support (catalogue service)
- User defined metadata profiles
- INSPIRE metadata profile
- Web interface for metadata editing
- Multilingual (both user interface and metadata records). Currently 16 languages supported. It is possible to dynamically extend the system for other languages.
- Context help (multilingual)
- Import from the following metadata formats are supported:
    - FGDC CSDGM,
    - ISO 19139,
    - OGC services (WMS, WFS, WCS, CSW)
    - Feature catalogue XML
- Export – ISO 19139, GeoRSS, HTML, PDF, JSON, GeoRSS, Atom, KML, OAI_PMH and OAI_MARC2
- Support of thesauri and gazetteers.
- Display of changes with GeoRSS
- Template base interface with possibilities to change according to user requirements
- Possibility of deep cooperation with any of map clients for display of on-line map services

The MICKA instance provides the following functionality:

- editing and validation of INSPIRE metadata,
- import metadata from existing services like WMS, WFS, CSW, SOS and their editing and validation against the INSPIRE profile,
- advanced metadata search by several criteria,
- full discovery catalogue, supporting multi catalogue search.

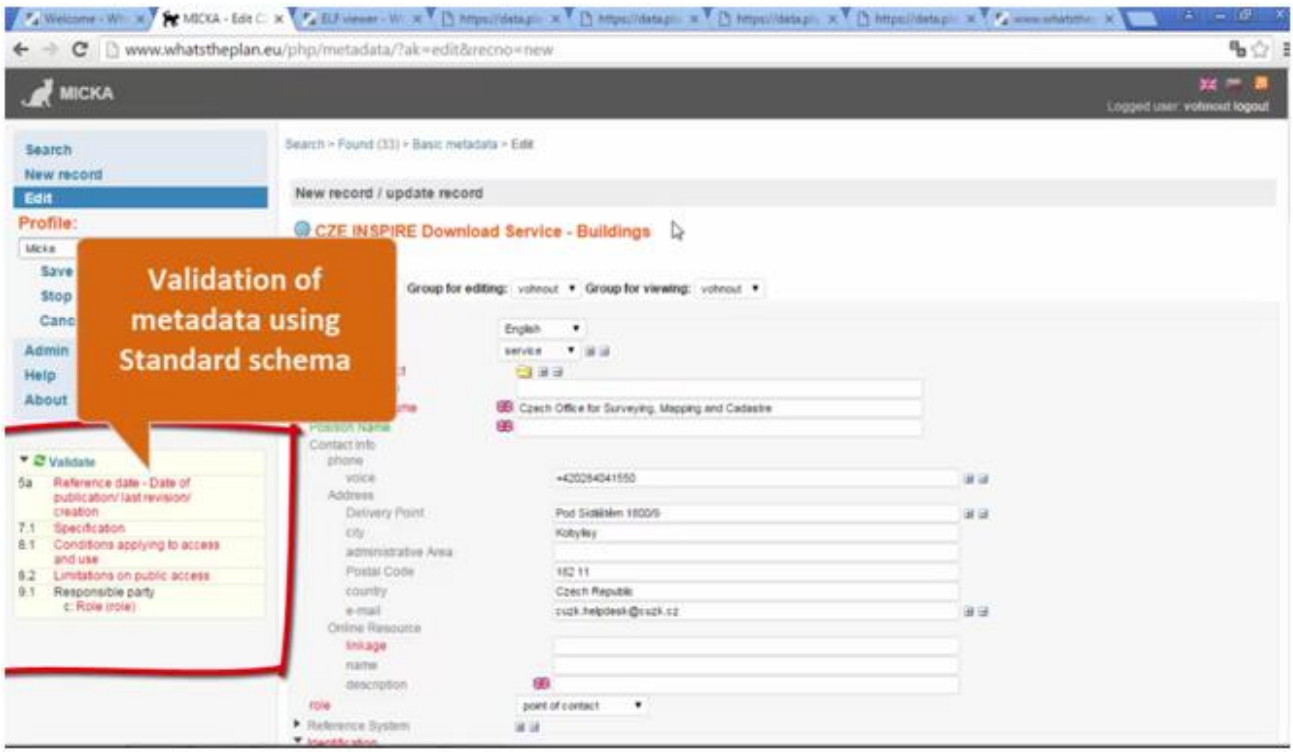

Figure 7: The Figure shows MIcKA UI for creating metadata records.

3.5 OGC Servers

3.5.1 Geoserver

GeoServer is an open source software server written in Java that allows users to share and edit geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards.

Being a community-driven project, GeoServer is developed, tested, and supported by a diverse group of individuals and organizations from around the world.

GeoServer is the reference implementation of the Open Geospatial Consortium (OGC) Web Feature Service (WFS) and Web Coverage Service (WCS) standards, as well as a high performance certified compliant Web Map Service (WMS). GeoServer forms a core component of the Geospatial Web.

Geoserver is being used mainly by Layman but it can be used also as standalone to provide posibility to share spatial data directly by advanced GIS users.

3.5.2 MapServer

MapServer is an Open Source geographic data rendering engine written in C supported by OSGeo. Beyond browsing GIS data, MapServer allows you create “geographic image maps”, that is, maps that can direct users to content.
Most important features of MapServer:

- Advanced cartographic output
    - Scale dependent feature drawing and application execution

    - Feature labeling including label collision mediation
    - Fully customizable, template driven output
    - TrueType fonts
    - Map element automation (scalebar, reference map, and legend)
    - Thematic mapping using logical- or regular expression-based classes
- Support for popular scripting and development environments
    - PHP, Python, Perl, Ruby, Java, and .NET
- Cross-platform support
    - Linux, Windows, Mac OS X, Solaris, and more
- Support of numerous Open Geospatial Consortium (OGC) standards
    - WMS (client/server), non-transactional WFS (client/server), WMC, WCS, Filter Encoding, SLD, GML, SOS, OM
- A multitude of raster and vector data formats
    - TIFF/GeoTIFF, EPPL7, and many others via GDAL
    - ESRI shapfiles, PostGIS, ESRI ArcSDE, Oracle Spatial, MySQL and many others via OGR
- Map projection support
    - On-the-fly map projection with 1000s of projections through the Proj.4 library

4 ENABLERS FOR MOBILITY AND SENSORS UPDATE

4.1 SensLog

SensLog is software component for sensor data collecting, storing and publishing in the web. SensLog consists of database model and server-side application. Relational database model is implemented in PostgreSQL database system with PostGIS extension. Database model contains procedures and functions to process and manipulate with data in transactional form. Especially several triggers prevents data integrity. Database model was developed as universal model both for mobile and static sensors. It can be used for storing data from mobile devices (e.g in scenario Human-as-Sensor).

Server-side part of the SensLog contains most of the application logic. Diagram on figure 4.1 shows simple overview schema of the SensLog. Sensor data are pushed to SensLog from sensor networks (in fields or smart devices) through RESTful API. Receiver module contains check mechanism to prevent insertion of data in incorrect form. Data consumers can receive data by one of the interfaces after authentication. There are several services to provide not only measured data or status data of the network but also structure of sensor network itself.

Users can use Web GUI that allows visualization of measured data in form of simple charts and maps. Main access to sensor data is provided by RESTful or SOS interface and it is expected further processing in other clients or applications.

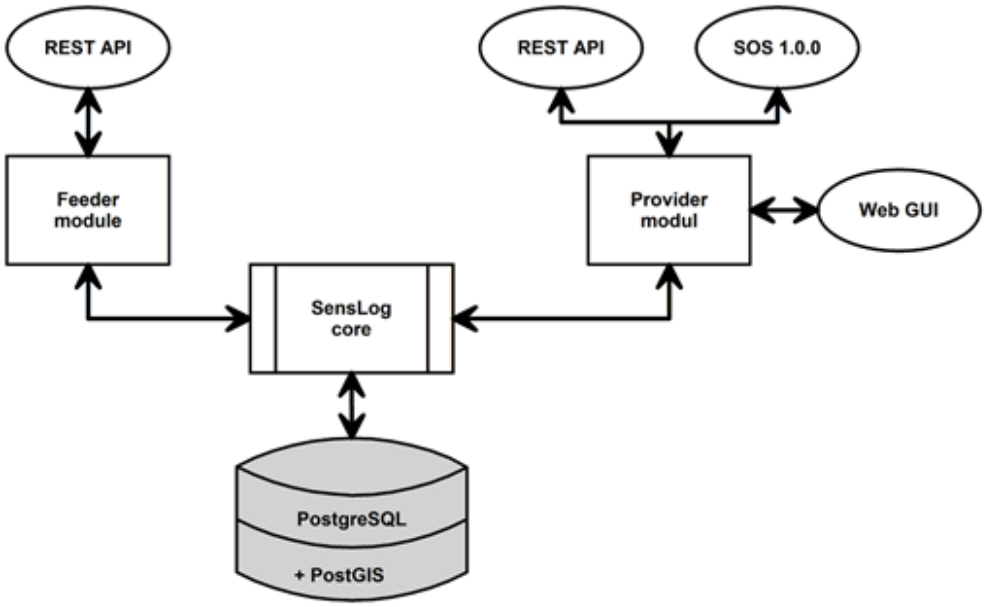

Figure 8: Overview diagram of SensLog structure

SensLog contains simple GUI for visualization of measured data on the Web. Web GUI contains visualization of unit positions and visualization of current values in sensor network and history of values for selected time span for specified sensor.

Teplota vzduchu	Vlhkost vzduchu	Rosný bod	Tlak vzduchu	Rychlost větru (min)	Rychlost větru (ø)	Rychlost větru (max)	Směr větru (min)	Směr větru (ø)	Směr větru (max)	Déšť akumulace	Déšť intenzita	Déšť trvání	Kroupy akumulace	Kroupy intenzita	Kroupy trvání
2.5 °C	65.400002 %	-3.132891 °C	971.900024 Pa	0.4 km/h	1.5 km/h	1.8 km/h	248 °	303 °	277 °	0 mm	0 mm/h	0 s	0 zás/cm²	0 zás/cm²/h	0 s
11.02.2016 21:30:02	11.02.2016 18:50:02	11.02.2016 20:45:02	11.02.2016 20:00:02	11.02.2016 19:50:02	11.02.2016 21:45:02	11.02.2016 18:20:02	11.02.2016 21:25:02	11.02.2016 19:15:01	11.02.2016 19:30:02	11.02.2016 21:25:02	11.02.2016 20:45:02	11.02.2016 21:40:03	11.02.2016 21:00:04	11.02.2016 21:40:03	11.02.2016 19:25:02

Počet dnů historie: 7 30 90 365

Figure 9: Example of current observed values of selected unit

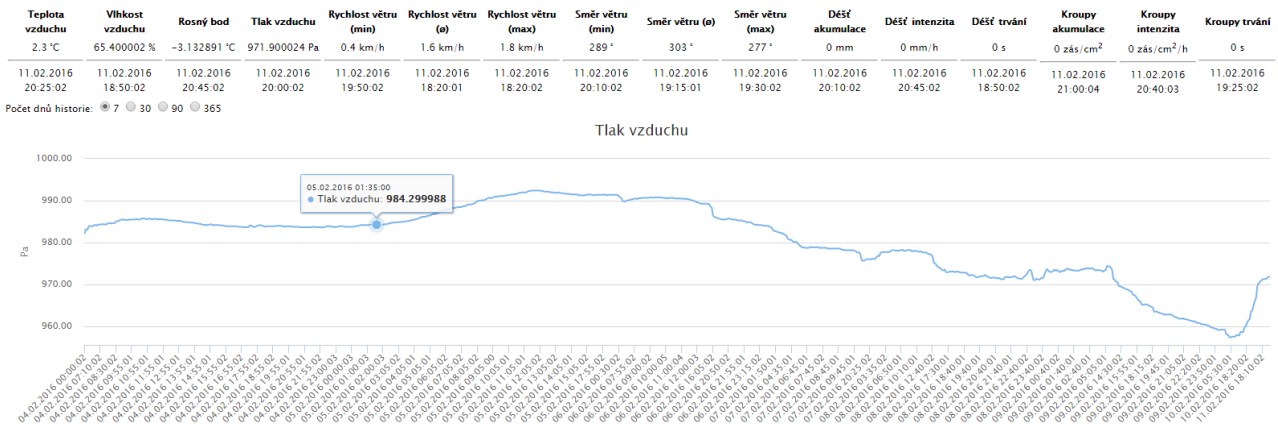

Figure 10: Example of 7-day history of air pressure sensor of selected unit

Important features of current version of SensLog are:

- Collecting sensor data from both mobile and static sensors
- Collecting observations in numerical data type, positions of sensors in WGS-84 coordinates, alert events occurred during measurement
- Storing data in relational database model, prepared for long time series by partitioning of most used tables observations and positions
- Publishing stored data in raw or preprocessed form, running predefined analyses on observations
- Publishing data in JSON format, by OGC SOS 1.0.0 service, or in form of chart visualization

4.2 IoT Discovery

IoT Discovery is Generic Enabler from FIWARE group. The IoT Discovery allows discovery of IoT objects, by providing a repository to register Things, Resources, and Devices, using semantically-annotated Descriptions based on the Internet of Things Architecture ontology models. The IoT Discovery uses Sense2Web IoT Linked Data Platform, which provides a repository for the CRUD (Create, Read, Update and Delete) management of semantic IoT descriptions, that complies with the IoT-A ontology models. Sense2Web can also associate different IoT object ontologies to domain data and other resources on the Web using Linked Open Data.
The IoT Discovery provides a set of interfaces a user can interact with. The first is a Web User Interface whereby a user can perform CRUD operations on the IoT Descriptions, and also query the IoT Descriptions as well. When registering or updating, a user can either upload an IoT Description or complete a form which is then sent to the server to be converted to RDF, and storing it in the RDF database.
The second interface is a RESTful CRUD and SPARQL interface. This interface mainly supports M2M interactions. An application can also perform CRUD operations on the IoT descriptions in the repository, and query for a particular piece of information from the descriptions using SPARQL.

Register
The current ontologies that are supported are the IoT-A ontologies that define a Resource, Entity and Service. Registering can be done either by uploading a description file or by completing a form. The form will not be accepted unless an ID, Name and Latitude/Longitude coordinates are at least entered. Once the form is submitted the page will return links for viewing the RDF result of the submission in various formats i.e. RDF/XML, RDF/XML-ABBREV,RDF/JSON, N3, N-TRIPLE and TURTLE.

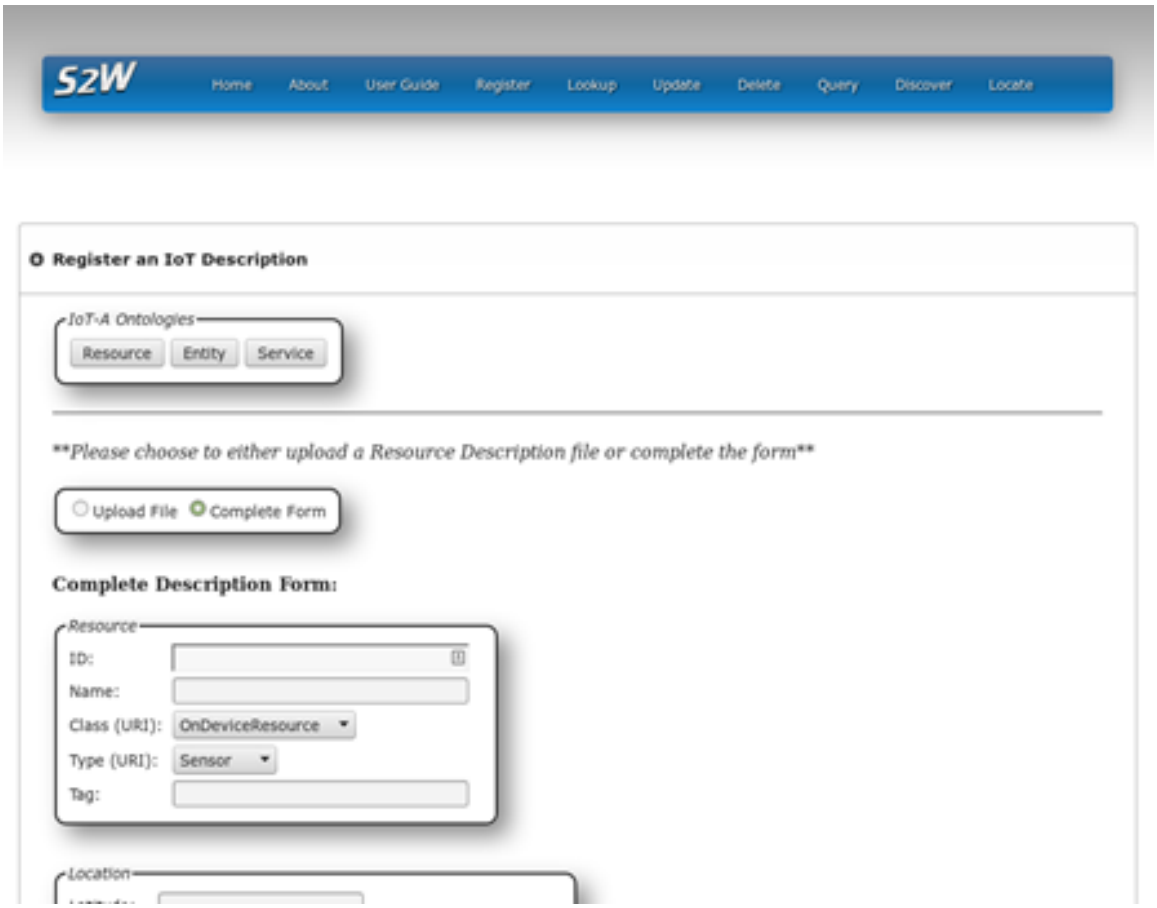

Figure 11: Register form example

Update

The Update operation is similar to the Register with the exception that the current values for a description can be retrieved and populated into the fields by entering the ID of the description in question, and clicking on the "retrieve" button.

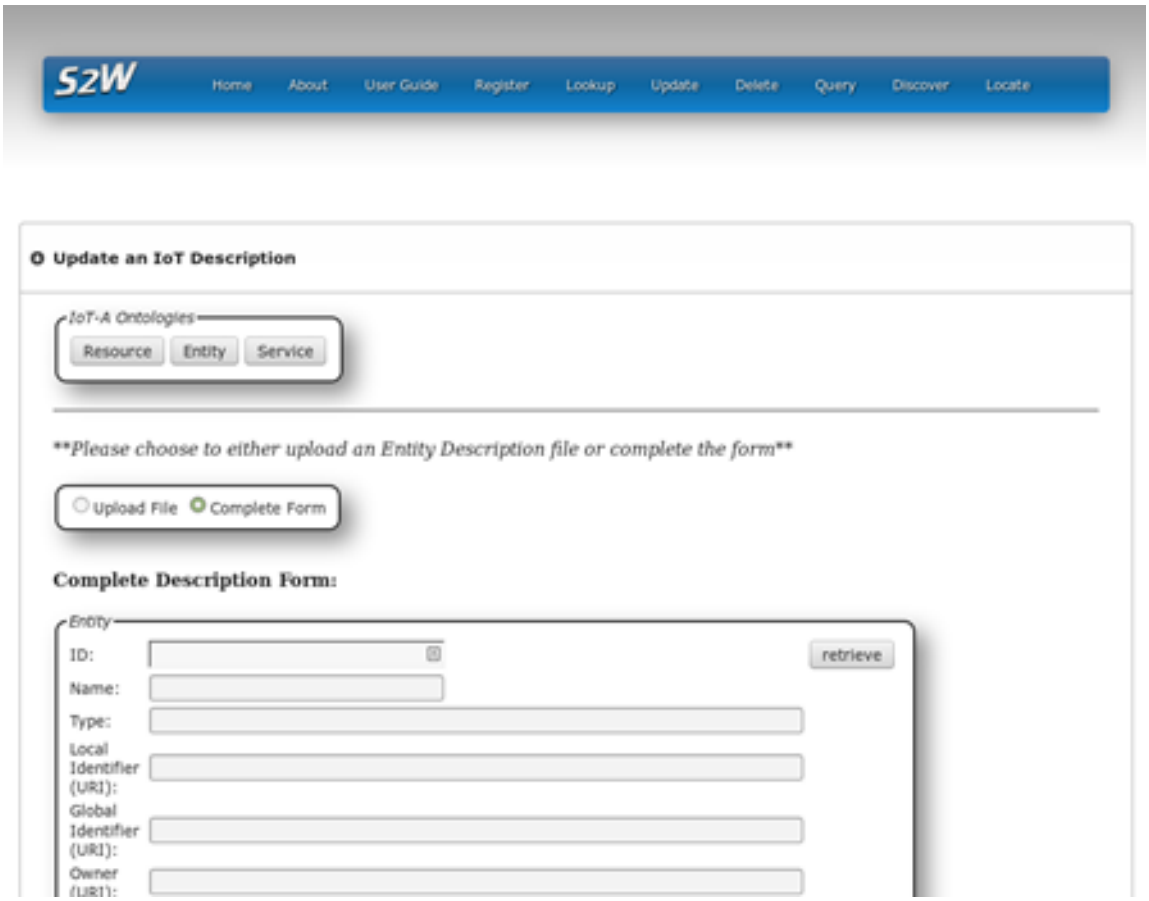

Figure 12: Update form example

Query

The platform supports SPARQL for querying of descriptions. When choosing a particular type of description, the respective SPARQL template is provided for a user to use and edit.

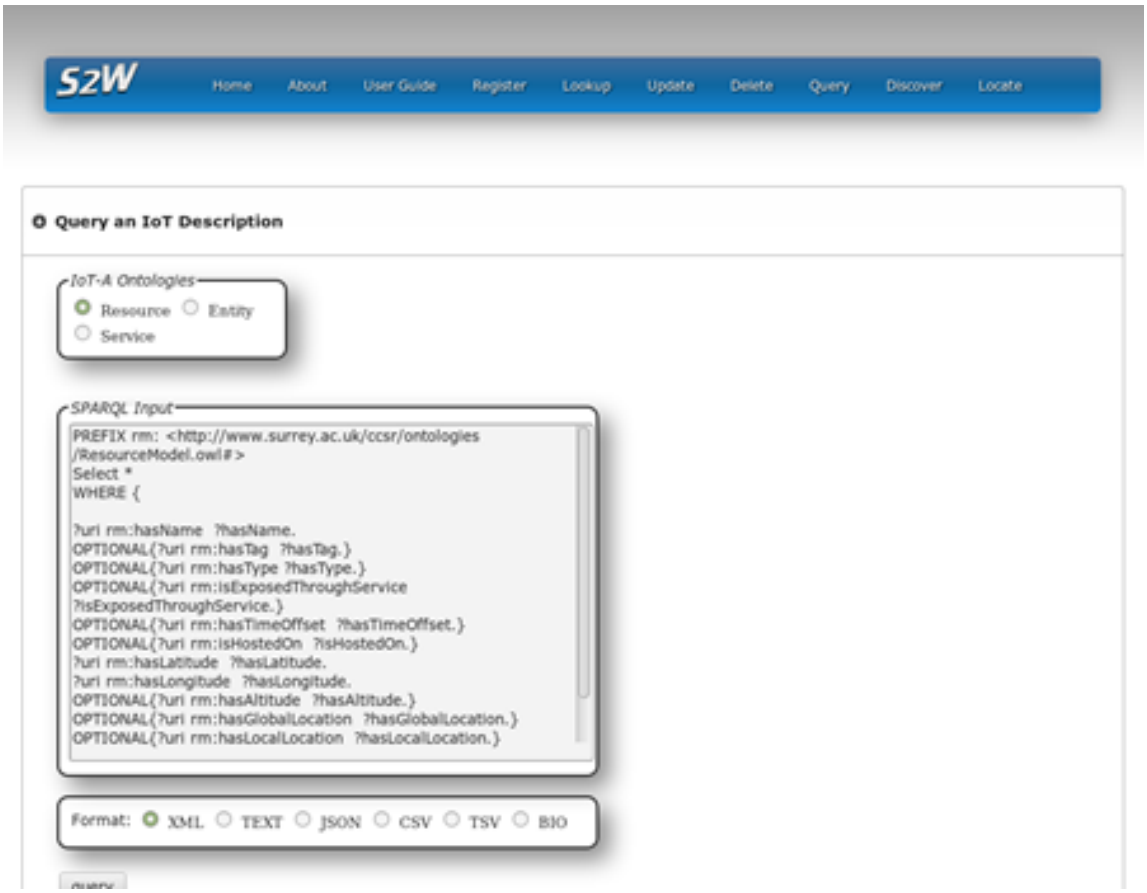

Figure 13: Query form example

Main features of the IoT Discovery enabler at this version are:

- Registering metadata of sensor data producers in RDF format or by form
- Querying and searching for registered sensors in the catalogue by GUI operations or by SPARQL queries

5 OPEN API UPDATE

The Open API consists of a set of server components and web services deployed as part of the platform. In the following text, elements of the API is referred to as ‘services’.

5.1 Web Feature Service

Implemented by ‘GeoServer’. Described under enablers in chapter 3.5.1

5.2 Web Map Service

Implemented by ‘MapServer’. Described under enablers in chapter 3.5.2

5.3 Web Catalog Service

Implemented by ‘Micka’. Described under enablers in chapter 3.4

5.4 Authentication Service

The authentication service used by SDI4Apps specific clients and is used by specific enablers ‘Layman’ (described under enablers in chapter 3.2) and ‘HSLayers-NG’ (described under enablers in chapter 3.1)

5.5 Data Management Service

The data management service is used by SDI4Apps specific client tool ‘LayMan’ that permits upload and publication of GIS data files to a SDI4Apps platform instance in the ‘cloud’.

5.6 Routing Service

The routing service is implemented as a combination of the basic enablers ‘PostgreSQL’ and ‘PostGIS’ with the add-on component ‘pgRouting’.

PostGIS extends PostgreSQL with methods corresponding to the OGC Simple Features Specification methods for creating, accessing and querying simple geometries. pgRouting extends PostGIS with a custom set of methods for network analysis and implements a wide range of well-known routing algorithms including Dijkstra, A-star, Johnson’s, Floyd-Warshall’s etc.

All these methods operate in real-time on a network of edges and nodes stored in PostgreSQL tables. The data structure can be used on ‘any’ data - but well-proven data ingestion processes exist for OpenStreetMap downloads. This makes the service versatile as it can easily be added to a new route network.

Real-time network calculations are expensive and not suitable for GoogleMaps style ‘drag and drop routes’ as the processing time for a single query can extend to several seconds. The use case for real-time network calculations is for transport management and planning tasks where it is necessary to simulate flow-patterns, accessibility and distance matrices based on manipulating the underlying transport network.

5.6.1 How the web service has been implemented

The web service is implemented in Java as a Servlet and is installed in the SDI4Apps Java Servlet container 'Apache Tomcat'.
It provides a web service end-point that implements four public methods:

- GetNearestNode - route calculations take place between nodes in the network and in order to issue a successful query it is necessary to identify the ids of the start, end and potentially via nodes that the query should include. This method accepts as input a longitude, latitude pair of coordinates (WGS84) and a search radius tolerance defined in meters.
- GetShortestPath - given a start node and an end-node, this method calculates the shortest directed path (taking into consideration turn-restrictions) between them on the underlying graph of nodes and edges. The method returns a list of segments and their length/attributes.
- GetReachableArea - given a start node and either a time/speed or distance in meters, this method calculates the reachable area and returns this as a polygon.
- GetOptimalRoute - given a start node, an end-node and a set of nodes, this method calculates the shortest route to travel between all of the nodes and returns a result similar to that of GetShortestPath

The servlet implements non-caching CORS headers in the HTTP response so that the service may be accessed from any JavaScript client.

I.e. it sets the HTTP header: 'Access-Control-Allow-Origin: *' as well 'Pragme: no-cache' and a few others that secure that the request will always be reissued on the dynamic dataset and not returned from a proxy or browser cache.

This has a utility when calling the service from the SDI4Apps client-side JavaScript API library which like other JavaScript APIs can be installed on any hosting environment while consuming spatial processing capabilities from an SDI4Apps platform instance.

5.7 Advanced visualization API - WebGLayer

WebGLayer is an open source javascript API (BSD licence) dedicated for advanced visualization of large, multidimensional, geospatial dataset. The library is based on WebGL and thus uses GPU for fast rendering and data filtering.
Current version support these visualization techniques:

- Histograms
- Point Symbol map
- Heat map
- Parallel coordinates

The library supports these interactions:

- Filter by attribute (brush&link)
- Filter by polygon (polybrush)
- Filter by the level of density

The library is capable to visualize hundreds of thousand of records that have multiple attributes and a spatial location. The advanced filtering and rendering technique enable fast interactions where the filtering in one view update all the other views within the time below 60ms.

Within the SDI4Apps this library has been extended by API that is integrated within the SDI4Apps platform. Furthemore the API was integrated with HSLayers-NG (the mapping API). The example of advanced visualization is depicted in Fig14.

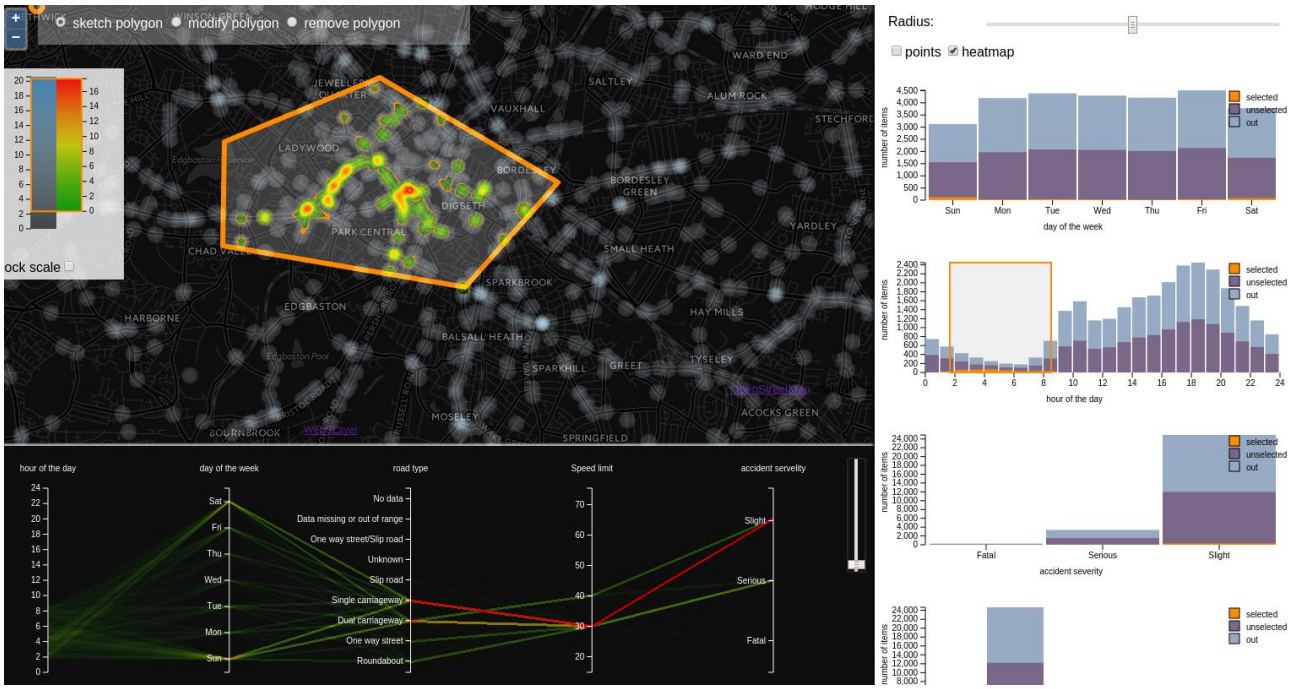

Figure 14: The Figure shows coordinated multiple views combining the heat map, parallel coordinates and histograms

5.8 Search Service

This service corresponds to the information retrieval module of the client-side JavaScript library)

In this module we provide the implementation of a set of services for data access, both trough REST API and a helper javascript library, which exposes REST API as methods with same name and signature.
Notice that services implement a REST API, with JSON response format, to retrieve specific records of data from datasets, so that functionality are directly available in any client, e.g. an end-user (developer) javascript application, without any needs of an intermediate client layer such as the helper javascript library.
Implemented services support these operations:

- Attribute based search
- Location based (geographic) search
- Full text search
- Mixed (alphanumeric and geographic) search
- Entity retrieval
- Metadata retrieval

The services implemented are:

1. full text search à searchDigitalLocationsByText
2. attribute/geometry criteria à searchDigitalLocations
3. categories retrieval (“category” is a metadata which relate an object to a dataset) à GetCategories
4. object retrieval à GetDigitalLocationByIds

The typical usage pattern is the following:

- retrieve all category (which are related to dataset classification)
  search objects applying filter on category, geometry (bounding box), attribute values (e.g. full text)
- retrieve short object description, along with object ID
- retrieve full object description trough object ID obtained at previous step.

5.8.1 Full text search (searchDigitalLocationsByText)

Search for a text among objects specifying a bunch of criteria. All unspecified parameters will be ignored. Results will be ordered and paginated.

Parameters

Name	Type	Description	Notes
boundingBox	BoundingBox	bounding box coordinates	Returned object must be placed into the given bounding box (x1 y1 x2 y2 coordinates separated by a space)
bufferGeometry	String	The area into which the object must be placed, using WKT coordinates	
bufferRadius	Long	The tolerance distance from the selected geometry	
categoriesId	List(Long)	Categories among which compute the search	Returned object must belong to almost one of the specified categories
text	String	Search text	
states	List(DigitalLocationState)	object's valid states	returned object must be in one of the given states
ordC	Integer	Specifies the ordering: 1. for attribute value 2. for distances from a point 3. for dl creation date 4. - 5. for dl modification date 6. dor dl source type	
ordV	String	Order by orderByCreteria parameter	

		value — Ordinamento basato sul valore del parametro orderByCriteria: 1. Id of the attribute whose value will be used for ordering. If an object doesn't contains a value for that attribute, it's value will be put to null and the digital location will be put in the end of list. In there is more than a value for that attribute in a object, will be considered only the first. In case of a tie, is ordered by primary key. 2. starting point to check the distance (x and y coordinates separated by a space)	
ordD	Integer	Ascending (1) or descending (-1) ordering	default=ascending
numResultsPerPage	Integer	Number of results for each page	
numPage	Integer	Number of the page to return	
idProfile	Integer	API profile to use (low=0, high=1)	default=low
apiKey	String	Api_key for application's authentication	required

Table 1: Full text search

Here is an example:
http://sdi4apps.hyperborea.com/api/SearchDigitalLocationsByText?apikey=apikey2&text=Uffizi

5.8.2 Search criteria (searchDigitalLocations)

Searches objects according to a set of criterion.

Parameters

Name	Type	Description	Notes
boundingBox	BoundingBox	bounding box coordinates	Returned object must be placed into the given bounding box (x1 y1 x2 y2 coordinates separated by a space)
bufferGeometry	String	The area into which the object must be placed, using WKT coordinates	
bufferRadius	Long	The tolerance distance from the selected geometry	
attId	List(Long)	The ordered list of attribute ids on wich compute the search	
attVal	List(String)	The ordered list of attribute values on wich compute the search (the search will be done by substring and is not case sensitive)	
operator	String	Logic operator between attributes conditions (“and” or “or”)	
categoriesId	List(Long)	Categories among which compute the search	Returned object must belong to almost one of the specified categories
states	List(DigitalLocationState)	object’s valid states	returned Digital Location must be in one of the given states
ordC	Integer	Specifies the ordering: 1. for attribute value 2. for distances from a point 3. for dl creation date 4. - 5. for dl modification date 6. dor dl source type	
ordV	String	Order by orderByCreteria parameter value — Ordinamento basato sul valore del parametro orderByCriteria:	

		1. Id of the attribute whose value will be used for ordering. If an object doesn't contains a value for that attribute, it's value will be put to null and the digital location will be put in the end of list. In there is more than a value for that attribute in a object, will be considered only the first. In case of a tie, is ordered by primary key. 2. starting point to check the distance (x and y coordinates separated by a space)	
ordD	Integer	Ascending (1) or descending (-1) ordering	default=ascending
numResultsPerPage	Integer	Number of results for each page	
numPage	Integer	Number of the page to return	
idProfile	Integer	API profile to use (low=0, high=1)	default=low
apiKey	String	Api_key for application's authentication	required

Table 2: Search criteria

Here is an example:
http://sdi4apps.hyperborea.com/api/SearchDigitalLocations?apikey=apikey2

5.8.3 Categories retrieval (GetCategories)

Returns the categories list for objects.

Parameters

Name	Type	Description	Notes
idProfile	Integer	API profile to use (low=0, high=1)	default=low

apiKey	String	application's authentication key	required

Table 3: Table Categories retrieval

Here is an example:
http://sdi4apps.hyperborea.com/api/GetCategories?apikey=apikey2

5.8.4 Object retrieval (GetDigitalLocationByIds)

Get a list of objects from their identifiers.

Parameters

Name	Type	Description	Notes
digitalLocationId	List(Long)	Object identifiers	required
idProfile	enum (high, low)	API profile to use	default=low
apiKey	String	api_key for application's authentication	required

Table 4: Table Object retrieval

Here is an example:
http://sdi4apps.hyperborea.com/api/GetDigitalLocationsByIds?apikey=apikey2&digitalLocationId=10092&digitalLocationId=10050

5.9 Sensor Data Service

The sensor data service has been implemented using 'SensLog' and 'IoT Discovery' components and is described under enablers in chapters 4.1 and 4.2.

5.10 Feature Synchronization Service

The feature synchronization service is implemented as a combination of the basic enablers 'PostgreSQL' and 'PostGIS'.
It allows an application to check-out data for an area from a PostGIS table, i.e. a GIS feature layer that is either a point, line or polygon layer. The checked out data are available as a GeoJSON dataset that can be visualized and/or edited using any standards compliant technology. The only requirement of the source data table is that it must include a unique identifier that can be used to resolve changes once data are checked back in. At the time of checkout, a timestamp is added to each feature.

Once the file has been edited it can be checked back in by passing the modified file along with the original file back to the web service. If there are no conflicts with other, concurrent edits, the changes are written directly back into the PostGIS table, if conflicting changes have been made, the changes will be returned to the client and must be resolved by issuing additional calls stating which of two or more conflicting edits 'wins'.

The typical use-case is to download an extract of a larger dataset. Edit it using an Openlayers based client.

5.10.1 How the web service has been implemented

The web service is implemented in Java as a Servlet and is installed in the SDI4Apps Java Servlet container 'Apache Tomcat'.

It provides a web service end-point that implements four public methods:

- CheckOut - Returns a GeoJSON object to the client based on a supplied layer identifier (identifying the PostGIS table to select from the database), the name of the unique id field and a bounding box (an object exposing the properties minX, minY, maxX, maxY) in lon/lat WGS84
- CheckIn - Passes the original and modified GeoJSON object back to the server, detects conflicts and executes updates. Returns a list of conflicts, if any. Each conflict has a description, two GeoJSON features with geometry and properties as well as a unique conflict ID.
- GetConflicts - Returns unresolved synchronization conflicts for a layer identifier
- Resolve - Based on a conflict ID determined which one of the conflicts are to be resolved.

The servlet implements non-caching CORS headers in the HTTP response so that the service may be accessed from any JavaScript client. As described above in chapter 6.6, this has utility for the SDI4Apps client-side JavaScript library.

5.11 Tile Data Service

The tile data service enables users to download pre-generated tile data files in MBtiles format. The use case for the service is to download offline background maps for use in mixed-connectivity mobile apps.

Instead of 'polluting' the local file system on the device with a very large number of image files, data are bundled into a SQLite based format called MBTiles. Tiles can be generated using tools like the excellent Open Source TileMill and then uploaded to the service. Predefined cartography templates for OpenStreetMap data dumps exist and make it very easy to create custom tiles for new areas.

An initial concept of real-time tile generation was abandoned due to scalability issues and high server-side CPU usage that would render this feature impermissible in a shared execution environment.

The servlet implements non-caching CORS headers in the HTTP response so that the service may be accessed from any JavaScript client. As described above in chapter 6.6, this has utility for the SDI4Apps client-side JavaScript library.

5.11.1 How the web service has been implemented

The web service is implemented in Java as a Servlet and is installed in the SDI4Apps Java Servlet container 'Apache Tomcat'.

It provides a web service end-point that implements three public methods:

- Upload - Adds a new MBTiles file to the SDI4Apps platform instance
- List - List all available MBTiles files in the SDI4Apps platform instance
- Download - Download an MBTiles file from the SDI4Apps platform instance

The servlet implements non-caching CORS headers in the HTTP response so that the service may be accessed from any JavaScript client. As described above in chapter 6.6, this has utility for the SDI4Apps client-side JavaScript library.

5.12 Extended Storage Services

At present, only one extended storage service has been added to the platform, namely a triple store, Virtuoso. Virtuoso is described under chapter 3.3.

5.13 Custom Data Service

In order to provide persistent server-side storage for JavaScript applications, the SDI4Apps platform implements a simple custom data service that permits users to store arbitrary JSON data.

An object has a type - and a type belongs to an application. That way, information can be grouped into ‘type collections’ equivalent to tables and isolated from other information by ‘applications’ equivalent to databases.

This simple data service enable HTML5 applications to store arbitrary application data and information in the SDI4Apps platform without having to manually set up a data structure on the server side, instead data types can be declared as and when needed - and even dynamically.

5.13.1 How the web service has been implemented

The web service is implemented in Java as a Servlet and is installed in the SDI4Apps Java Servlet container ‘Apache Tomcat’.

It provides a web service end-point that implements ninepublic methods:

- CreateApplication - Creates a new application handle that the data will be attached to
- ListApplications - Lists all available application handles in the data
- CreateType - Creates a ‘type’ that can be used to group together objects that share the same data model.
- ListTypes - Returns all available ‘types’ in the SDI4Apps platform instance
- AddObject - Adds a new object of type to an application
- UpdateObject - Updates an existing object with specified ID (replaces the entire JSON object)
- DeleteObject - Deletes an existing object with specified ID
- GetObject - Retrieves an existing object with specified ID
- QueryObject - Return all objects within application of type that optionally satisfies a query expression

The servlet implements non-caching CORS headers in the HTTP response so that the service may be accessed from any JavaScript client. As described above in chapter 6.6, this has utility for the SDI4Apps client-side JavaScript library.

6 CONCLUSION

The Updated platform available as virtual server is now ready for utilization by pilots and external users. First test started on Riga Hackathon, where separate instantiation were easy deploy for pilots.

Virtual servers are now important part for future business modeling, because it was considered as one from potential Open Source product, where support to potential users will be one from potential future incomes. There is necessary to have strong focus to reach level of commercial services.