DOI 10.12694/scpe.v19i2.1350
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 19, Number 2, pp. 181–187. http://www.scpe.org

SOLUTIONS FOR DATA DISCOVERY SERVICE IN A VIRTUAL RESEARCH ENVIRONMENT *

VLADIMIR DIMITROV† AND STILIYAN STOYANOV‡

Abstract. Scientific computing requires many and large volumes of complex structured data and metadata that are scattered across data centers. Researchers find it difficult to discover the specific data they need for their research. Traditional search engines, such as Google, are not effective in most of these cases. Moreover, some of the scientific data are confidential and are not publicly indexed. Therefore, it is necessary to develop a custom solution that satisfies the researchers' need for flexible search. This article introduces the Data Discovery Service designed to serve the Virtual Research Environment (VRE) during the VI-SEEM project. The solution is based on a specially configured and upgraded version of the CKAN platform. The main installation procedure is described, as well as the authors' contributions for the purpose of regularly harvesting data from different sources and updating the available content on which user queries are to be executed.

Key words: data service, data harvesting, search engine, Virtual Research Environment

AMS subject classifications. 68P20

1. Introduction. Nowadays, researching often requires the processing and analysis of many and large volumes of data and metadata. One of the frequent difficulties for researchers is to find exactly the data they need for the analysis. Usual and freely available search engines are seldom useful because scientific data is highly specialized and complexly structured. We will present the VI-SEEM Data Discovery Service (VDDS), which is a solution to search scientific data and it is developed during the VI-SEEM project [1]. The main purpose of this project is to create a useful virtual environment for the researchers located in Southeast Europe and the Eastern Mediterranean. Researchers are organized into thematically targeted communities which are: Climate, Life sciences and Digital Cultural Heritage.

The VRE (Virtual Research Environment) consists of several services which are available to the scientific users [2]. The services for VI-SEEM project are integrated in a public catalog which is accessible at this link: [3]. There are three main groups of services: Data storage, Computing, Application specific and Authentication/ Authorization. The presented Data Discovery Service belongs to Data storage group.

There are several popular platforms for hosting and searching across a variety of metadata, Open Data Portals that are mainly used in scientific areas. We looked at the most commonly used ones, such as DSpace, CKAN, Zenodo and Figshare [4] [5] [6] [7]. Our main goal is to have a flexible, easily upgradable search service for scientific metadata, with a rich and well-documented API (Application Programming Interface). Other requirements we have are that the chosen platform must be long-established, open source and has big community. Zenodo [8] and Figshare [9] dropped out of our choice because they are mainly focused on processing scientific papers and documentation. However, we need a platform that hosts and performs search in any metadata, while offering a convenient editing interface. DSpace [10] is designed to create and manage repositories for large volumes of diverse data, and there is also a search engine that is not as flexible in complex metadata. In addition, it is already used to host the project's main repository as a part of the VRE and therefore it has been dropped as a choice and finally we selected CKAN [11].

We present the main points of the installation and configuration of the CKAN platform, which is the basis of the VDDS and the additional resources such as Data Synchronization Tool that are developed and added to form the complete solution. Primary performance tests and evaluation are presented too.

2. Data Discovery Service design and implementation. As a basis for our solution, the CKAN open source platform is chosen. It is a comprehensive system with rich functionality for building large and complex data sites, including a flexible search engine. CKAN has convenient, easy to operate interface on one hand and very rich API on the other. Having user friendly interface is very important because the target users are

*This work is supported by EC programme H2020, Grant Agreement No 675121, VI-SEEM Project.

†Institute of Information and Communication Technologies - Bulgarian Academy of Sciences, Bulgaria (vgd@acad.bg)

‡Institute of Information and Communication Technologies - Bulgarian Academy of Sciences, Bulgaria (sstoyanov@parallel.bas.bg)

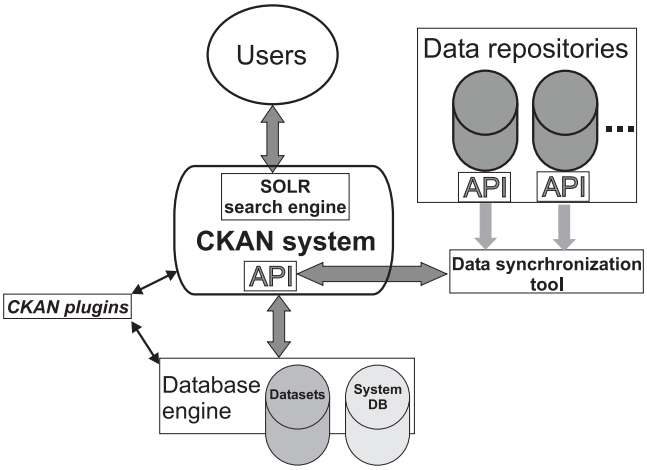

FIG. 3.1. *VDDS common architecture.*

researchers who are competent in their specialized field of study but not necessary an advanced IT proficient. Moreover, it provides well documented and supported API [12] and options for third party extensions (plugins). Many research, governmental and non-governmental organizations use CKAN for building and managing their open data sites.

On the back end the programs which form CKAN are written mainly in Python and uses Pylons web framework and SQLAlchemy [13], the database toolkit as its Object Relational Mapping (ORM). The front end is implemented in JavaScript. The database engine is the popular PostgreSQL.

In our case, special interest is SOLR [14], an open source enterprise search engine, written in Java, developed by Apache Software Foundation, which is an integral part of the system.

3. Software architecture. The common architecture of VDDS is depicted on Figure 3.1.

Customized **CKAN system** is the core of our VDDS. It is configured to support several organizations which correspond to the VI-SEEM project participants and three groups which correspond to the main research communities in the project: Climate sciences, Life sciences and Digital Cultural Heritage. Each organization and community can maintain metadata for their datasets. In addition there are common purpose communities for generic usage, software projects and testing purposes.

SOLR is the main search engine which is accessed by the end users.

API is the documented CKAN Application Programming Interface which is used internally by **Data Synchronization Tool** to harvest external metadata from **Data Repositories**. Also there are different **API**s for the external **Data Repositories**. This tool is written in Python.

Data repositories. These are external sites containing data that are of particular interest to us.

CKAN plugins. These are additional programming modules that extend the functionality of the system.

Database engine. Software components and storage space that hold and maintain the datasets.

4. Data Synchronization Tool. In order to search for data uploaded on the VI-SEEM Repository Service (VRS) or VI-SEEM Simple Storage Service (VSS) using the data discovery service (VDDS) data synchronization needs to be made of VDDS's database and the databases of the other services. More precisely this synchronization process is one-directional data flow from the source of the original data to the server running the data discovery service.

In order VDDS to be useful for its users, then the results of each search query have to be correct and complete. Therefore, this data flow process has to be executed regularly, i.e. the operation should be fully automated.

One proposed way to automatically extract the necessary metadata is via a harvester. DSpace [10], which is used to create the repository service, has such functionality implemented as OAI-ORE/OAI-PMH Harvester [15]. The main advantages of this solution are that no additional software is required to collect the data from

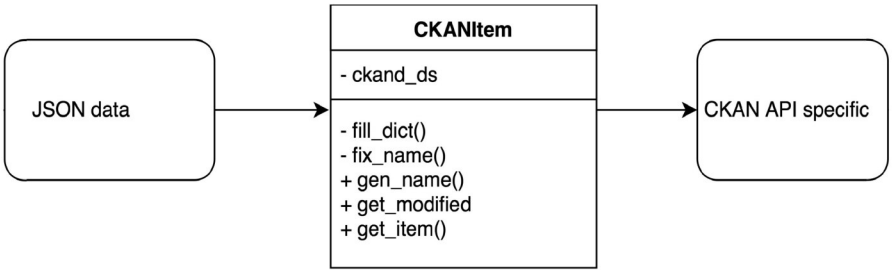

Fig. 4.1. *CKANItem core class*

VRS and also it is supposed that the procedure is more efficient. However, the output of the harvester's work usually is designed to be imported to similar interoperable repository or another DSpace software e.g. as a backup or redundant node part of more complex high availability system. It is not suitable format for our purpose and it needs significant transformations before it is ready to be uploaded on the data discovery service. Moreover, this solution is not very generic as it is very specific to the technology compiling VRS.

We decided to create our own software tool that solves the current problem and is universal to that degree so that it could easily be expanded in future to support more data sources if required. Accordingly, our solution must have modular design.

As the core of that solution one class is responsible for uploading data to VDDS and it is presented on Figure 4.1. Its role is to process some kind of generic data e.g. Java Script Object Notation (JSON) and return as output a specially formatted dictionary meeting the requirements of VDDS's CKAN based API. CKAN's Action API is RPC-style and has very rich functionality exposing all of CKAN's core features to API clients.

A function is implemented in another module that accepts the output and sends it to the data discovery service using the action call to the CKAN API package_create.

With these two modules available the task of synchronizing VDDS with any data repository is reduced to mapping the metadata tags and obtaining array of JSON formatted data.

Python version 3 is chosen as a programming language for implementing the tool. The main reason behind that decision is because of the fact that the CKAN software is written in Python and also has very good documentation and user guides with examples in that language on how to use the API. Also tools algorithm doesnt contain any compute-intensive part, neither execution time is of such importance to the performance of our system as the tests we conducted showed normal responsiveness and stable operation of the search service under greater than average system load. So using lower level programming language than Python to yield more efficient resource usage is not necessary at all. Although CKAN is written in the older version 2 of the language, we went for the newer version because it is actively supported by its core developers meaning that it is getting tha latest and better features. Third factor is that Python 3 interpreters are built in every Linux distribution available and deploying the tool doesn't require much effort.

VRS is a REST-compliant service and provides all the required operations to download the necessary metadata. We've made extensive tests of the load level and performance of the repository service when querying the REST interface because a total number of:

$$queries = 2\left(1 + \sum_i C_i\right) \tag{4.1}$$

where C_i is the number of collections of i-th community, are sent every time data is being synchronized and thus it is probably less efficient than harvester. But test results are very consistent and there are no signs of performance downgrade.

Figure 4.2 shows an example of the workflow of the tool when data is being synchronized between VRS and VDDS.

Dspacemetadownloader module executes the queries and does some additional conversion of the JSON as it is in nested format without unique keys that have to be combined before the JSON is passed to the

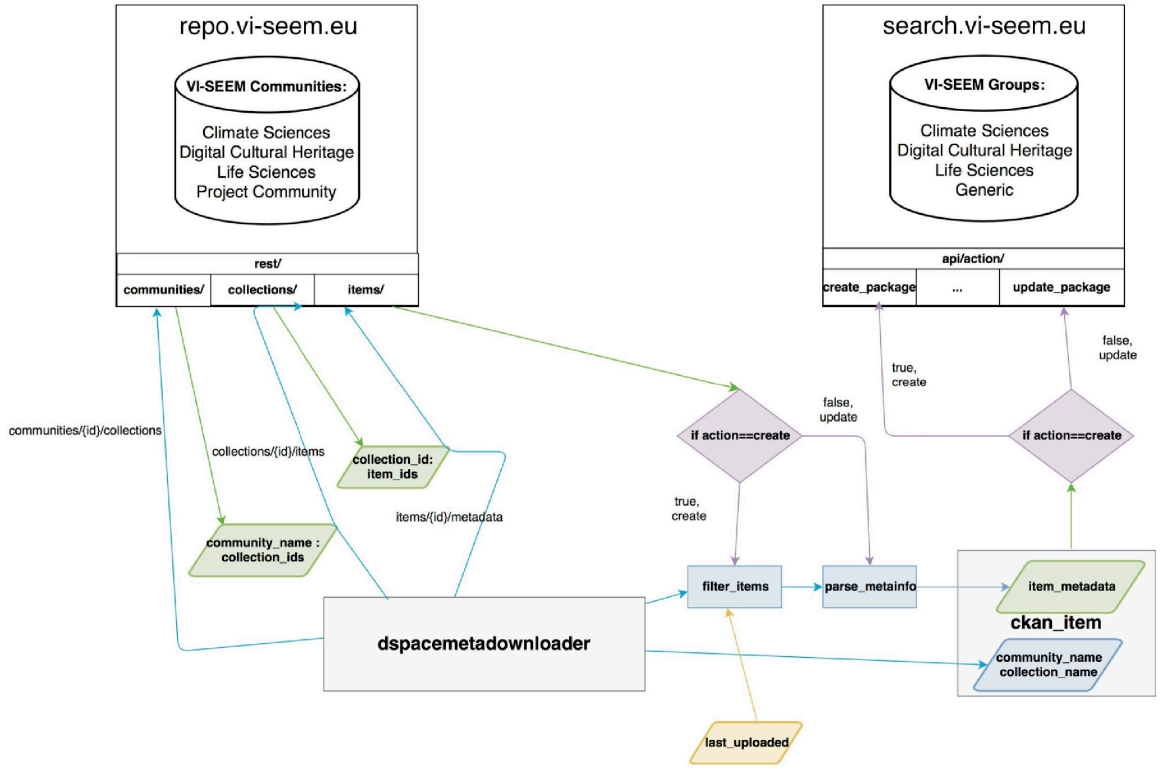

FIG. 4.2. *Workflow of the tool when synchronizing data between VRS and VDDS*

CKANItem class.

The tool runs in either create or update mode. Create is the default behavior and filters already synchronized items, uploading only new items to VDDS. If update mode is specified the metadata of all items will be checked and updated. Both modes operate absolutely independent of each other.

Unfortunately CKAN has very strict rules about the allowed characters uploaded using its "Action API". Hereof the main module **ckanitem** parses every metadata string and substitutes appropriately every unaccepted symbol. In case that such character is not caught by the module or for another reason the operation fails, unsuccessful upload is recorded in a log file with timestamp and error description.

5. Installation. The installation procedure of such kind of software is usually very tricky and capricious. We tried several different methods to install, including compiling the source code. Most of these methods had minor or big problems as of the middle of 2016 using the existing production versions of software components at this time. The only method that worked is recommended by the developers of CKAN and finished successfully. It requires OS Ubuntu 14, 64 bit, kernel 4.2.0-42-generic, Python 2.7.6, CKAN 2.5. For Data Synchronization Tool On the same machine Python 3 is installed too. Our experiences with other, even newer distributions and versions were painful.

The installation of Data Synchronization Tool is simple. It is a bunch of Python scripts which are copied in a directory and configured in **crontab** to be executed twice a day: around noon and around midnight.

Example screenshot of the VDDS front end is presented on Figure 5.1.

6. Solution evaluation. We conducted numerous load and performance tests after the installation of VDDS. For this purpose, we first need to import demo data in the database of the server [16]. These demo examples are available at the official source code repository of CKAN and contain many and diversified data. This is important, which is important for the tests that they are capable of generating significant stress on our server. For example, the dataset named Newcastle City Council: Payments over 500 contains over 23,000 rows

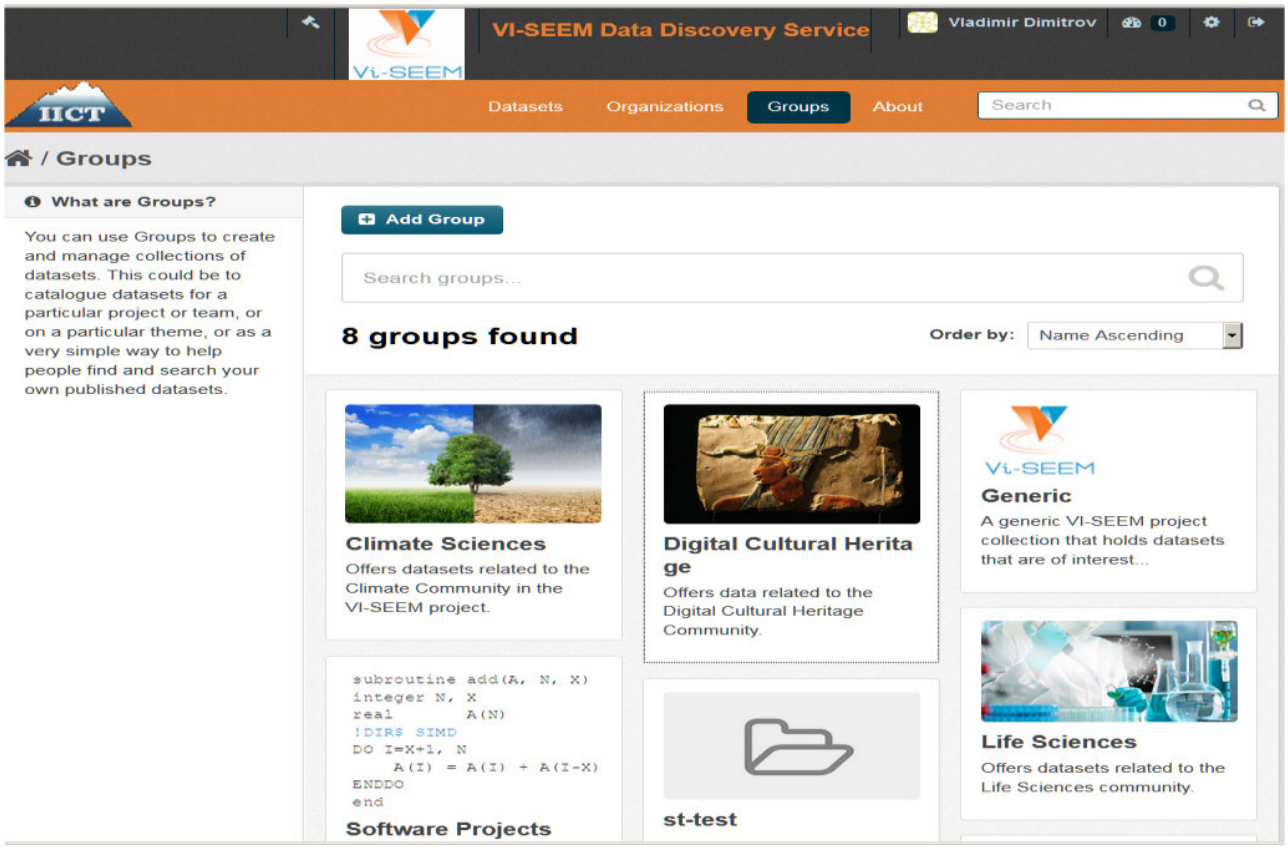

FIG. 5.1. *VDDS example screenshot. User groups section.*

TABLE 6.1
Averaged results obtained by running various requests during 60-second active connection to VDDS API.

	Latency (ms)				Requests per second				
No.	Avg	Dev	Max	Dev(%)	Avg	Dev	Max	Dev(%)	TX,KBs
1	32.21	12.65	179.85	97.24	31.94	5.79	40.00	66.33	18.61
2	111.48	14.90	247.78	96.88	9.19	1.69	10.00	89.80	429.63
3	133.82	15.74	264.15	97.11	7.95	2.60	10.00	67.41	367.15
4	67.40	11.10	217.30	97.11	14.82	5.09	20.00	46.86	354.97
5	655.05	33.43	757.82	89.01	0.96	0.20	1.00	95.73	38.66
6	655.68	31.88	772.43	91.21	0.94	0.24	1.00	93.87	40.10
7	610.92	31.70	747.05	90.82	0.97	0.18	1.00	96.63	37.74
8	70.83	17.04	313.68	96.44	14.26	5.04	20.00	52.90	366.99
9	268.53	25.50	485.26	94.26	3.20	0.63	6.00	76.44	116.36
10	108.40	65.87	504.06	66.84	54.44	7.67	60.00	92.62	492.12

in six CSV files for each month from September 2011 to February 2012. The tests use **wrk** program [17]. **wrk** is a HTTP benchmarking tool producing different levels of load on the server depending on the used command line arguments. We run it in multithreaded mode and adjust connection and duration parameters to tune the intensity of each test. Every additional connection simulates another active user working with VDDS. Produced outputs contain measurements of main performance metrics such as latency, bandwidth and number of requests.

Test results are shown in Table 6.1.

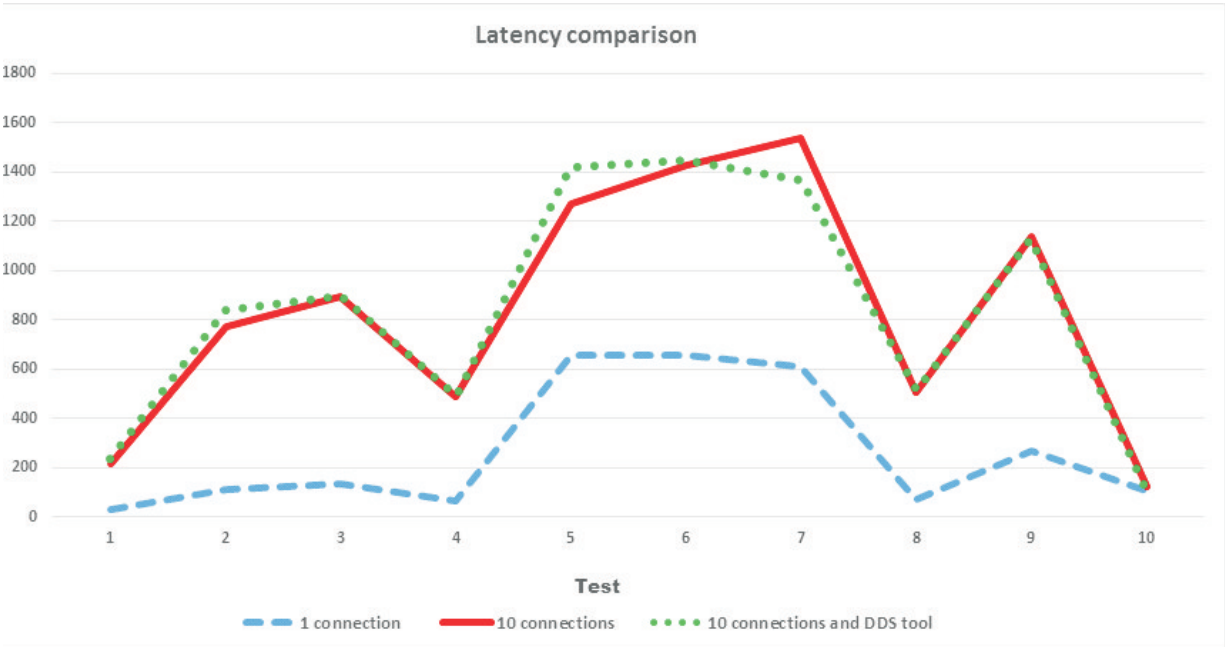

FIG. 6.1. *Latency comparison with different number of connections.*

The test suit consists of 10 HTTP requests to the following URLs:

1. /
2. /dataset
3. /dataset?page=2
4. /dataset?tags=transparency
5. /dataset/adur_district_spending
6. /dataset/us-national-foreclosure-statistics-january-2012
7. /dataset/activity/afghanistan-election-data
8. /dataset/adur_district_spending/resource/resource_id
9. /group/data-explorer
10. /package_list

Three main sections in the table are the corresponding metrics for latency, bandwidth and number of requests. For the first two sections we present statistical data in the columns for average(Avg) value, standard deviation (Dev), maximum (Max) and percentage of the requests within standard deviation. Last section is the average achieved transfer speed (TX) for every test. We increase the value of connection argument to increase load on server as more users would induce. That way we analyze how the performance scales with more stress and discover at which point the server reaches its limit. As we approach more than 20 active users at once, we cant find anything unusual and the evaluation of results from tests points the same. Non-linear dependence is observed between number of connections and performance drop across all tests. Image 6.1 shows this relation.

The blue dashed line represents average latency of all requests during 1 active connection. Continuous red solid line is for 10 simultaneous connections to the same APIs. Latency delay approaches a maximum 7.24 times higher with an average value of 4.68 times more across all tests for 10 times increase in load. Despite having 10 concurrent users acting on the system, the number of requests made by each one is only about 10 percent less. At the same time transfer speed increases up to 2.39 times during test eight. An average of 1.75 on the whole test suite indicates that the system doesnt starve for bandwidth. The dotted green line is present on the chart to visualize the impact of the used resources by data synchronization tool while the server has significant load at the same time. The previous 10 active connection tests are run together with dds tool. It is clear that

performance differs by a small margin and it does not affect user experience at all.

7. Conclusion. VI-SEEM Data Discovery Service uses metadata mapped onto standardized facets and which can be collected from various research and other repositories and provides the users with the possibility for flexible search and browsing. It is possible to search for keywords, partial phrases, creator, organization, publisher, time of publishing, versions, tags, research areas and communities etc. The results are presented in a user friendly form. Searching of a particular dataset is performed using easy to use command-line Python scripts or a simple web accessible form. The search task can be either different types of free-text search or so-called faceted search, concerning tags stored in the metadata accompanying the data. The users may refine their searches inside the received results.

By virtue of its nature, VDDS offers its services relying on the data of the original sources. For this reason, completeness and correctness of its data is vital and that is why it needs to be reconciled properly and regularly. We decided to implement our own tool to take care of this task. We called it **Data Synchronization Tool**. It is developed specifically for this purpose with possibility for future upgrades in mind. After first extensive tests had been made, the tool is deployed in production mode and now is repeatedly updating VDDS's database providing its intended services to VI-SEEM users.

Acknowledgments. This work is supported by EC programme H2020, Grant Agreement No 675121, VI-SEEM Project.

REFERENCES

[1] *VI-SEEM project: Virtual Research Environment (VRE) in Southeast Europe and the Eastern Mediterranean (SEEM)*, https://vi-seem.eu
[2] Atanassov, E., Karaivanova, A. and Gurov, T., *Services And Infrastructure For Virtual Research Environments For Use By Science And Business*, International Scientific Journal Industry 4.0, Issue 2, 2016, pp. 110-113, Published by Sci Tech Union of Mechanical Engineering, ISSN:2543-8582, (open access).
[3] *VI-SEEM Services Catalog*, https://services.vi-seem.eu
[4] Jeffery, K. G. and A. Asserson, *Data Intensive Science Shades of Grey*, Procedia Computer Science (2014), pp. 223.
[5] Amorim, R. C., J. A. Castro, J. R. da Silva, and C. Ribeiro, *A Comparative Study of Platforms for Research Data Management: Interoperability, Metadata Capabilities and Integration Potential.*, Advances in Intelligent Systems and Computing (2015). Vol. 353. Springer, Cham.
[6] Neumaier, S., J. Umbrich, and A. Polleres., *Automated Quality Assessment of Metadata Across Open Data Portals*, Journal of Data and Information Quality 8 (1). doi:10.1145/2964909.
[7] Kubler, S., J. Robert, Y. L. Traon, J. Umbrich, and S. Neumaier, *Open Data Portal Quality Comparison using AHP*, (2016), doi:10.1145/2912160.2912167.
[8] *Zenodo platform*, https://zenodo.org
[9] *Figshare platform*, https://figshare.org
[10] *DSpace software for open digital repositories*, http://www.dspace.org
[11] *CKAN platform*, https://ckan.org
[12] *CKAN Application Programming Interface*, http://docs.ckan.org/en/latest/api
[13] *Python SQL Toolkit and Object Relational Mapper*, https://www.sqlalchemy.org
[14] *SOLR open source enterprise search platform*, http://lucene.apache.org/solr
[15] *Open Archives Initiative Protocol for Metadata Harvesting*, https://www.openarchives.org/pmh/tools
[16] *CKAN demo data*, https://github.com/ckan/ckan-demo-data
[17] *wrk - HTTP benchmarking tool*, https://github.com/wg/wrk

Edited by: Anastas Mishev
Received: Dec 22, 2017
Accepted: Mar 30, 2018