

CERN SUMMER STUDENT REPORT 2014

[DropBox](#)

LJUBOV FEKLISTOVA
University of Tartu, Estonia

Supervisors: **Andreas Pfeiffer**

Co-supervisors: Giacomo Govi
Frank Glege
Salvatore Di Guida

PROBLEM STATEMENT

The main goal of the my summer student internship at CERN was to develop an application called *DropBox* which could allow end users to upload the descriptive data on the runs stored in SQLite files to the productive database as fast as possible.

EXISTING SYSTEM

The main problem of the existing system is relatively low productivity. The issue is connected to the two-step architecture (Fig. 1). At first, the data is stored into the *offline module* (as a temporary file in the intermediate database); then, the data is retrieved from the offline module by the *online module* and is sent to the productive database.

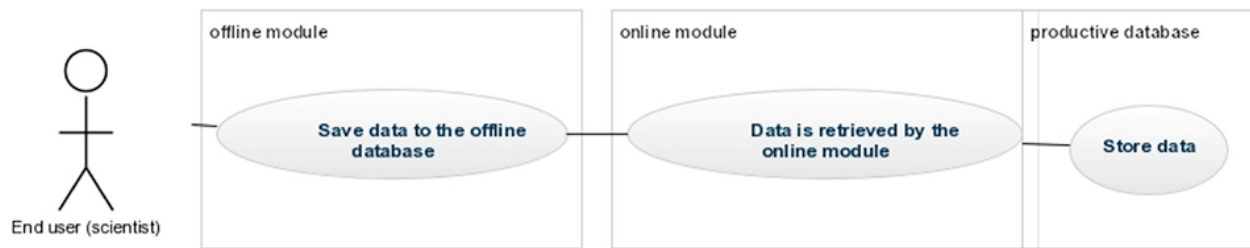


Figure 1. Existing system.

REQUIREMENTS TO NEW SYSTEM

Basic concepts

The **productivity** of the existing system could be improved by eliminating the offline module. The end user should be able send the data stored in SQLite files to the productive database via a *portlet* (Fig. 2). A **portlet** is a Java-based web component, managed by a portlet container on the portal. The advantage of the portlet technology is that portlets process requests and generate dynamic content [1].

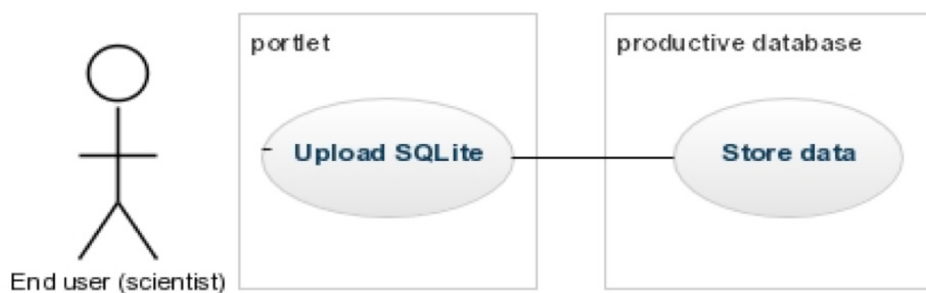


Figure 2. New DropBox.

Requirements to technologies

The project had to be realized using the following technologies:

- Jdeveloper 10.1.3.4
- Java 1.4
- Python 2.7
- Oracle 11g

IMPROVEMENT

Basic concepts

Since the end user should be given a choice to interact with the productive database via a web portal (portlet) and a traditional python script, the functionality of the data transfer between the devices was moved out from the python script and portlet to the *web service* (Fig. 3). A **web service** is a client and server application that communicates over the web and provides a standard means of interoperating between software applications running on a variety of platforms and frameworks [2].

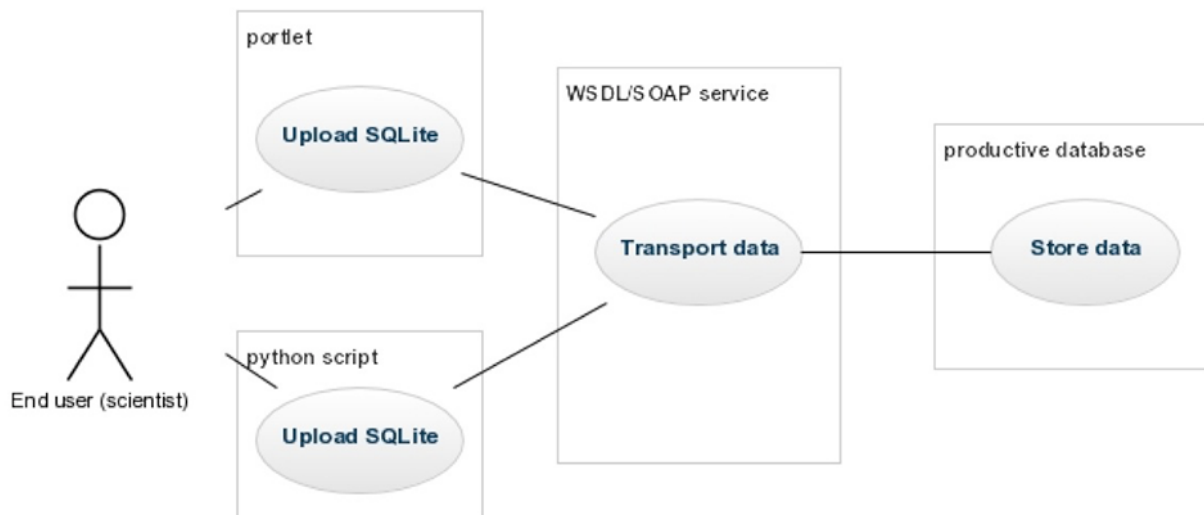


Figure 3. Improved DropBox model.

There are two types of web services [2]:

- **“big” web services**, which contain a machine-readable description of the operations written in the Web Services Description Language (WSDL) and an XML language for defining interfaces syntactically;
- **“RESTful” web services**, which use HTTP requests to post data; security features, encryption, session management, QoS guarantees, etc. must be built on the top of the HTTP manually.

Since *RESTful* web services have a list of constraints (the caching infrastructure is limited to the *HTTP GET* method, no formal way to describe the web service interface exists, the bandwidth is limited) and „big” web services are considered to be more reliable due to the existing *WS-** set of protocols (which provide standards for security and reliability) and developer/user friendly client – server approach (if the service is restarted, the client updates the signatures of the functions itself), the web service was implemented using *WSDL/SOAP* service.

A long list of *WSDL/SOAP* clients can be found in [3]; however, not all clients could be applied to the project: e.g., *pysimplesoap* does not support arrays/lists; *ZSI* requires at least python 3.0; *SOAPy* has not been maintained for a long time and does not work with python 2.5+. In the project, I used

SUDS since it meets the project requirement, lightweight and simple in creating *WSDL*-consuming *SOAP* clients.

Advantages of the approach

The proposed approach to use the *WSDL/SOAP* service has the following advantages:

- the code is not duplicated in the portlet and python script;
- the productivity of the python script is increased since it does not have to communicate with the portlet;
- *WSDL/SOAP* already contains interfaces and provides the required protocols for data of various types to be transported;
- the system is consistent since it does not have coding anomalies.

RESULT

Within eight weeks the following modules were developed:

- *uploadv4.py* python script
- *DropBox_Data_Process* *WSDL/SOAP* web service
- pseudo productive database
- *Dropbox* portlet (partly)

Python – WSDL/SOAP service – pseudo database

The following example could be considered as a documentation aimed for the end users who prefer to use the python script.

1. At first, the *SUDS* packaged should be downloaded and installed on the workstation. The instructions on the installation can be found at <http://diegosmusings.blogspot.ch/2011/03/how-to-install-python-module-in-windows.html>

2. The communication between the python script and web service depends on the web service address. If the following error is shown:

urllib2.URLError: <urlopen error [Errno 10060] A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond>

check the address in *uploadv4.py* line 33. It should be the same as when the web service started on a server.

3. Run the script from the terminal as *uploadv4.py database* or *uploadv4.py database.db* or *uploadv4.py database.txt*

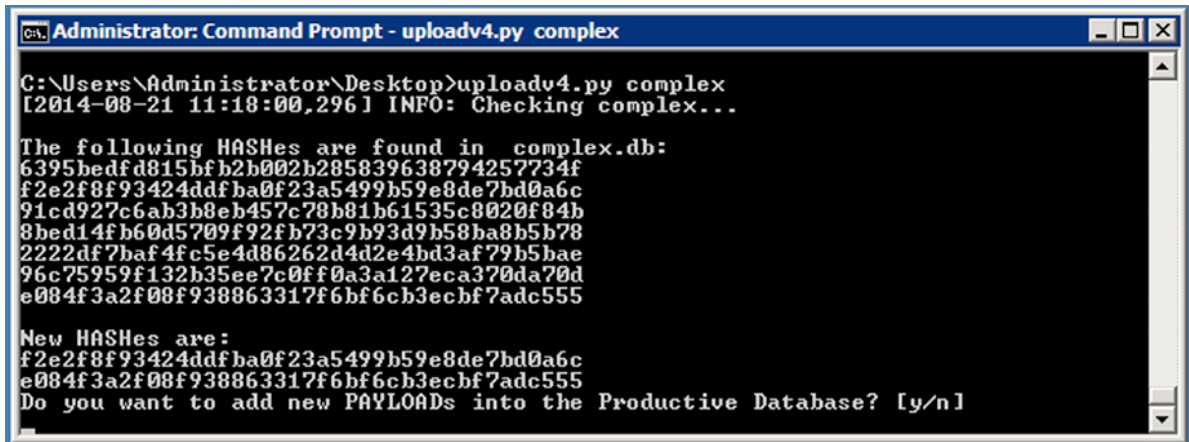
where *database* is the name of the SQLite database. Importantly, the database can be entered either without any filename extensions, or using *.db* or *.txt* filename extensions.

4. When the python script starts, firstly, it checks the entered file. If the entered file is not an SQLite database or does not exist in the same folder as the script, the following error message is shown:

ERROR: Impossible to open SQLite data file

5. If the entered file is an SQLite file, the script collects *hashes* from the SQLite *PAYLOAD* table into a list. The list is sent to the web service (the web server accepts the list and sends a request to the productive database to check if any of the hashes are already in the productive database). The python script receives from the web service a list of new hashes or an empty list.

5a. If the received list is not empty and contains new hashes, the user is asked to add new hashes into the productive database PAYLOAD table (Fig. 4).



```
Administrator: Command Prompt - uploadv4.py complex
C:\Users\Administrator\Desktop>uploadv4.py complex
[2014-08-21 11:18:00,296] INFO: Checking complex...

The following HASHes are found in complex.db:
6395bedfd815bfb2b002b285839638794257734f
f2e2f8f93424ddfba0f23a5499b59e8de7bd0a6c
91cd927c6ab3b8eb457c78b81b61535c8020f84b
8bed14fb60d5709f92fb73c9b93d9b58ba8b5b78
2222df7baf4fc5e4d86262d4d2e4bd3af79b5bae
96c75959f132b35ee7c0ff0a3a127eca370da70d
e084f3a2f08f938863317f6bf6cb3ecbf7adc555

New HASHes are:
f2e2f8f93424ddfba0f23a5499b59e8de7bd0a6c
e084f3a2f08f938863317f6bf6cb3ecbf7adc555
Do you want to add new PAYLOADs into the Productive Database? [y/n]
```

Figure 4. Insert new hashes into the pseudo productive database.

The answer can be: yes, y, no, n. Using other combinations will result in the repeating of the question.

6. On the next step, the script collects *tags* from the SQLite TAG table into a list. The list is sent to the web service (the web server accepts the list and sends a request to the productive database to check if any of the tags are already in the productive database). The script receives from the web service a list of new tags or an empty list.

6a. If the received list is not empty and contains new tags, the user is asked to add new tags into productive database. TAG table. The answer can be: yes, y, no, n. Using other combinations will result in the repeating of the question.

7. Finally, the script collects *iovs* from the SQLite IOV table into a list* and sends it to the service. The web service sends an *insert* statement to insert *iovs* into the productive database. If any *iov*'s tag was already in the TAG table before *step 6*, the service sends an *update* statement to update the tag's *modification_time* field in the productive database TAG table.

The whole scenario can be seen in Fig. 5.

* See Encountered Problems N2, 4.

```

Administrator: Command Prompt
C:\Users\Administrator\Desktop>uploadv4.py complex
[2014-08-21 11:18:00,296] INFO: Checking complex...

The following HASHes are found in complex.db:
6395bedfd815bfb2b002b285839638794257734f
f2e2f8f93424ddfba0f23a5499b59e8de7bd0a6c
91cd927c6ab3b8eb457c78b81b61535c8020f84b
8bed14fb60d5709f92fb73c9b93d9b58ba8b5b78
2222df7baf4fc5e4d86262d4d2e4bd3af79b5bae
96c75959f132b35ee7c0ff0a3a127eca370da70d
e084f3a2f08f938863317f6bf6cb3ecbf7adc555

New HASHes are:
f2e2f8f93424ddfba0f23a5499b59e8de7bd0a6c
e084f3a2f08f938863317f6bf6cb3ecbf7adc555
Do you want to add new PAYLOADs into the Productive Database? [y/n]
hmm
Please respond with 'y' or 'n'.
y

NEW PAYLOADs are ADDED into DB at 2014-08-21 11:35:08.961

The following TAGs are found in your file complex.db:
runinfo_v1
runinfo_v2
runinfo_v3

New TAGs are:
runinfo_v2
runinfo_v3
Do you want to add new TAGs into the Productive Database? [y/n]
y

NEW TAGs are ADDED into DB at 2014-08-21 11:35:11.727

Inserting IOU into the Productive database
runinfo_v1 222707 6395bedfd815bfb2b002b285839638794257734f
runinfo_v1 222708 f2e2f8f93424ddfba0f23a5499b59e8de7bd0a6c
runinfo_v1 222709 91cd927c6ab3b8eb457c78b81b61535c8020f84b
runinfo_v1 222716 8bed14fb60d5709f92fb73c9b93d9b58ba8b5b78
runinfo_v1 222717 91cd927c6ab3b8eb457c78b81b61535c8020f84b
runinfo_v1 222718 2222df7baf4fc5e4d86262d4d2e4bd3af79b5bae
runinfo_v1 222719 91cd927c6ab3b8eb457c78b81b61535c8020f84b
runinfo_v1 222785 96c75959f132b35ee7c0ff0a3a127eca370da70d
runinfo_v1 222786 91cd927c6ab3b8eb457c78b81b61535c8020f84b
runinfo_v1 222916 e084f3a2f08f938863317f6bf6cb3ecbf7adc555
runinfo_v2 222786 91cd927c6ab3b8eb457c78b81b61535c8020f84b
runinfo_v2 222916 e084f3a2f08f938863317f6bf6cb3ecbf7adc555
runinfo_v3 222916 e084f3a2f08f938863317f6bf6cb3ecbf7adc555
NEW IOUs are ADDED into DB at 2014-08-21 11:35:11.861

C:\Users\Administrator\Desktop>

```

Figure 5. Demonstration of uploadv4.py complex

The documentation of the WSDL/SOAP service aimed for developers can be found in the *javadoc* file.

To connect the pseudo productive database, the following connection properties can be used (Fig.6):

Host Name / IP Address:	int2r2.cern.ch
Port:	10121
Service Name:	int2r.cern.ch
UserName:	cms_test_dropbox_v3
Password:	can be obtained from Giacomo Govi

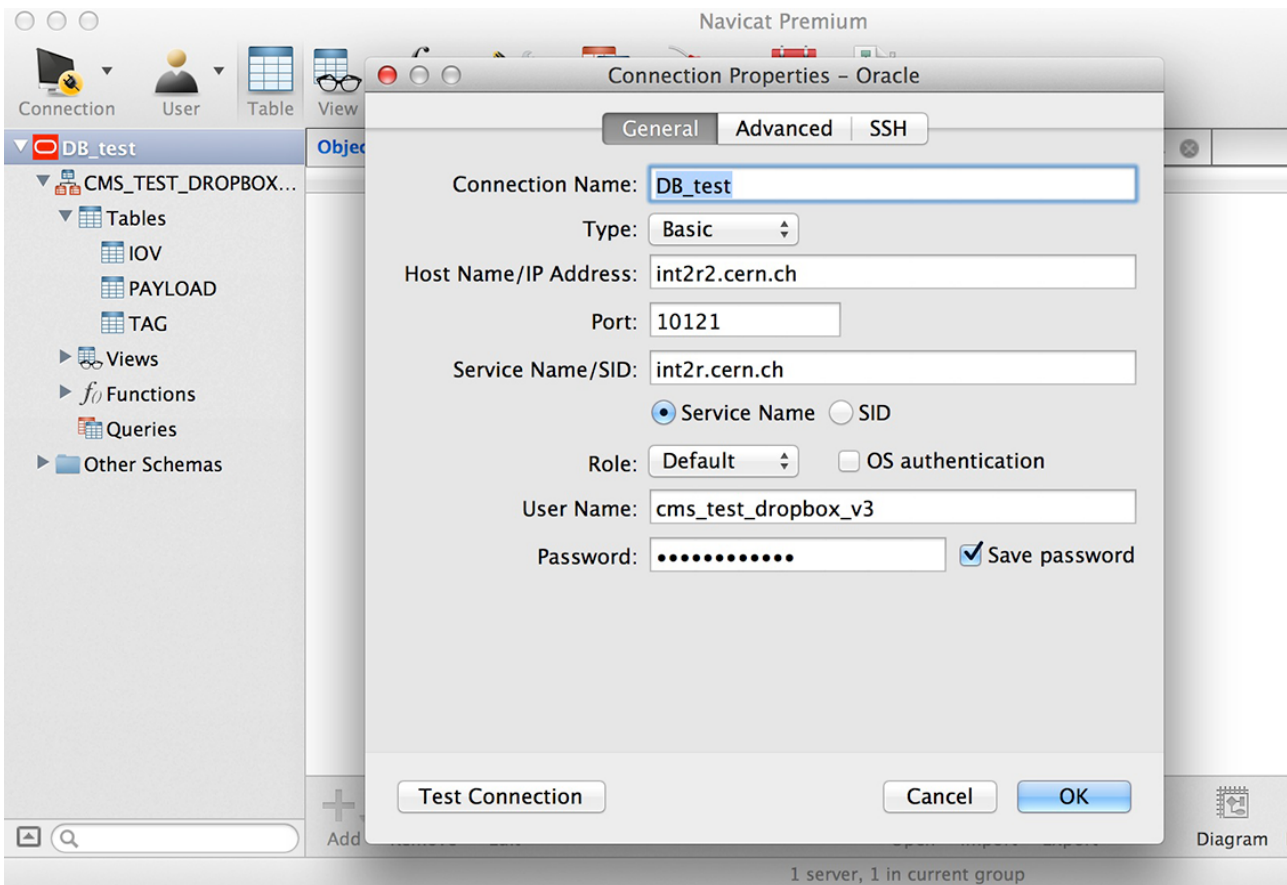


Figure 6. Connection properties of the pseudo productive database.

ENCOUNTERED PROBLEMS

During the development of the project the following problems raised.

1. Java 1.4 was released in 2002 and has very limited functionality. The documentation is not available online and can be only downloaded from <http://docs.oracle.com/javase/1.4.2/docs/api/>
2. Java 1.4 does not support *Blob* type data. In order to send blob objects from SQLite database to the productive database, the blob type object is converted into a string using *Base64* in the python script and decoded the same way on the service side before sending the data into the productive database. *Base64* guarantees that blobs are treated the way in the python script and on the service side.
3. Java 1.4 release for Windows OS does not support *Base64* (in comparison to the Java 1.4 release for OS X). Therefore, if the *WSDL/SOAP* is going to be deployed on Windows Server 2008, *Base64.java* class should be included. The source can be found from <http://www.java2s.com/Code/Java/Development-Class/Base64encoderdecoder.htm>
4. Jdeveloper 10.1.3.4 and *WSDL/SOAP* support only simple lists. Jdeveloper 10.1.3.4 does not support serialization needed for the value objects and *WSDL/SOAP* does not support complex lists. One of these approaches was needed in order to send a list of new hashes or tags (Fig. 7) from the SQLite database to the productive database.

NAME	TIME_TY	OBJECT_T	SYNCHR	END	DESCRIPTION	LAST	INSERTION_TIME	MODIFICATION_TIME
runinfo_v1	Run	RunInfo	Offline	-1	Created copying t	0	2014-07-16 09:34:31.722000	2014-08-21 11:35:11.861000
runinfo_v0	Run	RunInfo	Offline	-1	Created copying t	0	2014-08-15 12:29:40.103000	2014-08-15 12:29:40.186000
runinfo_v3	Run	RunInfo	Offline	-1	Created copying t	0	2014-08-21 11:35:11.727000	2014-08-21 11:35:11.861000
runinfo_v2	Run	RunInfo	Offline	-1	Created copying t	0	2014-08-21 11:35:11.727000	2014-08-21 11:35:11.861000

Figure 7. The data to be inserted into TAG table.

The problem is solved by a simple list of strings. Each string in the list represents data on a tag/payload. The string is created by merging the data about one payload/tag. The pattern 'k.,f_k.,jdm' is used as a separator. Indisputably, the best separator would be a combination of the ACSII characters, but the ASCII code is not supported by WSDL/SOAP. Nevertheless, 'k.,f_k.,jdm' separator can also be considered as a reliable pattern since the data is coded using Base64 (it does not contain dots and commas). On the service side the data is decoded into separate elements and inserted into the productive database.

Using the approach, the productivity of the system does not decrease significantly since:

- the data is transported from python to the service side at once in one list;
- sending complex lists or objects from the python application to the service side would also result in data compression by the python's methods.

5. The portlet module has been barely developed due to the constrictions on the permission to deploy anything to the server. Without permissions it was extremely difficult to develop functionality on the file upload and data extraction from the uploaded file (the corresponding servlet could not be deployed). At the moment, only a file upload form and proxy are realized in the portlet side.

6. Also it was found out that the current version of JDeveloper does not support a *FileItem*. The uploaded files can be treated by bytes, not as a file object, which will reduce the productivity of the portlet module.

7. In order to secure the WSDL service, LDAP was in the process of the development. Unfortunately, at the very last moment, it came out that a separately standing module did not fit the project requirements

FUTURE DEVELOPMENT

As any other application, the *DropBox* still needs further development and maintenance.

1. In order the end users could use the python script, the WSDL/SOAP service should be deployed into the CSMonline server.
2. Secondly, a reasonable approach should be found for the file upload and read at the portlet side. Once the data is fetched, it can be sent to the service side through the proxy (already implemented).

To summarize my CERN summer student internship, I would like to acknowledge my supervisors for the chance and trust to work for CMS. It was a big challenge to increase the productivity of the system and realize up-to-date ideas using the technologies of the previous decade. If I had a chance to start from the scratch, I would put more effort on the Low Level Socket Programming and Custom Protocol - this approach would meet the project requirements as well as requires more skills and experience.

REFERENCES

- [1] Oracle (2008). Java Portlet Specification V2.0. Available *online* [<https://portals.apache.org/pluto/portlet-2.0-apidocs/javax/portlet/Portlet.html>] [21.08.2014]
- [2] Oracle (2010). The Java EE6 Tutorial. Available *online* [<http://docs.oracle.com/cd/E19798-01/821-1841/gijti/index.html>] [21.08.2014]
- [3] DaleAthanasias (2014). Web Services. Available *online* [<https://wiki.python.org/moin/WebServices>] [21.08.2014]