2

DELIVERABLE REPORT D4.2

Descriptor Calculation Algorithms and Methods

GRANT AGREEMENT:	604134
ACRONYM:	eNanoMapper
NAME:	eNanoMapper - A Database and Ontology Framework for Nanomaterials Design and Safety Assessment
PROJECT COORDINATOR:	Douglas Connect GmbH
START DATE OF PROJECT; DURATION:	1 February 2014; 36 months
PARTNER(s) RESPONSIBLE FOR THIS DELIVERABLE:	IST
DATE:	29.12.201429.4.201529.4.201529.4.2015
VERSION:	[V.2.0.]

Call identifier	FP7-NMP-2013-SMALL-7
Document Type	Deliverable Report
WP/Task	WP4/ T4.2
Document ID	eNanoMapper D4.2
Status	Final

Partner Organisations	• Douglas Connect, GmbH (DC) • National Technical University of Athens (NTUA) • In Silico Toxicology (IST) • Ideaconsult (IDEA) • Karolinska Institutet (KI) • VTT Technical Research Centre of Finland (VTT) • European Bioinformatics Institute (EMBL-EBI) • Maastricht University (UM) • Misvik Biology Oy (MB)
Authors	Philip Doganis Georgia Tsiliki Haralambos Sarimveis Haralambos Chomenidis Georgios Drakakis Egon Willighagen Final review and edit by Barry Hardy
Purpose of the Document	To report on the progress on developing descriptor calculation methods, based on image analysis, crystallographic data, high-throughput and *in vitro* data, and structural information of cores and coatings.
Document History	1.Table of Contents, 18/03/2015 2.First draft, 29/04/2015 3. Second draft, 03/06/2015 4. Third draft, 16/06/2015 5. Full draft, 31/07/2015

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY 6
2. INTRODUCTION 7
3. DEVELOPMENT OF IMAGE DESCRIPTORS CALCULATION TOOLS 8
3.1 IMAGE ANALYSIS OPEN SOURCE SOFTWARE – ASSESSMENT AND SELECTION 8
3.2 FILTERS 9
3.3 IMAGE DESCRIPTORS 10
3.4 DEVELOPMENT OF THE IMAGE ANALYSIS WEB APPLICATION 11
3.5 APPLICATION WORKFLOW 12
3.5.1 Index Page 12
3.5.2 Filter Selection 13
3.5.3 Descriptor Selection 14
3.5.4 File Upload and Results 14
4. QUANTUM MECHANICAL DESCRIPTORS USING MOPAC OPENTOX SERVICE 19
5. GENE ONTOLOGY DESCRIPTORS (OR HIGH-THROUGHPUT AND IN VITRO OMICS DATA) .23
5.1 GO DESCRIPTORS 23
5.2 FORMALIZATION 24
5.3 APPLICATION TO PROTEIN CORONA DATA 25
6. CDK DESCRIPTORS 27
7. CONCLUSION 29
8. BIBLIOGRAPHY 30
9. ANNEX I: OBTAINING A TOKEN THROUGH THE SWAGGER INTERFACE 32
10. ANNEX II - TECHNICAL ARCHITECTURE OF THE IMAGE ANALYSIS WEB APPLICATION 33

TABLE OF FIGURES

Figure 1 A: Original TEM image. Images with identified numbered particles after application of filter: B: Huang, C: Intermodes, D: Isodata. Image from Taylor-Pashow et. al. 2011. 9
Figure 2 Application Index page. 13
Figure 3 Original Image (Nanoparticles, 30nm, Gold) 15
Figure 4 Image Preview with Huang filter. 15
Figure 5 Nanoparticle outlines with index numbers, after image processing with Huang filter 16
Figure 6 Results table with calculated image descriptors 17
Figure 7 Interface to upload .pdb structure files 19
Figure 8 Dataset for NiO with PNB showing NM structure diagram and InChI key. 19
Figure 9 Access policy page for https://apps.ideaconsult.net/enmtest/dataset/39 20
Figure 10 Output dataset with Quantum Mechanical calculated Descriptors 22
Figure 11 Swagger interface for tokens 32
Figure 12 Application Architecture 36
Figure 13 MVC design pattern description 37

GLOSSARY

Abbreviation / acronym	Description
BP	Biological Processes
CC	Cellular Components
CDK	Chemistry Development Kit
CV	Cross Validation
EN	Elastic Net
GLM	Generalized Linear Model
GO	Gene Ontology
GSEA	Gene Set Enrichment Analysis
HOMO	Highest Occupied Molecular Orbital
JSF	JavaServer Faces
JSON	JavaScript Object Notation
KEGG	Kyoto Encyclopedia Genes Genomes
LOO	Leave One Out
LUMO	Lowest Unoccupied Molecular Orbital
MF	Molecular Functions
MOPAC	Molecular Orbital PACkage
MS	Mass Spectrometry
nanoQSAR	Nano- Quantitative Structure-Activity Relationship
NN	Neural Networks
nQSAR	Nano- Quantitative Structure-Activity Relationship
PLS	Partial Least Squares
QSAR	Quantitative Structure-Activity Relationship
RBF-DDA	Radial Basis Dynamic Decay Adjustment
RF	Random Forest
RFE	Recursive Feature Elimination
SVM	Support Vector Machines
TEM	Transmission electron Microscopy
VIP	Variable Importance to Projection
Xvfb	X virtual framebuffer

1. EXECUTIVE SUMMARY

This deliverable focuses on extensions to the OpenTox descriptor calculation services, in order to account for the supra-molecular pattern (size, shape, porosity and irregularity of the surface area and coatings) governing the activities, effects and properties of ENMs. New descriptors have been derived by applying image analysis techniques on raw microscopy nanomaterial images. The quantum-mechanical semi-empirical calculation services using the MOPAC open-source software have been adapted and tested on the special requirements of ENM. Experimentally measured compositions of protein coronas (nano-bio interfaces) have been used as fingerprints of ENMs, addressing ENM exposure and life cycle assessment, as well as human and ecological hazard assessment. Further analysis of protein corona data, based on the integration of information form Gene Ontology (GO) resulted in the derivation of GO descriptors. Finally, the popular Java-based Chemistry Development Kit (CDK) has been extended to include nanomaterial descriptors.

2. INTRODUCTION

Current state of the art approaches in Quantitative Structure-Activity Relationship (QSAR) modelling concern small molecules and are primarily based on developing descriptors that are representative of chemical structures. Although a large number (in the order of thousands) of QSAR descriptors have been defined so far, they are often inadequate to express the supra-molecular pattern governing the unusual activity and properties of nanomaterials. Similarity of nanoparticles must accommodate many aspects other than chemical similarity, such as structural similarity including size, size distribution, shape, porosity and crystal structure (Winkler et al. (2013), Burello & Worth (2011), Puzyn (2011), Gajewicz et. al. (2012)). The likely presence of multiple coatings can further influence the material properties and should be taken into account when addressing nanoparticle similarity. The formation of lipid and protein coronas (Walkey et. al., 2014), whose composition can be considered as a biological fingerprint of nanoparticles is another factor that defines similarity or dissimilarity between two nanostructures. This deliverable describes the possibilities and approaches we have explored in the eNanoMapper project for expressing the supra-molecular pattern of engineered nanoparticles, by extending and adapting OpenTox descriptors or by deriving new sets of descriptors.

Section 3 describes the image analysis web application developed, which calculates image descriptors (such as size and shape) for nanoparticles based on microscopy images. Section 4 gives a thorough presentation of how crystallographic data for nanomaterials can be used for calculating quantum mechanical descriptors using the OpenTox services that implement the MOPAC semi-empirical methods. Section 5 introduces the so called GO descriptors that are computed by applying clustering techniques on protein corona (proteomics) data using information from the Gene Ontology (GO). Section 6 describes the extensions of the Java-based Chemistry Development Kit (CDK) to include descriptors for nanoparticles, with emphasis on metal oxides. The deliverable includes many examples with curl commands, which require a token. Appendix I shows a way for a user of the eNanoMapper computational web services to obtain a token through the Swagger interface. Appendix II presents in details the technical architecture used for developing the image web application.

3. DEVELOPMENT OF IMAGE DESCRIPTORS CALCULATION TOOLS

Images taken by Electron or Fluorescence microscopy are considered to be a valuable source of information, which describes the structure and shape of a nanomaterial. To this end, a web tool has been developed, which provides the user with a systematic framework for the automated analysis of nanomaterial microscopy images and the calculation of nanoparticle descriptors. The web application has been developed in the Java language and is based on the open source ImageJ software (Rasband, 1997-2014). Its input consists of microscopy nanomaterial images, as well as user-defined parameters. These comprise the choice of filter to be applied and the selection of descriptors to be calculated. The application outputs a table with the calculated descriptor values for each particle and the average value for all particles of the image, which can be produced in a range of transferable formats. The tool is released with a user interface and has been made available as a web service, thus allowing its integration into larger applications, such as the eNanoMapper Computational infrastructure, and facilitating the utilization of the produced descriptors for nanoQSAR modelling. The user interface is hosted in the following address: http://app.jaqpot.org:8880/imageAnalysis/. The source code can be found at: https://github.com/enanomapper/imageAnalysis.

The workflow of the web application can be summarized in four steps, which are described in detail later in this section. These consist of:

1) Uploading a digital image
2) Converting the image to greyscale
3) Application of Image analysis techniques
4) Storage and statistical processing of image descriptors.

3.1 IMAGE ANALYSIS OPEN SOURCE SOFTWARE – ASSESSMENT AND SELECTION

The image analysis library is a core module of the web tool. As mentioned in D4.1, the ImageJ open-source software was chosen after a thorough review of existing image analysis tools (for details please see Annex of D4.1).

ImageJ was chosen for the following reasons:

i. It **appeared in 1997** and has been **consistently upgraded** ever since.

ii. It has an **open API** that is straightforward to use and read (upgrade existing source code, build custom plugins and contribute to project evolution of this project).

iii. It is both a **stand-alone application and a library for development**. This means that its features can be modified, but also that users are already familiar with the terms and functionalities within the tool.

iv. It has been **tested and used by professionals**. Although the proficiency level among its users varies, ImageJ has a strong community support and the capacity to cover a wide range of image analysis tasks for professionals. This is due to the fact that available plugins and feedback come from very different fields.

In the next section, we describe the descriptors calculated, present the design and parameters selected, and outline the application workflow.

3.2 FILTERS

Five predefined filters for grey images have been included in the application: Huang, Triangle, Internodes, Isodata and IJ_IsoData (for more information, please consult ImageJ User Guide v1.46 by Ferreira&Rasband). Figure 1 shows the resulting images that occur after the application of three different filters to an original image (top left). It can be seen that these filters have clarified the elements in the image and can thus assist in the identification of distinct nanoparticles or agglomerates.

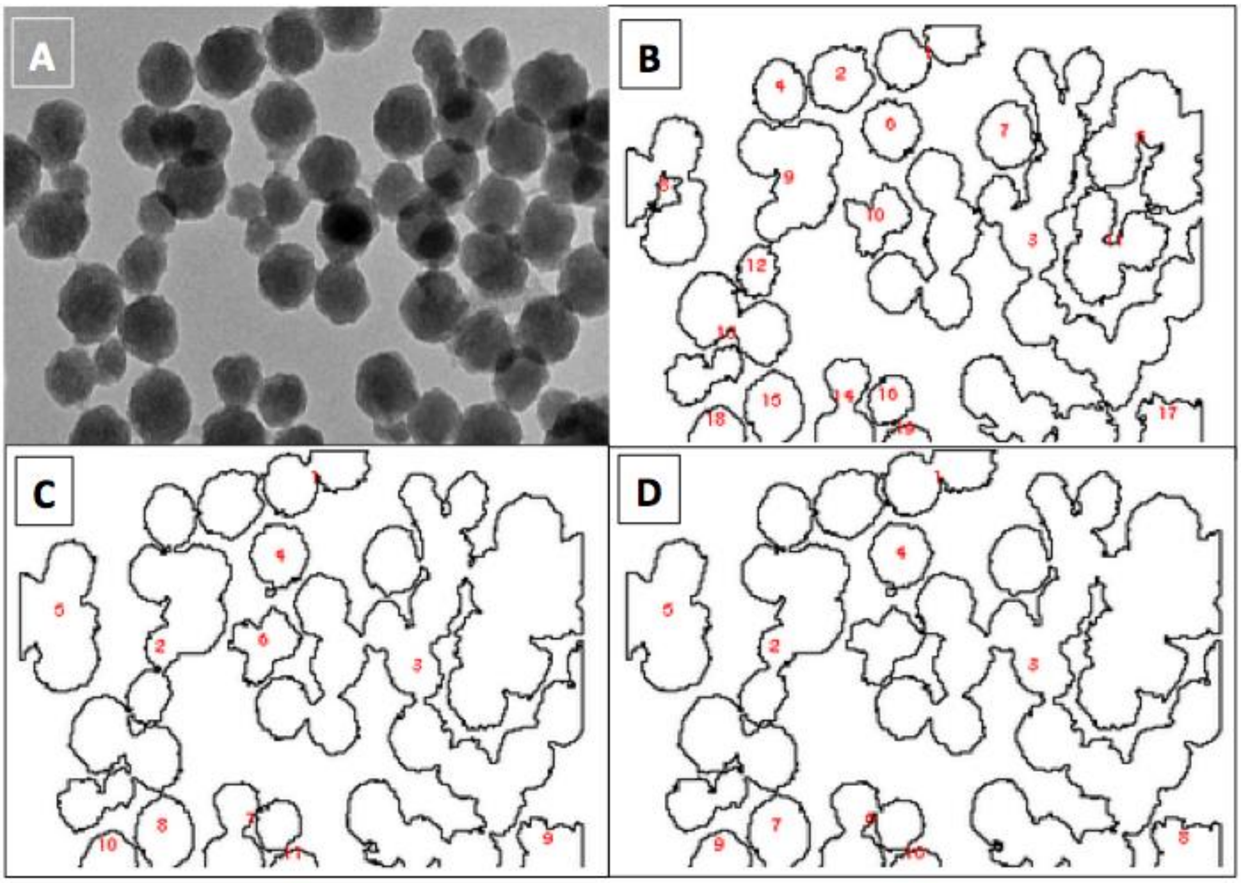


Figure 1 A: Original TEM image. Images with identified numbered particles after application of filter: B: Huang, C: Intermodes, D: Isodata. Image from Taylor-Pashow et. al. 2011.

3.3 IMAGE DESCRIPTORS

The majority of the descriptors currently implemented in this application are part of the ImageJ (http://rsb.info.nih.gov/ij/docs/menus/analyze.html) functionalities. These descriptors are presented below, along with a short description.

- **Area**: The area of selection in square pixels. This can be provided in calibrated units if a calibration of the image is provided.
- **Bounding Rectangle**: The smallest rectangle enclosing the selection. Uses the headings BX, BY, Width and Height, where BX and BY are the coordinates of the upper left corner of the rectangle.
- **Centre of Mass**: This is the brightness-weighted average of the x and y coordinates all pixels in the image or selection. Uses the XM and YM headings. These coordinates are the first order spatial moments.
- **Centroid**: The central point of the selection. This is the average of the x and y coordinates of all of the pixels in the image or selection. Uses the X and Y Results table headings.
- **Feret Diameter**: The longest distance between any two points along the selection boundary, also known as maximum caliper. Uses the Feret heading. FeretAngle (0-180 degrees) is the angle between the Feret diameter and a line parallel to the x-axis of the image. MinFeret is the minimum caliper diameter. The starting coordinates of the Feret diameter (FeretX and FeretY) are also displayed. The DrawFeretDiameter macro draws the Feret diameter of the current selection.
- **Fit Ellipse**: Fit an ellipse to the selection. Uses the headings Major, Minor and Angle. Major and Minor are the primary and secondary axes of the best fitting ellipse. Angle (0-180 degrees) is the angle between the primary axis and a line parallel to the x-axis of the image. The coordinates of the centre of the ellipse are displayed as X and Y if Centroid is checked. Note that ImageJ cannot calculate the length of either the major or the minor axis if Pixel Aspect Ratio in the Set Scale dialog is not 1.0.
- **Integrated Density**: Calculates and displays two values: "IntDen" (the product of Area and Mean Grey Value) and "RawIntDen" (the sum of the pixel values in the image or selection). "RawIntDen" is only available in ImageJ 1.44c or later. "IntDen" and "RawIntDen" values are the same for uncalibrated image. The Dot Blot Analysis example demonstrates how to use this option to analyse a dot blot assay.
- **Kurtosis**: The fourth order moment about the mean.
- **Mean Grey Value**: Average grey value within the selection. In this value the sum of all pixels within the selection is divided by the count of pixels. If the image is not in grey scale then a

transformation is made according to the following formula: *grey=0.299red+0.587green+0.114blue*.

- **Median**: The median value of the pixels in the image or selection.
- **Min and Max Grey Levels**: The highest and lowest grey scale values appearing in the selection.
- **Modal Grey Value**: Stands for the most frequently occurring grey value within the selection.
- **Perimeter**: The length of the outer boundary of the selection.
- **Shape Descriptors (previously Circularity)**: Calculate and display the following shape descriptors: Circularity, Aspect Ratio, Roundness and Solidity.
- **Skewness**: The third order moment about the mean. The documentation for the Moment Calculator plugin explains how to interpret spatial moments.
- **Standard Deviation**: The deviation of the grey values used to count the mean grey value.

A number of additional descriptors proposed in the literature (Gajewicz et. al., 2015, please also see Table 4, D4.1) have also been included in the application (porosity, sphericity, surface diameter, equivalent volume to surface, equivalent volume diameter). Some of these features (sphericity, equivalent volume to surface, equivalent volume diameter) use the **volume** of nanoparticles as a common base. It is well established that the automated calculation of volume from two dimension images not feasible in most cases and can only be performed when objects have specific angles. Therefore, the volume-related descriptors use **estimations** rather than **calculations** of volume. This estimation is based on the following assumption: *Given the circularity of an object, when it surpasses a defined threshold, it can be assumed that it tends to become a circle. In the opposite case it can be assumed that it tends to become a cylinder*. The threshold has been set empirically and is open to discussion and optimization. After characterizing a nanoparticle as circular or cylindrical, a volume calculation is made. If a nanoparticle is characterized as circular, the Feret diameter of the nanoparticle is used to calculate an equivalent sphere volume. In the opposite case, a cylinder equivalent volume is calculated in similar fashion.

3.4 DEVELOPMENT OF THE IMAGE ANALYSIS WEB APPLICATION

This section presents the basic components of the image analysis web application. Details of the technical architecture can be found in the Appendix of this deliverable.

The core class of this functionality is ***NanoparticleAnalyzer,*** which takes the Java object representation of the digital image and the selected measurements to be computed as input. Image analysis is then performed and the output is written to a table (***ResultsTable***) populated with the values measured. Each row comprises an image of identified nanoparticles along with a particle number assigned automatically by a counter, giving a complete overview of the nanoparticle's properties.

The web nature of the application dictated that the hosting of the entire program was undertaken by an http server. However http servers cannot support graphical interface and windows. As a result, the need of the application to draw an image when the ***NanoparticleAnalyzer*** was triggered led to Java headless exceptions. In order to address this issue, two solutions were examined. One solution would be to create a subclass of ***NanoparticleAnalyzer*** and override the methods responsible for image output. The second method was to insert a system between the source code and the http server, able to support window environment virtually. Although at first sight the first solution is more straightforward and did not require adding another server to the system, the methods in ImageJ are tangled in a complex way with the library source code. This complexity raised a huge risk in the case of overriding such an important method. The consequences in the entire library could not be easily foreseen. Integrating another system increased the complexity of the current application but left the source code intact. Thanks to this, a developer responsible for the maintenance of the source code and the system would not have to worry about any custom thrown exceptions from the overridden functions. The virtual window server provides more than the mentioned capabilities to the application. Activating this support in server side allowed support for a web service, such as a REST based API that could not react with any user interface at all.

This virtual window server chosen is XVFB (Wiggins, 2015). The concurrency between the three systems is vital as every single one of them depends on the other to operate correctly. Both the http server and the virtual window server start at the same time. When an http request arrives the http server triggers the ***NanoparticleAnalyzer*** and its functions. Subsequently, the ***NanoparticleAnalyzer*** applies OR performs image analysis to the digital image. When a request for window and screen is made, XVFB responds with a virtual screen. Currently the monitor returns as an environmental variable from XVFB #0. This is the default value for most programs. Java reads the value of this variable ($DISPLAY) and draws the graphic interface in this virtual window. As a result a headless exception is not triggered and the system can continue to the next step, showing the results. XVFB can only be installed in Linux and comes with Firefox-X11 plugin, which allows the windows captured by the virtual window server to be displayed inside the browser.

In order to create a web application with all the above functionalities and systems implemented and integrated respectively, a user interface able to provide input and present output from and to the user became necessary. Java Server Faces (JSF) is a Java branch that is focused on building user interfaces for web applications, that allows developers to build web applications by assembling reusable UI components in a page, connecting these components to an application data source / model, and wiring client-generated events to server-side event handlers. The user interface code runs on the server, responding to events generated on the client. The reusable user interface components that were mentioned above are built in XML programming language. Currently HTML is used for these components, which are known as Facelets. The servlet processes and loads the appropriate view template. The view template rendered is responsible and able to handle events and the response of the server to the client. Due to the fact that in the lifecycle of an application quite a few requests need to be handled asynchronously, we used the library by PrimeFaces, which managed to create an AJAX (Asynchronous JavaScript and XML) framework with JSF components.

3.5 APPLICATION WORKFLOW

In this section, we provide a detailed presentation for each structural element of the application, its components and functionality.

3.5.1 INDEX PAGE

The path of this index page is set in the web.xml, a configuration file used by JSF in order to recognize the hierarchy of the pages served. This index page is written in XHTML and includes the following user interface components:

- Threshold (Filter) Selection
- Command Buttons
- Select Many CheckBox
- Panel
- File Upload Mechanism

All of these components are available in the PrimeFaces web page along with examples. In this Index Page the user must make the following actions:

1. Select a filter
2. Select full image analysis or identification and analysis of nanoparticles
3. Select the descriptors to be calculated
4. Upload a file
5. Show preview of the processed file
6. Submit this form and receive the results

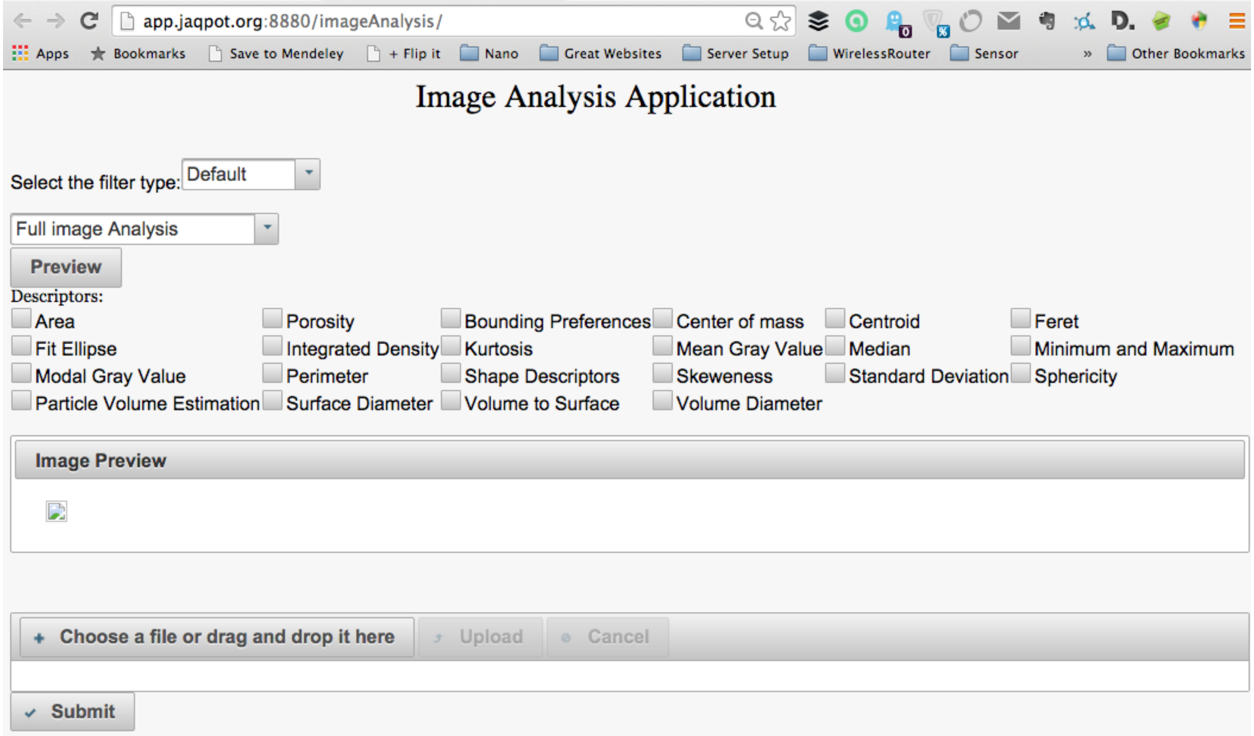


Image Analysis Application

Select the filter type: Default

Full image Analysis

Preview

Descriptors:

□ Area □ Porosity □ Bounding Preferences □ Center of mass □ Centroid □ Feret
□ Fit Ellipse □ Integrated Density □ Kurtosis □ Mean Gray Value □ Median □ Minimum and Maximum
□ Modal Gray Value □ Perimeter □ Shape Descriptors □ Skeweness □ Standard Deviation □ Sphericity
□ Particle Volume Estimation □ Surface Diameter □ Volume to Surface □ Volume Diameter

Image Preview

+ Choose a file or drag and drop it here   Upload   Cancel

✓ Submit

Figure 2 Application Index page

3.5.2 FILTER SELECTION

In this step, the user may select a filter among the five options mentioned in Section 3.2. He can also choose among performing the analysis on the entire image or let the tool identify individual nanoparticles and perform analysis on each one of them. If an image file has been uploaded, a new image preview with the applied filter is presented, in order to provide the user with a direct presentation of the differences among filter selections, without the need to upload the image again. The key mechanism in this selection is based on AJAX. When a user changes a selection menu item a JavaScript "*onChange*" event is triggered.

When this event is triggered, an asynchronous request is made to the managed bean of this view ("*Index Controller*") and a listener receives this request. Given the fact that a file is already uploaded, "*Index Controller*" combines the selected filter and the uploaded image and triggers the "*apply threshold*" method of "*Application Main Helper*". This method calls the relevant method in Process Helper and when the threshold is applied it returns a buffered image object as a result.

This buffered image object is returned as a callback in the preview property of the index page that is hosted inside a "panel" component. The final step is the update of the panel component with a PrimeFaces built-in action. According to this, the html fragment is updated and thus the preview of the image is presented. A quick survey revealed that most of the digital images captured by microscopes are high in resolution quality. However, showing a full-scale preview was far from ideal, both for the user and the optimization of the page. Therefore, a replica of the image was created and transformed into a "*DefaultStreamContent*.

3.5.3 DESCRIPTOR SELECTION

In the next step, the user selects some or all the descriptors presented in Section 4.3. In order to enable this functionality, a *Measurement Model* was created (map hosting descriptors). Each descriptor is allocated a unique key. Users may select multiple descriptors just by clicking the relevant checkboxes. During the submission of the form, these selections are saved inside an *arrayList* instance of *IndexController*. This *arrayList* is mapped with the values of the default measurements and a total measurement result is returned.

Due to the fact that more than one default descriptors are implemented and used inside the application, a descriptor mapper that matches the default and custom descriptors with user friendly names was created. This mapper is responsible for handling dependencies between separate descriptors. For example, when a user selects the volume descriptor that has a direct dependency to the Feret and Circularity descriptors, these two descriptors are automatically checked.

3.5.4 FILE UPLOAD AND RESULTS

In this step, a user is able to upload a file to the server. This component provides not only the ability to open a file browser on the clients system but also a client side validation of the files selected. Only image files (".png", ".gif", ".jpg" and ".jpeg") are accepted to be uploaded. Another main constraint is on the size limit of uploaded files, which is limited to a maximum of 10 Mb, in order to avoid uploading huge files, which may create a traffic impediment for the server. When a file is selected for upload, an input stream buffer opens and begins to convert the file into a byte stream structure. During the final stage and if no problem or exception has appeared, a message is presented to the user informing him of the state of the file.

A TEM image of 30nm gold nanoparticles shown in Figure 3 was used as an example. The image was provided by the authors of Walkey et.al. 2014.

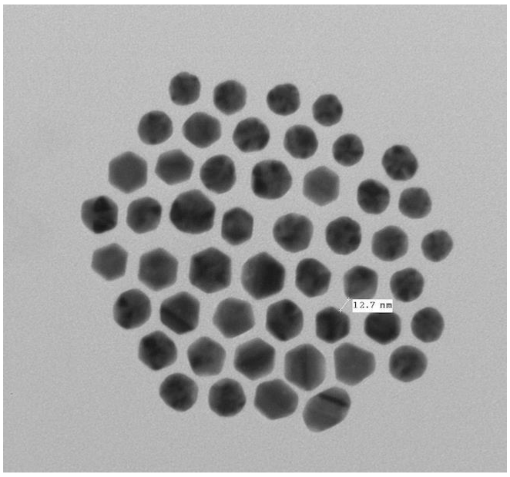


Figure 3 Original Image (Nanoparticles, 30nm, Gold)

The application of the Huang filter results in the preview shown in Figure 4**Error! Reference source not found.**:

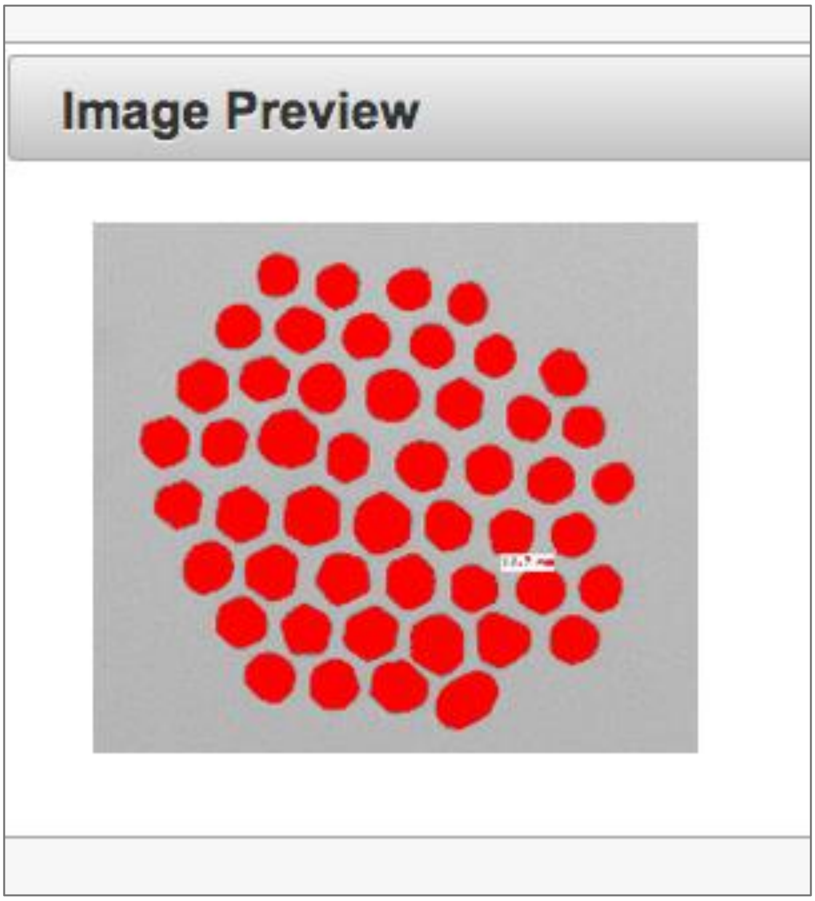


Figure 4 Image Preview with Huang filter

In this example the "Particle count and analysis" option and the full list of descriptors were chosen. After clicking on the "Submit" button, the image calculation process begins. During the process, selected measurements, filter and image files are fed into the Process helper. The results are saved to a CSV file and a parser transforms the CSV file into a list of Java models. This list is stored inside a Result model, which is passed through the session from the index page to the result page. The processed image and statistical data are also saved in the result model. The last command executed in the managed bean responsible for the index page is the redirection to the result page.

When the page has finished loading, the output of the processed image is shown at the top (Figure 5). Each nanoparticle recognised is shown with a red border and a unique counting number (index) within it. The image is shown in full scale for two main reasons. Firstly, scaling the image would make it impossible to display the nanoparticle numbers, which are important for the user. Secondly, a detailed image in the same dimensions as the original would allow the user to compare the two images.

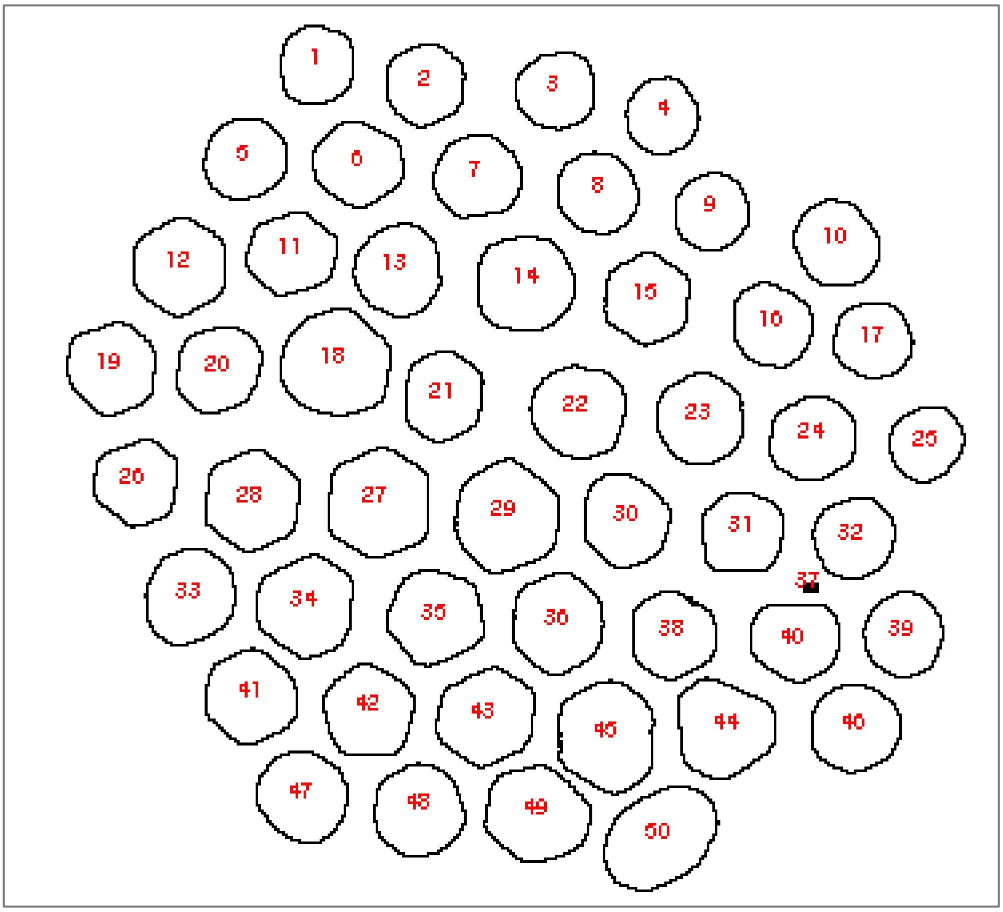
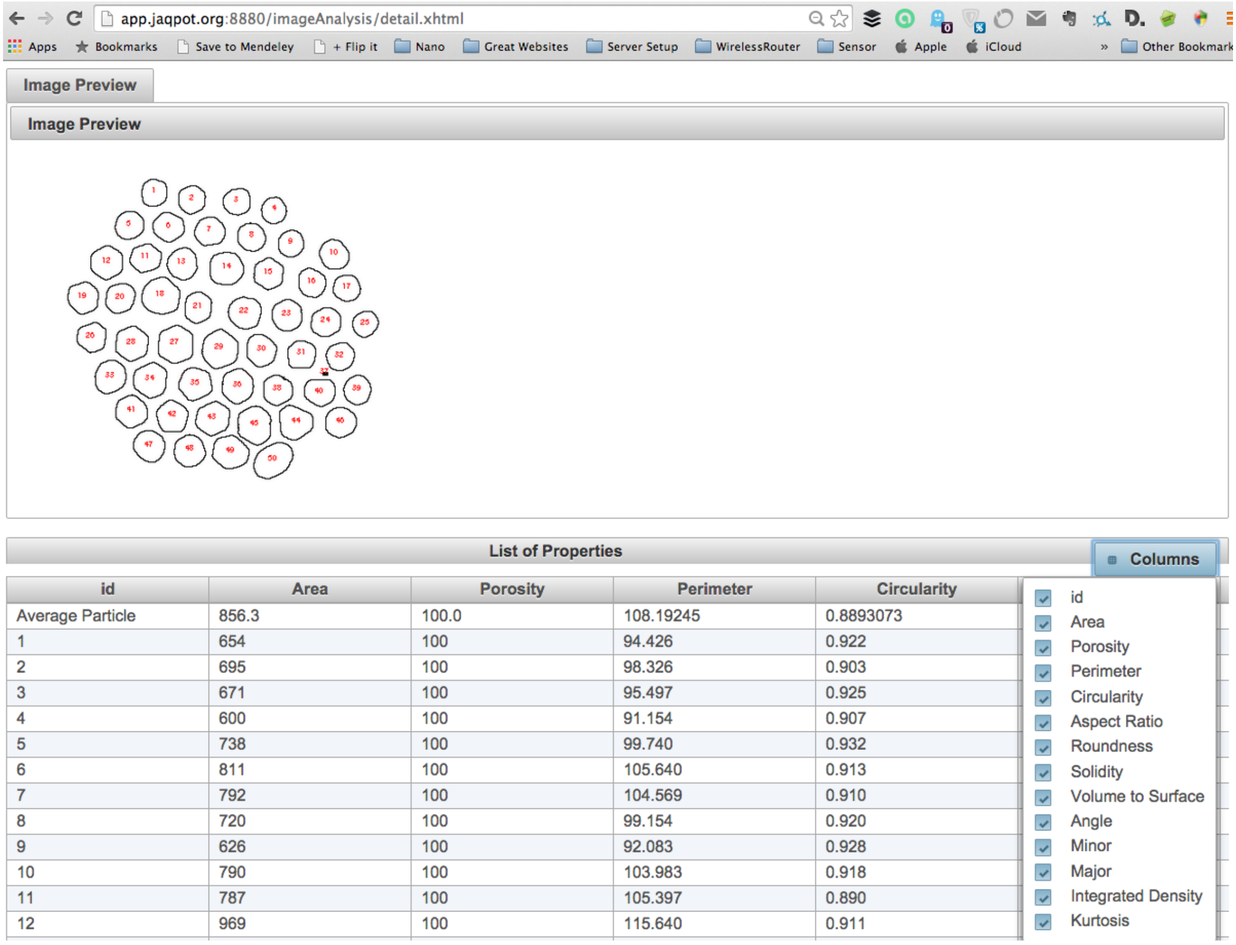


Figure 5 Nanoparticle outlines with index numbers, after image processing with Huang filter

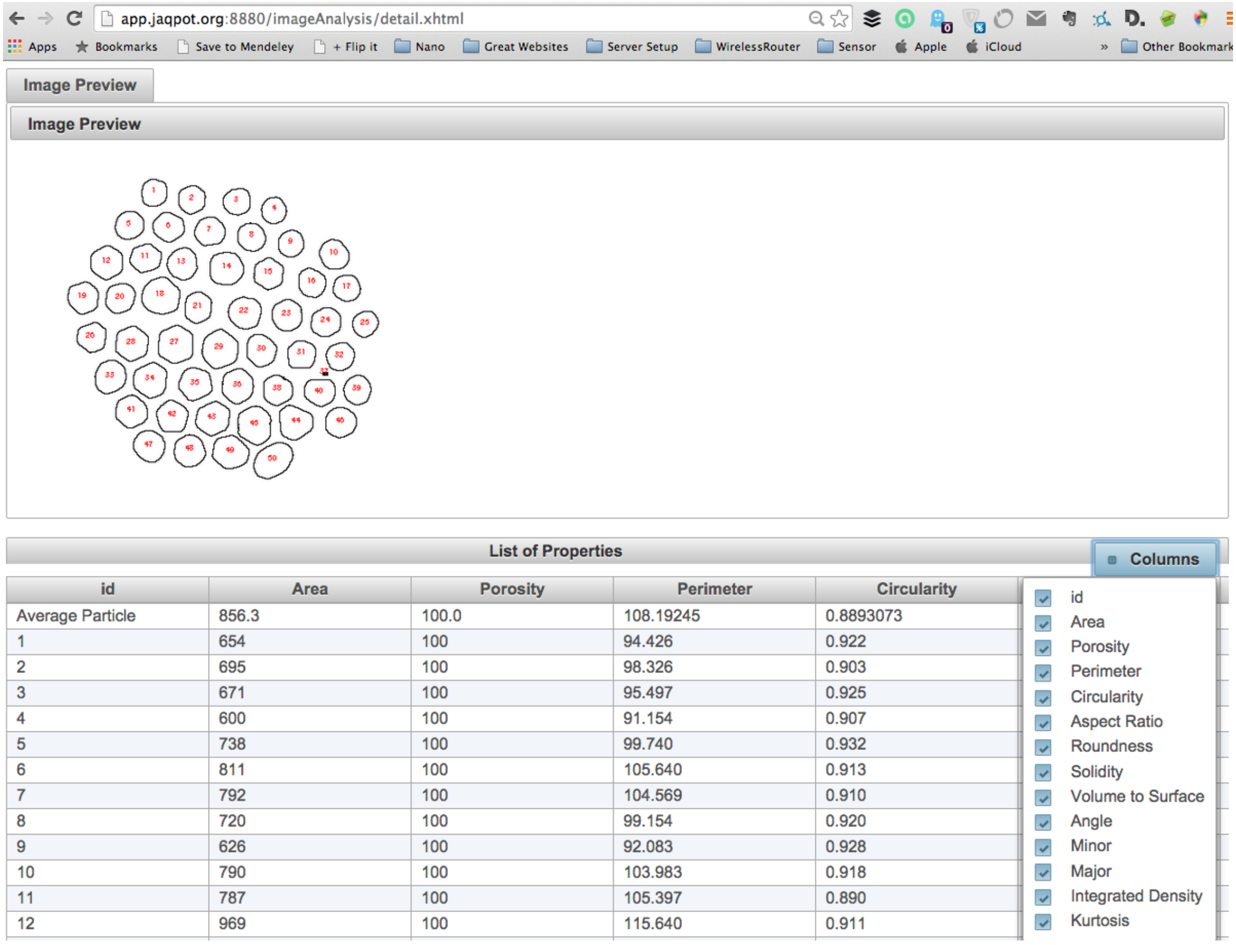
Below the image, lies the result table (



id	Area	Porosity	Perimeter	Circularity
Average Particle	856.3	100.0	108.19245	0.8893073
1	654	100	94.426	0.922
2	695	100	98.326	0.903
3	671	100	95.497	0.925
4	600	100	91.154	0.907
5	738	100	99.740	0.932
6	811	100	105.640	0.913
7	792	100	104.569	0.910
8	720	100	99.154	0.920
9	626	100	92.083	0.928
10	790	100	103.983	0.918
11	787	100	105.397	0.890
12	969	100	115.640	0.911

- ☑ id
- ☑ Area
- ☑ Porosity
- ☑ Perimeter
- ☑ Circularity
- ☑ Aspect Ratio
- ☑ Roundness
- ☑ Solidity
- ☑ Volume to Surface
- ☑ Angle
- ☑ Minor
- ☑ Major
- ☑ Integrated Density
- ☑ Kurtosis

Figure 6). Rows of the data table correspond to the distinct nanoparticles of Figure 5. In the first row, a statistical virtual nanoparticle is shown, created by dynamically calculating the average value of each column.



List of Properties

id	Area	Porosity	Perimeter	Circularity
Average Particle	856.3	100.0	108.19245	0.8893073
1	654	100	94.426	0.922
2	695	100	98.326	0.903
3	671	100	95.497	0.925
4	600	100	91.154	0.907
5	738	100	99.740	0.932
6	811	100	105.640	0.913
7	792	100	104.569	0.910
8	720	100	99.154	0.920
9	626	100	92.083	0.928
10	790	100	103.983	0.918
11	787	100	105.397	0.890
12	969	100	115.640	0.911

☑ id
☑ Area
☑ Porosity
☑ Perimeter
☑ Circularity
☑ Aspect Ratio
☑ Roundness
☑ Solidity
☑ Volume to Surface
☑ Angle
☑ Minor
☑ Major
☑ Integrated Density
☑ Kurtosis

Figure 6-Results table with calculated image descriptors

The image analysis procedure has been wrapped into an image descriptor calculation web service. An example of using the service where the analysis is performed on the entire image follows next. We call the service which is at http://app.jaqpot.org:8880/imageAnalysis/service/analyze and feed it with the example image located at http://app.jaqpot.org/imageRepo/10.1021_nn406018q/30nmWalkey.png, requesting for the Huang filter to be applied and a “Count” analysis, which provides results for individual nanoparticles and the average values of descriptors across all the nanoparticles. The value at the *subjectid:* field is a token for the user to access eNanoMapper services. A token can be obtained using the Swagger interface as shown in Appendix I.

```
curl -X POST -H
"subjectid:AQIC5wM2LY4SfczQb8RVPSGXhynrstgFwyKymCyCdeOjUk8.*AAJTSQACMDEAAlNLABMyMT
A4MDg0MTE1MDUxMDUyMTYy*" http://app.jaqpot.org:8880/imageAnalysis/service/analyze -d
"image=http://app.jaqpot.org/imageRepo/10.1021_nn406018q/30nmWalkey.png&filter=Huang&type
=Count"
```

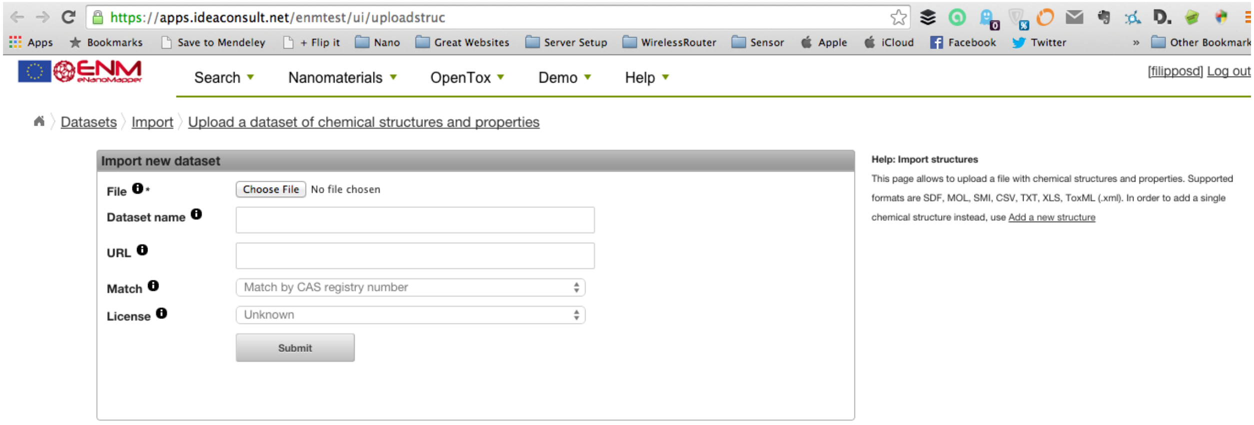
What we get in response is all the descriptors in JSON format, with the first entry being that of an “Average Particle”, where the average values of descriptors are stored, followed by entries for individual particles:

[{"id":"Average Particle",
"angle":"86.3328",

```
"area":"856.3",
"area_fraction":"100.0",
"aspect_ratio":"1.0919636",
"circularity":"0.8893073",
"feret":"35.582375",
"feret_angle":"84.634834",
"feret_x":"194.98",
"feret_y":"187.24",
"integrated_density":"62570.8",
…
{"id":"1",
"angle":"70.287",
"area":"654",
"area_fraction":"100",
"aspect_ratio":"1.076",
"circularity":"0.922",
"feret":"31.890",
"feret_angle":"41.186",
"feret_x":"115",
"feret_y":"48",
"integrated_density":"50014”,
…
```

4. QUANTUM MECHANICAL DESCRIPTORS USING MOPAC OPENTOX SERVICE

Quantum Mechanical descriptors for nanomaterials can be calculated using the desktop version of MOPAC by providing an input file containing the crystal structure of the nanomaterial and the desired MOPAC parameters. eNanoMapper web services can accept ***.pdb*** (PDB, Protein Data Bank, Berman et.al. 2000) files as input at https://apps.ideaconsult.net/enmtest/ui/uploadstruc (Figure 7). It should be noted that users need to be logged in to the system before file upload, in order to be granted access to the dataset that will be produced and subsequently be able to modify the access privileges for it.



Import new dataset

File *: Choose File No file chosen

Dataset name:

URL:

Match: Match by CAS registry number

License: Unknown

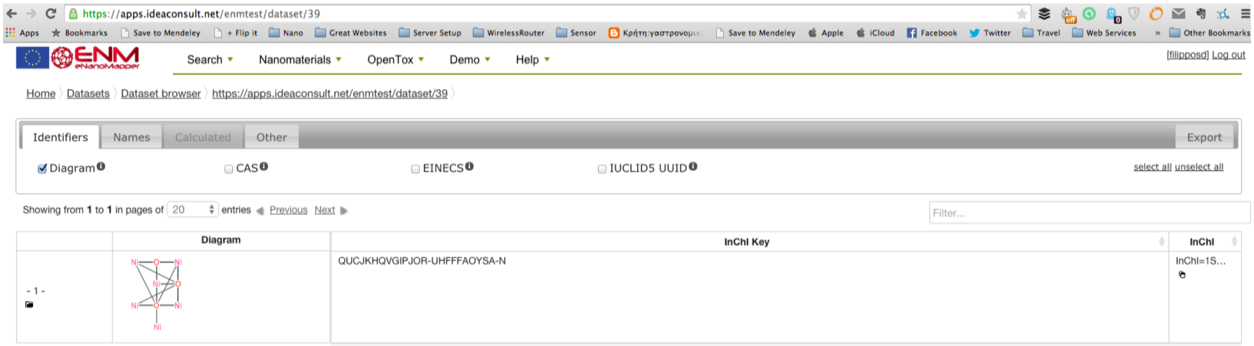
Submit

Help: Import structures

This page allows to upload a file with chemical structures and properties. Supported formats are SDF, MOL, SMI, CSV, TXT, XLS, ToxML (.xml). In order to add a single chemical structure instead, use Add a new structure

Figure 7 Interface to upload .pdb structure files

The PDB files are stored in the eNanoMapper database in the form of a dataset that contains structural information for the substance. Figure 8 shows the dataset entry for NiO, here accessible at the URI: https://apps.ideaconsult.net/enmtest/dataset/39.



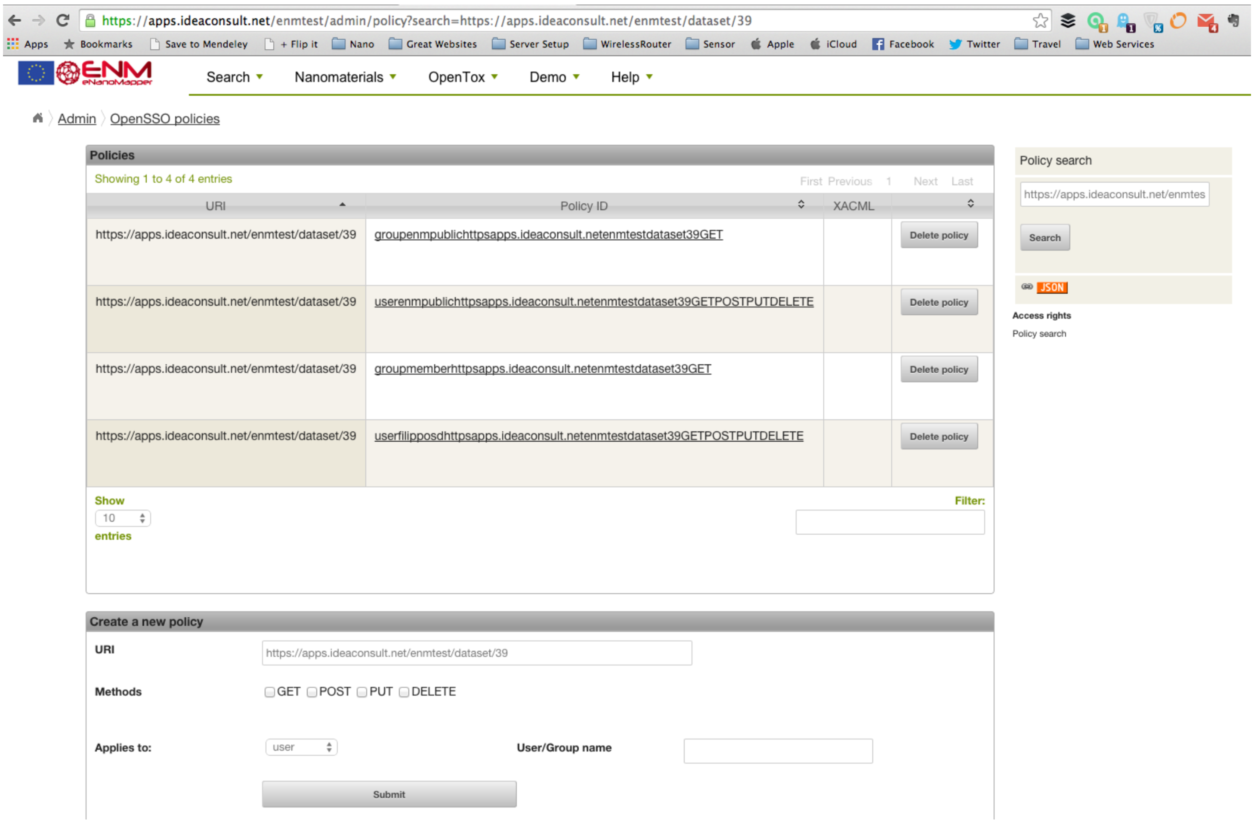
☑ Diagram ☐ CAS ☐ EINECS ☐ IUCLID5 UUID

Showing from 1 to 1 in pages of 20 entries ◀ Previous Next ▶

	Diagram	InChI Key	InChI
- 1 -		QUCJKHQVGIPJOR-UHFFFAOYSA-N	InChI=1S...

Figure 8 Dataset for NiO with PNB showing NM structure diagram and InChI key.

A specific access policy for the dataset https://apps.ideaconsult.net/enmtest/dataset/39 can be specified by accessing the service at https://apps.ideaconsult.net/enmtest/admin/policy, more specifically at: https://apps.ideaconsult.net/enmtest/admin/policy?search=https://apps.ideaconsult.net/enmtest/dataset/39. At this page, the GET-PUT-POST-DELETE privileges for all users and user groups can be specified (Figure 9).



Policies

Showing 1 to 4 of 4 entries

First Previous 1 Next Last

URI	Policy ID	XACML	
https://apps.ideaconsult.net/enmtest/dataset/39	groupenmpublichttpsapps.ideaconsult.netenmtestdataset39GET		Delete policy
https://apps.ideaconsult.net/enmtest/dataset/39	userenmpublichttpsapps.ideaconsult.netenmtestdataset39GETPOSTPUTDELETE		Delete policy
https://apps.ideaconsult.net/enmtest/dataset/39	groupmemberhttpsapps.ideaconsult.netenmtestdataset39GET		Delete policy
https://apps.ideaconsult.net/enmtest/dataset/39	userfilipposdhttpsapps.ideaconsult.netenmtestdataset39GETPOSTPUTDELETE		Delete policy

Show 10 entries

Filter:

Policy search

https://apps.ideaconsult.net/enmtes

Search

JSON

Access rights

Policy search

Create a new policy

URI: https://apps.ideaconsult.net/enmtest/dataset/39

Methods: □ GET □ POST □ PUT □ DELETE

Applies to: user

User/Group name:

Submit

Figure 9 Access policy page for https://apps.ideaconsult.net/enmtest/dataset/39

The JSON file for this dataset is as follows:

```
// 20150611164513
// https://apps.ideaconsult.net/enmtest/dataset/39?media=application%2Fjson
{
  "query": {
    "summary": "query"
  },
  "dataEntry": [
    {
      "compound": {
        "URI": "https://apps.ideaconsult.net/enmtest/compound/280/conformer/374",
        "structype": "D2noH",
        "metric": null,
        "name": "",
        "cas": "",
        "einecs": "",
        "formula": "Ni14O6"
      },
      "values": {
    },
     "facets": [
    ],
     "bundles": {
     }
   }
 ],
 "model_uri": null,
 "feature": {
 }
}
```

After the structural information of the nanomaterial has been uploaded to a dataset (for which access has been granted) the user can POST the dataset to the MOPAC algorithm using the following syntax:

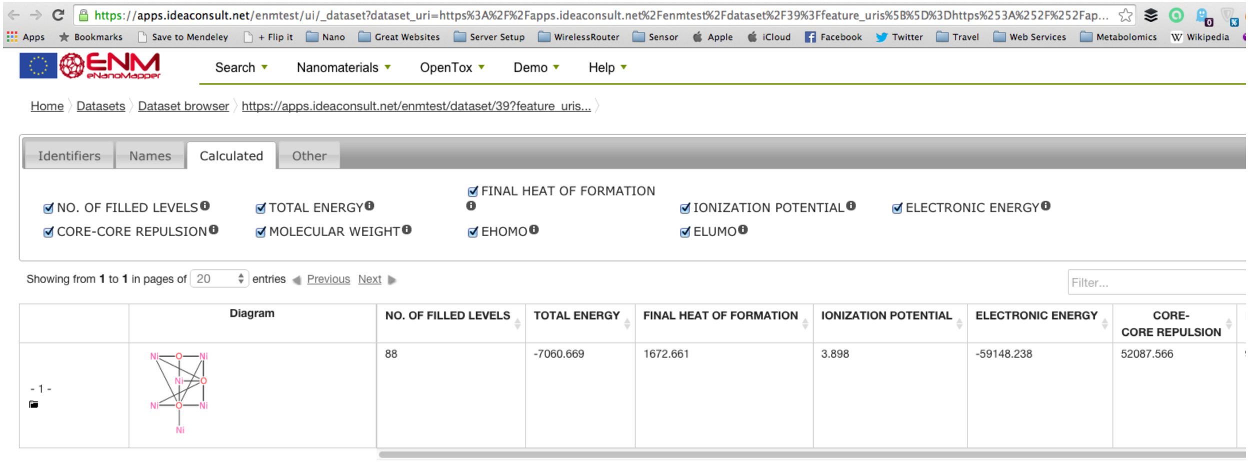
```
curl -X POST -H "subjectid:
AQIC5wM2LY4SfcyTqLk_FE87boI7KjTtgWf0ajV6MNz3Kzs.*AAJTSQACMDEAAlNLABM4NzAwNzI4NTQwO
TIxOTA5MjU1*"
-H "Accept:application/json
-d "dataset_uri=https://apps.ideaconsult.net/enmtest/dataset/39&mopac_commands=PM3 NOINTER
MMOK BONDS MULLIK GNORM=1.0 T=30.00M"
https://apps.ideaconsult.net/enmtest/algorithm/ambit2.mopac.MopacOriginalStructure  -k -i
```

The entry: *-H "Accept:application/json* requests JSON as the output format, while in the next fields: *-d "dataset_uri=https://apps.ideaconsult.net/enmtest/dataset/39&mopac_commands=PM3 NOINTER MMOK BONDS MULLIK GNORM=1.0 T=30.00M"* the user specifies the input dataset with the nanomaterial structure (*dataset_uri*) and the parameters for the MOPAC algorithm (*mopac_commands*). MOPAC methods up to PM7 are available. In the desktop version of MOPAC, these parameters would be specified in the .mop file that serves as input to the program.
POSTing the command above, a call is made to the *https://apps.ideaconsult.net/enmtest/algorithm/ambit2.mopac.MopacOriginalStructure* algorithm, which returns the following output:

```
HTTP/1.1 202 Accepted
Date: Thu, 11 Jun 2015 13:05:08 GMT
Server: Restlet-Framework/2.0m6
Content-Language: en
Vary: Accept-Charset,Accept-Encoding,Accept-Language,Accept
Accept-Ranges: bytes
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET,POST,PUT,DELETE,OPTIONS
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
Access-Control-Allow-Credentials: true
Access-Control-Max-Age: 60
Content-Type: application/json;charset=UTF-8
Content-Length: 483
Strict-Transport-Security: max-age=15552000

{"task": [
{
        "uri":"https://apps.ideaconsult.net/enmtest/task/930bf206-3d53-4c79-84dd-9c56a23a95a1",
        "id": "930bf206-3d53-4c79-84dd-9c56a23a95a1",
        "name": "Apply MOPAC descriptors (Energy, EHOMO, ELUMO,etc.) to
https://apps.ideaconsult.net/enmtest/dataset/39",
        "error": "",
        "policyError": "",
        "status": "Running",
        "started": 1434027908970,
        "completed": -1,
        "result": "https://apps.ideaconsult.net/enmtest/task/930bf206-3d53-4c79-84dd-
9c56a23a95a1",
        "errorCause": null
}
]
```

Visiting the task URI highlighted above returns information on its status. Upon completion, by following the link to the output dataset shown below, the user has access to the structure (as in the input dataset and a number of Quantum Mechanical calculated Descriptors (Figure 10).



☑ NO. OF FILLED LEVELS ☑ TOTAL ENERGY ☑ FINAL HEAT OF FORMATION ☑ IONIZATION POTENTIAL ☑ ELECTRONIC ENERGY
☑ CORE-CORE REPULSION ☑ MOLECULAR WEIGHT ☑ EHOMO ☑ ELUMO

Showing from **1** to **1** in pages of 20 entries

	Diagram	NO. OF FILLED LEVELS	TOTAL ENERGY	FINAL HEAT OF FORMATION	IONIZATION POTENTIAL	ELECTRONIC ENERGY	CORE-CORE REPULSION
- 1 -		88	-7060.669	1672.661	3.898	-59148.238	52087.566

Figure 10 Output dataset with Quantum Mechanical calculated Descriptors

The entire procedure has been wrapped into a quantum mechanical descriptor calculation web service. An example of using the service for calculating MOPAC descriptors for NiO, follows next:

```
curl -X POST –H "subjectid:AQIC5wM2LY4Sfcz502vOCINRXZQdKJvajfOzG-Xyw0Wpw-
0.*AAJTSQACMDEAAlNLABQtNDA0Nzk3MDY1OTExNDI5ODk5Ng..*"
http://app.jaqpot.org:8080/algorithms/service/mopac/calculate -d
"pdbfile=http://enanomapper.ntua.gr/pdbRepo/17002-ICSDNiO.pdb"
```

5. GENE ONTOLOGY DESCRIPTORS (OR HIGH-THROUGHPUT AND IN VITRO OMICS DATA)

High-throughput experimental methods such as RNA deep sequencing transcriptomic approaches, oligonucleotide microarrays and mass spectrometry (MS) experiments, are becoming increasingly popular and their usefulness has been demonstrated on nanoparticle data (McDermott et al., 2013; Walkey et al., 2014). It has also been shown that a single type of data is insufficient for understanding complex machinery, such as tumour behaviour (Chin and Gary, 2008). However, the vast amount of omics data has simultaneously complicated the problem of extracting meaningful molecular signatures of biological processes from these complex datasets. For that reason, more sophisticated approaches to integrating purely statistical and expert knowledge-based approaches are needed (McDermott et al., 2013). For example, studies incorporating genomic knowledge such as pathways or protein–protein interaction networks based on transcriptomics and proteomics data, have been developed to increase their power in predicting biologically relevant information (Yang et al., 2012; Balbin et al., 2013). The findings of those studies suggest that integrating omics data with genomic knowledge to construct pre-defined features, results in higher performance in predicting clinical outcomes and higher consistency between the results of different studies.

5.1 GO DESCRIPTORS

Bioinformatics workflows often determine lists of genes involved in a given biological process. A common practice is to map these genes to known pathways or phenotypes and determine links between the genes and the equivalent annotation. An example would be to pinpoint which pathways are overrepresented in a given set of genes. The scope is to provide insight into the molecular and chemical interactions, cellular phenotypes, and disease processes of the system examined. Researchers have now access to more than 300 web resources (see http://pathguide.org/) providing information for pathways and networks that document millions of interactions between genes, proteins, and small molecules. Gene Ontology (GO) and Kyoto Encyclopedia of Genes and Genomes (KEGG) are two of the most popular databases. GO (http://geneontology.org/) is a major bioinformatics initiative to annotate gene and gene products in order to maintain their controlled vocabulary. Each GO term (GO id) corresponds to a particular molecular function, biological process or cellular component (sub-domain areas); those sets of actions are performed by a set of genes in the cell. KEGG (http://www.genome.jp/kegg/) is a database resource for understanding high-level functions and utilities of the biological system, such as the cell, the organism and the ecosystem, from molecular-level information (BioPortal http://bioportal.bioontology.org/, BioPax http://www.biopax.org/).

Using GO information, we have developed a novel methodology to generate a new set of descriptors, referred to as GO descriptors, which would efficiently summarize omics data and also integrate Gene Ontology data (or any other grouping of protein ids). Our goal is to enrich the data using gene set information whilst emphasizing the importance of -omics data in modelling ENM toxicity. GO was selected as the golden standard for annotation in three ontology branches, namely Cellular Components (CC), Molecular Functions (MF), and Biological Processes (BP) containing 41,694 classes.

The steps followed to determine GO descriptors are the following:

1. From a data set of interest, e.g. transcriptomics data, we extract the gene names or gene identifiers, using for example NCBI/Entrez Gene (www.ncbi.nlm.nih.gov/gene) or Ensembl (http://www.ensembl.org/) databases.
2. Perform Gene Set Enrichment Analysis (GSEA) (Subramanian et al., 2005) for GO ontologies and filter pathway terms given the hypergeometric p-value (p-value<=0.05).
3. Construct a binary membership matrix to establish the membership of each gene to each of the significant pathways found by GSEA, i.e. 1 indicates that gene *i* is part of GO term *j*, and 0 otherwise.
4. Cluster the membership matrix to estimate groups of genes, using
   a. hierarchical clustering (Euclidean, ward)- results shown here use these specifications
   b. bi-clustering (blockcluster package from cran: https://cran.r-project.org/web/packages/blockcluster/index.html)
5. Summarize the original data set (transcriptomics data) based on the estimated groups of genes. GO descriptors correspond to the summarized data based on estimated gene groups. The summary statistics used are: mean, median, and quantile statistics. As an example, for descriptor *i*, the descriptor value that corresponds to each NP equals the mean of relative abundances for all proteins belonging to cluster *i*. Other alternatives can be used.

It should be noted that the number of new descriptors equals the number of clusters estimated by one of the algorithms above. Although different cuts of the produced dendrogram have been examined in order to estimate groups of proteins, it was decided to provide this as a user-specified parameter for both clustering algorithms. The reason behind this was to allow the user to decide to what extent the data should be summarized, but also to what depth should the interaction relationships be detailed between the significant GO ids. Both hierarchical clustering and bi-clustering algorithms were introduced in D4.1 and have been implemented as API compatible web services (please see D4.3).

GO descriptors can then be used in nanoQSAR models and be further exploited for their biological relevance and functional similarity.

5.2 FORMALIZATION

The above framework has been written as two separate R packages called GOdescrPred and GOdescrCalculus. Although they both follow the above procedure, we have made some structural changes between the two, so that the user can either select to save the suggested clustering model or to just generate the new data set of GO descriptors. In particular, GOdescrPred aims to save the cluster memberships of the proteins for a particular data set, and based on them to produce GO descriptors for a new set of data. In this case, it is a prerequisite that the new data given by the user should include the same protein/gene/ etc. ids as the original. Alternatively, GOdescrCalculus can be employed when the user wants to calculate GO descriptors for a particular set of omics data and does not have the 'clustering model', i.e. those set of proteins have not been previously used to create a set of GO descriptors. If the objective is to just create a set of new descriptors, then GOdescrCalculus should be used, where its output will be a data matrix (or array) with different columns corresponding to different GO descriptors.

GOdescrPred includes the following functions:

- generate.biclust.model to generate a model using biclustering algorithm, given the supplied data set
- generate.hierar.model to generate a model using hierarchical clustering, given the supplied data set
- pred.descr to produce a new set of descriptors based on the 'model', in this case the cluster memberships supplied by either generate.biclust.model or generate.hierar.model

Further details can be seen in http://147.102.82.122/ocpu/library/GOdescrPred/info.

GOdescrCalculus includes the following functions:

- generate.param.model to generate a serialized raw R model which in this case only includes a list with the parameters needed for the clustering of the omics data
- generate.descr.biclust to generate the new set of GO descriptors using bi-clustering
- generate.descr.hierar to generate the new set of descriptors using hierarchical clustering.

Further details can be seen in http://147.102.82.122/ocpu/library/GOdescrCalculus/info.

The GO descriptor calculation R packages have been integrated in the eNanoMapper computational infrastructure as an API compatible algorithm web service, named ocpu-go:
http://app.jaqpot.org:8080/jaqpot/services/algorithm/ocpu-go

5.3 APPLICATION TO PROTEIN CORONA DATA

As nanoparticles enter physiological fluids, proteins and other biomolecules such as lipids, adsorb to their surfaces with various exchange rates leading to the formation of the biomolecular corona (Nel et al.,2009; Pearson et al. 2015). As a consequence, the "synthetic identity" of the nanoparticle is lost and a distinct "biological identity" is acquired. This new identity alters the way in which nanoparticles interact with cells and affects cell response (Ge et al., 2011; Lesniak et al., 2012). The composition of the biomolecular corona is dynamic and is highly dependent on the initial biological environment, indicating the possibility of exposure memory (Pearson et al. 2015).

Using a bioinformatics-inspired approach, Walkey et al., developed a protein corona fingerprint model that accounts for 64 different parameters to predict the cellular interactions of nanoparticles (Walkey et al., 2014). Authors identified 129 serum proteins by LC-MS/MS which were considered suitable for relative quantification. The relative abundance of each of these proteins on a nanoparticle formulation defines the serum protein `fingerprint' for that formulation.

We used the protein corona LC-MS/MS data set (http://app.jaqpot.org:8080/jaqpot/services/dataset/corona-proteomics), along with GO information specific to each protein, to calculate GO descriptors, as defined in the previous section. This dataset includes relative abundances of all 129 proteins for 84 nanoparticles of multiple sizes having gold as the core element and many different coatings. According to the procedure above, the steps followed are:

1. UNIPROT protein ids are translated to Entrez gene ids
2. All GO ids that involve the gene ids specified in step 1 are selected.
3. GO ids are filtered using GSEA (hypergeometric test p-value<0.05)
4. A binary membership matrix is constructed using only the significant GO ids.
5. Cluster membership matrix is calculated using hierarchical clustering or bi-clustering algorithms.
6. Protein corona data is averaged given the estimated cluster memberships, to generate GO descriptors.

The application of the ocpu-go algorithm on the protein corona dataset with the following parameters: {"key":["UNIPROT"],"onto":["GO","MF"],"pvalCutoff":[0.05],"nclust":[4,2],"FUN":["mean"]} resulted in the creation of a GO model that contains two GO descriptors: http://app.jaqpot.org:8080/jaqpot/services/model/YLQVFBe95xlP (the information for this model can be viewed by entering its id (YLQVFBe95xlP) at http://app.jaqpot.org:8080/jaqpot/swagger/#!/model/getModel).
Five or seven parameters should be included depending on whether we use generate.descr.biclust or generate.descr.hierar functions. In the above example we use generate.descr.biclust() function with five parameters: the 'key' parameter which is a character string for gene/protein/etc names or id (for the corona dataset 'UNIPROT' indicates that data are named with UNIPROT ids), 'onto' that is a character vector indicating the ontology used and any possible subdirectories (for the protein corona data we use the GO ontology and particularly its MF ontology, c('GO','MF')), 'pvalCutoff' which is a numeric value for hypergeometric p-values cutoff necessary for the GSEA filtering (in the example above we are using 0.05), 'nclust' that is a numeric vector indicating the number of clusters for genes/proteins (y axis) and GOs (x axis) (e.g. c(4,2)), and 'FUN' which is a string indicating an R function to summarize the clustered data into descriptors (e.g. 'mean'). If GO descriptors are produced using the generate.descr.hierar() function the parameters expected are 'key', 'onto', 'pvalCutoff', 'distMethod', 'hclustMethod', 'nORh', 'FUN', where in 'distMethod' the user should define the distance method, in 'hclustMethod' the hierarchical clustering method, and in 'nORh' the number of clusters or height of the dendrogram.

Once the model is applied to the protein corona data, the following dataset is produced, containing the values of the two GO descriptors for all 84 gold nanoparticles:
http://app.jaqpot.org:8080/jaqpot/services/dataset/cly14DsNQKssS0

Here is the initial fragment of this dataset in JSON format:

```
{
  "meta": {
    "comments": [
      "Created by task TSKyG0Mz9XeCEuR"
    ],
    "hasSources": [
      "https://apps.ideaconsult.net/enmtest/bundle/15"
    ]
  },
  "datasetURI": "http://app.jaqpot.org:8080/jaqpot/services/dataset/corona-proteomics",
  "dataEntry": [
    {
      "compound": {
        "URI": "FCSV-8b479138-4775-3aba-b9cc-f01cc967d42b"
      },
      "values": {
        "http://app.jaqpot.org:8080/jaqpot/services/feature/1lE0XkuNuxBS": 4.189,
        "http://app.jaqpot.org:8080/jaqpot/services/feature/JnBtmOKyW40F": 2.2736
      }
    },
…
```

6. CDK DESCRIPTORS

NanoQSAR makes the general assumption that nanoparticles with the same chemical or biological identity have the same effects on endpoints. This means that two nanoparticles with almost the same structure, elicit a very similar biological effect, whereas two nanoparticles with totally different structure, may not. NanoQSAR is the process of finding the aspects of the nanoparticle structure that correlates with the biological endpoint value. This aspect may be the chemical composition of the core, but may also be a property of just the coating, or just be a property of the full particle, such as the size.

Structural descriptors for nanomaterials are, however, different from the equivalents for chemical structures in the more classical QSAR field. Firstly, nanomaterials are substances rather than compounds, and descriptors are best described as distributions of values. That is, a size descriptor can describe the nanoparticle size or it can describe the nanomaterial size distribution. However, our current approaches are aimed describing nanomaterial properties (even size) as single values. This is because all available statistical modeling approaches expect that all materials are described in the same way. Specifically, in the vector of numerical descriptor values, each position must have the exact same value.

A second thing to consider is that structural descriptors may be either theoretical or based on physicochemical or biological characterization, size descriptor being an example of the latter. Examples of the former include descriptors based on the chemical formula, such as weight or the number of oxygen atoms in the formula (Liu et.al. 2011). These theoretical descriptors can be based on quantum chemical calculations (see Chapter 4). A further note is that some descriptors were proposed or developed specifically for a class of nanomaterial or even a subgroup of that class. For example, the conductivity and valence band energy descriptors only apply for metal oxide nanoparticles, but only if they are larger than some 20-30 nanometers (Burello & Worth, 2011).

To implement structural descriptors for NP, we are currently extending the Java-based Chemistry Development Kit (CDK) with nanomaterial descriptors, which previously had atom, atom pair, bond, protein, and molecular descriptors (version 1.5.10) (Steinbeck et.al. 2003, Steinbeck et.al. 2006). In support of this development, we have proposed a new interface for substances, ISubstance, based on a list of IAtomContainers. Each container represents a component of a nanomaterial, following the design of the AMBIT software (see D3.1. One such component may be an impurity. Following the IMolecularDescriptor design of the CDK, we designed an ISubstanceDescriptor interface. Each descriptor implements this interface. The first patches have been peer-reviewed by the CDK community, and has recently been applied (https://github.com/cdk/cdk/pull/135).

A second patch for the CDK is being developed and discussed that introduces a data model to capture measured data (https://github.com/egonw/cdk-old/commits/feature/measure). This new API will introduce a common mechanism for storing measured data, ensuring we can capture size information for nanomaterials, which is used for physico-chemical descriptors, as available in the NanoJava project.

Currently, in this project we have developed a number of descriptors developed specifically for metal oxides (http://github.com/enanomapper/nanojava). Each descriptor is annotated with a term from the eNanoMapper ontology using the CHEMINF ontology (Hastings et.al. 2011) and following practices previously adopted by the CDK (Steinbeck et.al. 2006), such as:

EnergyBandDescriptor	Conductivity and valence band energies	metal oxide
MetalAtomCountDescriptor	The number of metal atoms in the molecular formula of metal oxides (Liu et.al. 2011)	metal oxide
ParticleSizeDescriptor	Collapses the size information into a single value.	any nanomaterial

These examples from the set of descriptors currently implemented show the various natures of these descriptors. The EnergyBandDescriptor uses data from quantum chemical calculations, while the MetalAtomCountDescriptor is based purely on the chemical formula associated with the nanomaterial (which commonly does not reflect the actual composition of the nanomaterials, merely the element ratios), and the ParticleSizeDescriptor is based on experimentally determined physicochemical properties. The latter constitutes a new kind of descriptor for the CDK, as it primarily involves normalizing that experimental data, into a uniform numerical descriptor. Specifically, the input data may be a range, a mean, etc, while the output must have the same meaning for all materials.

We are actively studying how we can use the molecular descriptors for organic coatings, and determine if combining these in NanoQSAR studies make models more predictive. Results of these studies are expected later this year.

7. CONCLUSION

A number of algorithms and methods have been developed in Task 4.2 of the eNanoMapper project to calculate descriptors, which together with experimental data are adequate to describe the supra-molecular pattern of nanoparticles. This information is used by the tools described in D4.3 to generate predictive nanoQSAR models. Therefore, special effort was put into the development of web service implementations in order to ensure that they are compatible with the eNanoMapper APIs and can be integrated in the overall eNanoMapper computational infrastructure. Some of the descriptors can also be used for further analysis. For example, the GO descriptors can be used for mechanistic modelling and pathway analysis. The derivation of nanomaterial-specific descriptors is an on-going process in the research community, and we plan to consistently update our descriptor calculation tools with major new advancements.

8. BIBLIOGRAPHY

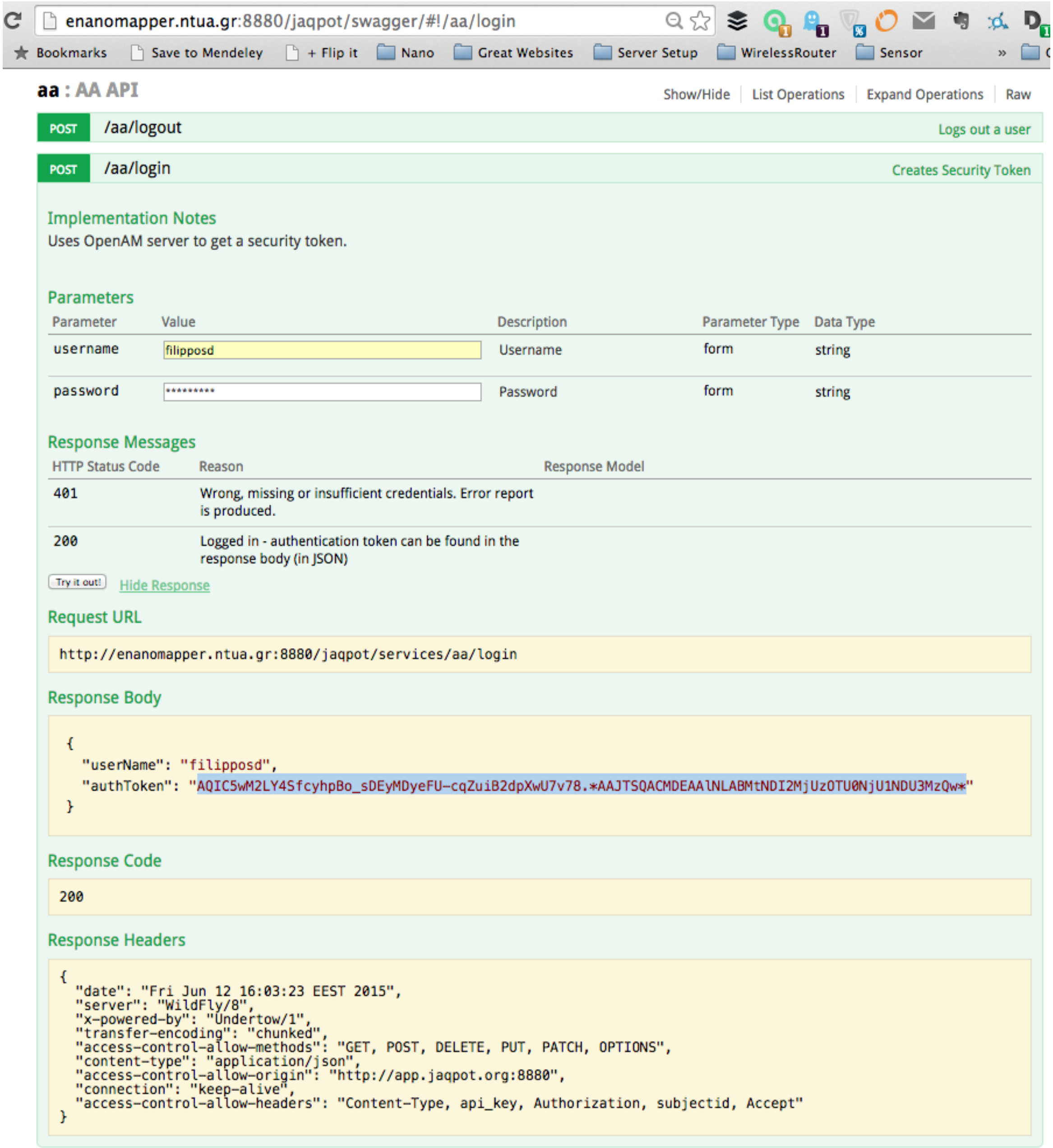
1. Balbin, O. A.; Prensner, J. R.; Sahu, A.; Yocum, A.; Shankar, S.; Malik, R.; Fermin, D.; Dhanasekaran, S. M.; Chandler, B.; Thomas, D.; et al. Reconstructing targetable pathways in lung cancer by integrating diverse omics data. Nat. Commun. 2013, 4, 2617 DOI: 10.1038/ncomms3617.
2. Berman, H. M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T. N.; Weissig, H.; Shindyalov, I. N.; Bourne, P. E. The Protein Data Bank. Nucleic Acids Res. 2000, 28, 235–242 DOI: 10.1093/nar/28.1.235.
3. Burello, E.; Worth, A. P. A theoretical framework for predicting the oxidative stress potential of oxide nanoparticles. Nanotoxicology 2011, 5, 228–235 DOI: 10.3109/17435390.2010.502980.
4. Chin, L.; Gray, J. W. Translating insights from the cancer genome into clinical practice. Nature 2008, 452, 553–563 DOI: 10.1038/nature06914.
5. Ferreira, T.; Rasband, W. ImageJ User Guide: IJ 1.45m http://imagej.nih.gov/ij/docs/guide.
6. Fourches, D.; Pu, D.; Tassa, C.; Weissleder, R.; Shaw, S. Y.; Mumper, R. J.; Tropsha, A. Quantitative nanostructure-activity relationship modeling. ACS Nano 2010, 4, 5703–5712 DOI: 10.1021/nn1013484.
7. Gajewicz, A.; Rasulev, B.; Dinadayalane, T. C.; Urbaszek, P.; Puzyn, T.; Leszczynska, D.; Leszczynski, J. Advancing risk assessment of engineered nanomaterials: application of computational approaches. Adv. Drug Deliv. Rev. 2012, 64, 1663–1693 DOI: 10.1016/j.addr.2012.05.014.
8. Gajewicz, A.; Schaeublin, N.; Rasulev, B.; Hussain, S.; Leszczynska, D.; Puzyn, T.; Leszczynski, J. Towards understanding mechanisms governing cytotoxicity of metal oxides nanoparticles: Hints from nano-QSAR studies. Nanotoxicology 2015, 9, 313–325 DOI: 10.3109/17435390.2014.930195.
9. Ge, C.; Du, J.; Zhao, L.; Wang, L.; Liu, Y.; Li, D.; Yang, Y.; Zhou, R.; Zhao, Y.; Chai, Z.; et al. Binding of blood proteins to carbon nanotubes reduces cytotoxicity. PNAS 2011, 108(41), 16968-16973.
10. Hastings J, Chepelev L, Willighagen E, Adams N, Steinbeck C, Dumontier M (2011) The Chemical Information Ontology: Provenance and Disambiguation for Chemical Data on the Biological Semantic Web. PLoS ONE 6(10): e25513. doi:10.1371/journal.pone.0025513
11. Lesniak, A.; Fenaroli, F.; Monopoli, M.P.; Aberg, C.; Dawson, K.A.; Salvati, A. Effects of the presence or absence of a protein corona on silica nanoparticle uptake and impact on cells. ACS nano 2012 6(7), 5845-5857.
12. Liu, R.; Rallo, R.; George, S.; Ji, Z.; Nair, S.; Nel, A. E.; Cohen, Y. Classification NanoSAR development for cytotoxicity of metal oxide nanoparticles. Small 2011, 7, 1118–1126 DOI: 10.1002/smll.201002366.
13. McDermott, J.E.; Wang, J.; Mitchell, H.; et al. Challenges in Biomarker Discovery: Combining Expert Insights with Statistical Analysis of Complex Omics Data. Expert opinion on medical diagnostics. 2013, 7(1):37-51 DOI: 10.1517/17530059.2012.718329.
14. Nel, A.E.; Madler, L.; Velegol, D.; Xia, T.; Hoek, E.M.V.; Somasundaran, P.; et al. Understanding biophysicochemical interactions at the nano-bio interface. Nat.Mater 2009, 8, 543–557 DOI: 10.1038/nmat2442
15. Pearson, R.M.; Juettner, V.V; Hong, S. Biomolecular corona on nanoparticles: a survey of recent literature and its implications in targeted drug delivery. Frontiers in chemistry 2014, 2 DOI: 10.3389/fchem.2014.00108.
16. Puzyn, T.; Rasulev, B.; Gajewicz, A.; Hu, X.; Dasari, T. P.; Michalkova, A.; Hwang, H.-M.; Toropov, A.; Leszczynska, D.; Leszczynski, J. Using nano-QSAR to predict the cytotoxicity of metal oxide nanoparticles. Nat. Nanotechnol. 2011, 6, 175–178 DOI: 10.1038/nnano.2011.10.
17. Rasband, W.S.,ImageJ, U. S. National Institutes of Health, Bethesda, Maryland, USA, http://imagej.nih.gov/ij/, 1997-2014.
18. Steinbeck C., Hoppe C., Kuhn S., Floris M., Guha R., Willighagen E.L. Recent Developments of the Chemistry Development Kit (CDK) - An Open-Source Java Library for Chemo- and Bioinformatics. Curr. Pharm. Des. 2006; 12(17):2111-2120 DOI: 10.2174/138161206777585274.
19. Steinbeck, C.; Han, Y.; Kuhn, S.; Horlacher, O.; Luttmann, E.; Willighagen, E. The Chemistry Development Kit (CDK): an open-source Java library for Chemo- and Bioinformatics. J. Chem. Inf. Comput. Sci. 2003, 43, 493–500 DOI: 10.1021/ci025584y.
20. Subramanian, A.; Subramanian, A.; Tamayo, P.; Tamayo, P.; Mootha, V. K.; Mootha, V. K.; Mukherjee, S.; Mukherjee, S.; Ebert, B. L.; Ebert, B. L.; et al. Gene set enrichment analysis: a knowledge-based approach for

interpreting genome-wide expression profiles. Proc. Natl. Acad. Sci. U. S. A. 2005, 102, 15545–15550 DOI: 10.1073/pnas.0506580102.

21. Taylor-Pashow, K. M. L.; Rocca, J. Della; Lin, W. Mesoporous Silica Nanoparticles with Co-Condensed Gadolinium Chelates for Multimodal Imaging. Nanomaterials 2011, 2, 1 DOI: 10.3390/nano2010001. Image used under CC BY 3.0 license https://creativecommons.org/licenses/by/3.0/.
22. Walkey, C. D.; Olsen, J. B.; Song, F.; Liu, R.; Guo, H.; Olsen, D. W. H.; Cohen, Y.; Emili, A.; Chan, W. C. W. Protein corona fingerprinting predicts the cellular interaction of gold and silver nanoparticles. ACS Nano 2014, 8, 2439–2455 DOI: 10.1021/nn406018q.
23. Wiggins, D.P., Xvfb − virtual framebuffer X server for X Version 11, The Open Group, Inc. http://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml
24. Winkler, D. a; Mombelli, E.; Pietroiusti, A.; Tran, L.; Worth, A.; Fadeel, B.; McCall, M. J. Applying quantitative structure-activity relationship approaches to nanotoxicology: current status and future potential. Toxicology 2013, 313, 15–23 DOI: 10.1016/j.tox.2012.11.005.
25. Yang, X.; Regan, K.; Huang, Y.; Zhang, Q.; Li, J.; Seiwert, T. Y.; Cohen, E. E. W.; Xing, H. R.; Lussier, Y. A. Single sample expression-anchored mechanisms predict survival in head and neck cancer. PLoS Comput. Biol. 2012, 8 DOI: 10.1371/journal.pcbi.1002350.

9. ANNEX I: OBTAINING A TOKEN THROUGH THE SWAGGER INTERFACE

A token can be retrieved, among other ways, through the Swagger interface at: http://enanomapper.ntua.gr:8080/jaqpot/swagger/#!/aa/login by using the user's credentials, as shown in Figure 11.



aa : AA API

POST /aa/logout — Logs out a user

POST /aa/login — Creates Security Token

Implementation Notes

Uses OpenAM server to get a security token.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
username	filipposd	Username	form	string
password	*********	Password	form	string

Response Messages

HTTP Status Code	Reason	Response Model
401	Wrong, missing or insufficient credentials. Error report is produced.	
200	Logged in - authentication token can be found in the response body (in JSON)	

Request URL

```
http://enanomapper.ntua.gr:8880/jaqpot/services/aa/login
```

Response Body

```
{
  "userName": "filipposd",
  "authToken": "AQIC5wM2LY4SfcyhpBo_sDEyMDyeFU-cqZuiB2dpXwU7v78.*AAJTSQACMDEAAlNLABMtNDI2MjUzOTU0NjU1NDU3MzQw*"
}
```

Response Code

```
200
```

Response Headers

```
{
  "date": "Fri Jun 12 16:03:23 EEST 2015",
  "server": "WildFly/8",
  "x-powered-by": "Undertow/1",
  "transfer-encoding": "chunked",
  "access-control-allow-methods": "GET, POST, DELETE, PUT, PATCH, OPTIONS",
  "content-type": "application/json",
  "access-control-allow-origin": "http://app.jaqpot.org:8880",
  "connection": "keep-alive",
  "access-control-allow-headers": "Content-Type, api_key, Authorization, subjectid, Accept"
}
```

Figure 11 Swagger interface for tokens

10.ANNEX II - TECHNICAL ARCHITECTURE OF THE IMAGE ANALYSIS WEB APPLICATION

This annex examines in great detail the main tools and the practices followed towards the creation of such an application, the challenges encountered throughout the development process and the decisions that had to be made in order to overcome them. It should be emphasized that all the tools used in it through the development process, namely all servers, libraries and third party dependencies are completely open source, thus enabling us to provide services and source code open to the nanosafety community for exploitation.

TOOLS OVERVIEW

Given that image analysis is a really popular field among professionals, more and more algorithms, libraries and methods are being developed with the aim of extracting information from desired images. All these methods are coded in programming languages different in nature and properties. Each and every language is unique and its advantages and disadvantages really change not only the way the problem is solved but also sometimes the whole architecture of an application. Specifically, in this implementation the language selected was Java and the components of this application are:

Programming Language	Java
Build Automation Tool	Maven
Server	Glassfish and WildFly
Virtual Window Server	Xvfb
Image Analysis Library	ImageJ
User Interface	Primefaces
Sub versioning	Git

A more detailed analysis of the reasons the specific tools were chosen will be provided next.

CHOICE OF PROGRAMMING LANGUAGE - JAVA

We ought to point out a few features of Java. Java is a strict type, general purpose programming language that is concurrent, object-oriented and based on classes, initially created in order to help programmers build applications able to run “everywhere”, which is achieved because Java Applications and Java code in general are compiled in bytecode that can run on any Java virtual machine regardless of the computer architecture. Java SE 7 was used inside our application and one of the main reasons is that ImageJ, the image processing library we chose, is stable and tested running on Java 7. Another practical reason is that this version is stable, most of its defects have been solved, the community that would have to implement our tools is aware of any system exceptions and how to handle them and last but definitely not least, most of the third party libraries have already adapted to Java 7. As a result, by choosing this version we had numerous privileges that we would certainly not have if we chose the next Java version (Java 8).

BUILD AUTOMATION TOOL - APACHE MAVEN

In order to handle the dependencies, the build management and the deployment of the project efficiently, a build automation tool was needed. In our case that tool was Maven. Maven is a build automation tool that is extremely popular in Java projects. The main advantages of implementing such a tool inside a project are the following:

- Minimization of the bad builds
- Elimination of redundant tasks
- Acceleration of the compile time
- Elimination of manually- handled dependencies
- Ability to have history of builds

Finally, using Maven offers an additional privilege/benefit. When a project is built as a Maven project then it can easily publish an artifact id and then be smoothly imported in any other bigger project. This was a particularly significant feature for our application as we targeted it to be a stand-alone application but also able to be part of bigger applications.

SERVERS

HTTP Servers - Glassfish and WildFly

In order to cover the need for multiple simultaneous users in this application there were two main ideas: Either to build a stand-alone application able to be installed and run from a personal computer in order for each researcher to be completely independent, or to create a web application, which could be expanded to a web service. The choice of the second proposal was really easy. Publishing this application to the web would make it easier for more and more users to visit the web site, not being attached to a physical machine and in the meantime test how easy it would be to provide a web based solution.

This choice prompted us to set up an http server. In order to comply with the open source ideology and provide stability and support, Glassfish was chosen as the http server. Glassfish is most of the time used as a reference as one of the simplest, most stable and robust servers. It was created as an implementation for Java EE and as such it is completely compatible with Java Server Faces (JSF).

At first Glassfish was hosted locally in testing and development machines. When the whole application began to grow and take its final form, a migration took place. The next http server we migrated to was WildFly, formerly known as JBoss. It is completely written in Java and it supports Java EE too.

Virtual Window Server - Xvfb

During the description of ImageJ, it was mentioned that ImageJ was initially built as a native application. The main feature of this application was the ability to execute it in multiple environments regardless of the operating system. Attempting to run ImageJ on a server was an unknown and poor in previous documentation idea. One of the main challenges of this kind of implementation was the need for graphical interface from the side of ImageJ.

In order to be consistent with the chapter structure no details of what triggered this challenge should be mentioned here and neither what the commands in order to begin this server are. Xvfb or X virtual frame buffer is a display server implementing the X11 display server protocol. Primarily Xvfb intended to be used for testing purposes. The community though, found many practical and novel applications. One of them was to be used as a background-rendering engine.

The special thing about Xvfb is that it performs all graphical operations in memory without showing any screen output. The outputs, screen requests served by XVFB are handled like any other in any other X server. However, no output is shown. Like every other server it does not require from the computer it is running on to have any output or input device. As a result, using Xvfb adds an unobtrusive way to run applications and programs that don't really need an X server but require one to operate. The only thing necessary to establish communication is a network layer.

Some of the main features of this server are the following:

- Really easy installation
- Easy command to begin and stop
- Straightforward commands to customize initiation
- SSH Communication

APPLICATION ARCHITECTURE

The architecture of this system was designed influenced mainly by practices and patterns followed by professionals but primarily by the need of making a distinction among the structural and functional entities of the system. Due to that, functionalities were separated according to their objective target and structures that handled and stored information were also customized and separated, in order to provide an optimized result and the required readability every project needs. The complete architecture is displayed in **Error! Reference source not found.** with small shapes. In order to make the role of modules more distinct and readable, inside this ecosystem different shapes where chosen in order to show the place and role of each element. The circular shape in the bottom illustrates the source core of the application, ImageJ. Most of the functionalities used and implemented in this application made complete use of generic algorithms of ImageJ. Thus, most of the back end process "work" depends on the library and in quite a few cases one module of the application sends data into ImageJ for processing and after that it returns them into another module for further processing.

As a result, ImageJ is the "heart" of the application and currently the greatest dependency. The other shapes used in the following image are circle for the helpers, square for the models, rectangular for the managed beans and rhomb for the views. The whole process from capturing user input (views and managed beans) into processing and delivering it to ImageJ is presented in the following figure as well.

Helpers like Process, File and CSV are responsible for processing and returning information to the main branch of the application. On the other hand, *ApplicationMain* helper is only responsible for triggering the right methods in other helpers. In this architecture, *ApplicationMain* is the switch and the helpers are the endpoints.

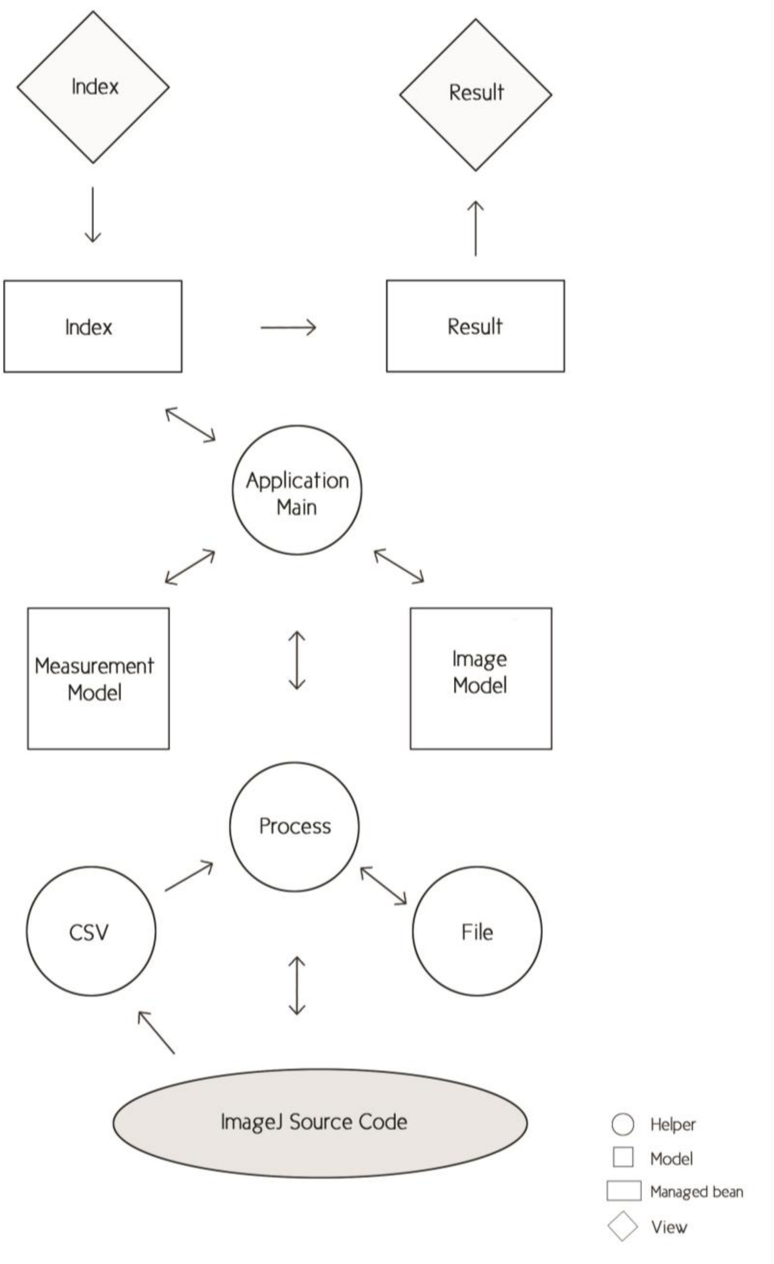


Figure 12 Application Architecture

MVC

In this application, the MVC (Model – View – Controller) design pattern was followed. Due to the fact that more and more developers tend to follow the architectural guidelines of MVC, the application results tend to be more modular and with distinctly-appointed roles.

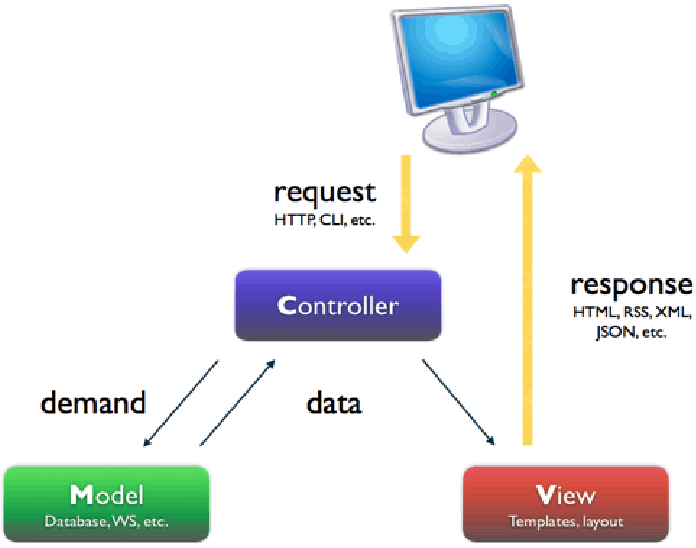


Figure 13 MVC design pattern description

So in this application, MVC is applied in the following way:

Models are completely responsible for storing the data and informing the controllers and the views for any change that may happen in them in order to trigger and update in view display or controllers set of commands.

Controllers are responsible to deliver messages between views and models, which are going to trigger the related updates in these two entities.

Views are only responsible for displaying information and updating the user interface according to the manipulation of the model.

In order to have an optimized and modern application structure, it is advisable to not only follow the guidelines of MVC design pattern but also to bear in mind that these patterns are only valuable when controllers really work as messengers and have as least functionality as possible. On the other hand, models should handle every data manipulation. In order to comply with these practices another entity was introduced. This entity was helpers. A helper is a level between controller and model whose aim is to emphasize the distinction of the roles. Currently models are used in order to save data and handle a small amount of functionality. Controllers handle the requests from the views and trigger functionalities inside the helpers. Helpers find all the required models and combine their data. Subsequently, helpers make the processing of the combined data with requests of the ImageJ. Eventually, the result is returned to controllers and from there to views, which are updated. Helpers play a huge role inside this application and this decision was made in order to keep models and controllers clean, from an overlapping and ambiguous type of functionality that would not be easily separated according to the pattern guidelines.

Respectively the helpers created and implemented inside the application are the following:

- Process-Helper
- Validation-Helper
- Analysis-Helper
- Constant-Helper

- File-Helper
- ImageModel-Helper
- CSV-Helper

The names of the Helpers indicate that they include and implement the key functionalities of the application.

Apart from the helpers, another major category of entities inside this application are the models. As it was mentioned before, models were built with the strict perception of storing data with small exceptions of processing their own data when needed. The list of the models implemented is the following:

- Image Model
- Result Model
- Measurement Model
- Nanoparticle Result Model

Last but not least, we ought to mention the managed beans and the html views that implement the JSF (Java Server Faces) part of the application. This was the last pillar that made the application completely compatible with http requests and provided an easy way to use and update the user interface. Managed beans contain the “business logic” and all the getters and setters needed to populate the views (html views) with the result of the back end process. Currently, managed beans handle the form submission from the view, trigger and update elements in view like the preview image and are responsible for all the updates of the user interface. For example, the redirection in the detail page, the loading of values inside the result data table and many more.