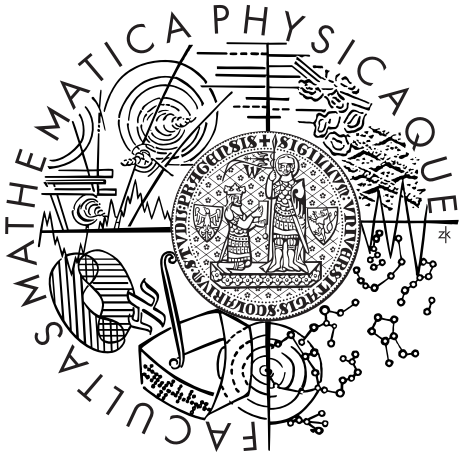
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Khanh Chuong Le

DEBRA - Descriptor Barn

Department of Software Engineering

Supervisor of the bachelor thesis: RNDr. David Hoksza, Ph.D.
Study programme: Computer Science
Specialization: General Computer Science

Prague 2013

I would like to express sincere gratitude to my supervisor RNDr. David Hoksza, Ph.D., for his immense patience throughout my work on this thesis, doc. Daniel Svozil, Ph.D. who showed me some useful cheminformatics softwares and also Jan Kofroň, Ph.D. and RNDr. Petr Hnětynka, Ph.D. who exposed me beauty of Java. I would also like to thank my partner and my family for their limitless support.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature of the author

Název práce: DEBRA - Descriptor Barn

Autor: Khanh Chuong Le

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. David Hoksza, Ph.D.

Abstrakt: Jedním z klíčových problému cheminformatiky je reprezentace chemických sloučenin. S touto prací je vytvořena klient-server aplikace, která umí spočítat velké množství takových reprezentací. Server umožňuje vkládání modulů, které využívají aplikace třetích stran ke spočítání reprezentací. Tyto aplikace můžeme spouštět přes příkazový řádek, nebo voláním metod webových služeb. Klient zobrazí možné typy reprezentaci sloučenin. Uživatel si může vybrat množinu reprezentací a vstupní množinu molekul soplu a tento požadavek zašle serveru. Server potom zavolá moduly a nazpět zašle výsledek.

Klíčová slova: molekulární deskriptory, framework, klient-server

Title: DEBRA - Descriptor Barn

Author: Khanh Chuong Le

Department: Department of Software Engineering

Supervisor: RNDr. David Hoksza, Ph.D.

Abstract: One of the key tasks of cheminformatics is representation of chemical compounds. In this thesis we develop a client server software that can generate large amount of such representations. The server allows adding plugins which exploit third party applications. These applications are launched via a command line interface, or calling methods of a web service. The client displays the possible representation types. The user will choose a set of representations and an input set of molecules and this query sends to the server. The server then calls plugins and return a result.

Keywords: molecular descriptors, framework, client-server

Contents

Introduction

Over the last few decades, information technology and computer driven systems have helped scientists discover new drugs. The initial phase of the drug discovery process is taken place in a virtual environment, which lets researchers manipulate with the chemical structures on the computer screens to create new compounds having given properties. E.g., one might want to find a certain chemical structure that is similar to the target structure but having different side effects. This notion is based on *similar property principle* of Johnson and Maggiora, which states: *similar compounds should have similar biological properties* [16]. In the best case, we hope that the new found chemical structure produces no effects or less critical effects on a consumer.

The cheminformatics databases (such as Pubchem [8], ChEMBL [12], ZINC [15]) work with massive amounts od data. These data are analyzed, organized and further investigated to give new insights for chemical researches. *In silico*[1] analysis also reduces the need for expensive laboratory experiments and clinical trials but on the other hand it also may speed up the discovery of new drugs.

The aim of this thesis is to implement such a framework that enables user to select input molecules, select chemical descriptors and get the representations. More specifically, it contains

1. the implementation of the graphical user interface that displays a set of available chemical descriptors from plugins and the user can select the files and the descriptors and make requests to the server,

2. the implementation of the server side application that can dynamically load plugins which call third party applications, e.g. through their command line interface, or call web services.

The structure of the thesis follows,

Chapter 1 briefly introduces the reader to the fundamental basis of cheminformatics.

Chapter 2 describes other developed softwares. Some of them will be integrated as plugins to framework.

Chapter 3 discusses about applied methods and the problems I encountered while developing this framework.

Chapter 4 describes architecture and its implementation.

Chapter 5 contains user documentation and an example of plugin implementation.

And the whole work is concluded in **Conculusion** with future improvement possibilities.

[1] To perform in a virtual simulation on computer.

1. Theoretical Background

There are atoms and space. Everything else is opinion.
– Democritus

Cheminformatics [9] (or chemoinformatics) is an interdisciplinary research field between Computer Science and Chemistry. One of its main tasks that cheminformatics faces are storing and retrieving information about chemical compounds. These chemical information are stored in *Information Systems* embedded with storing, retrieving and searching algorithms.

Searching in cheminformatics has following main tasks:

1. Find a way to represent chemical compounds and store them in a database.
2. Search for an exact molecule (also called *object*, *compound*, *structure*)[1].
3. Search for compounds similar to a given structure, ranked by an objective (similarity) function.
4. Search for all compounds that contain a partial molecule (*substructure*, *pattern*, *feature* or *target*) specified by the user. The substructure might be a functional group, or a core structure representing a class of molecules.

For our purpose we can think of molecules as weighted graphs. Its nodes are atoms and its edges with weights are the covalent bonds between atoms. Then a substructure search can be formulated as an isomorphism problem:

> *Two graphs S and T are given as input, determine whether structure S contains a subgraph that is isomorphic to target T.*

This problem is known to be in a low hierarchy of NP-complete class. Nonetheless, it's lucky that the real substructure search in cheminformatics has computational complexity of $O(N^2)$ or $O(N^3)$ most of the times. Even though the detection of a feature in a structure runs generally in the non-polynomial time, we can detect an absence of a substructure much faster, often in the linear $O(N)$ time. The detection of an absence uses structural keys.

One of the basic ways to represent the molecules is called *valence model* of chemistry. A molecule is represented as atoms joined by semi-rigid bonds that can be single, double, or triple according to the rules of *valence bond theory*. From this observation, computer scientists transform the valence model into a graph, where the nodes are atoms and the edges are bonds.

However, the valence model has many shortcomings. For example the benzene in Figure 1.1 has molecular formula C_6H_6 and it obviously lacks the information

[1] There is technical difference between molecules and compounds. The former one is formed when two or more atoms of an element bond together. If the types of atoms are different from each other, a compound is formed. In the manner of this thesis I will interchange these terms.

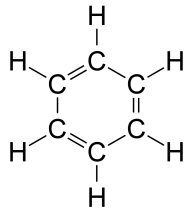


Figure 1.1: Benzene

about aromaticity.

In any cheminformatics process we need to choose between computer molecular representations [18]:

Graph-based representation is based on storing full information available in the 2D structural molecule of the compound. We can also store molecules in 3D structure, in which case we need to add 3D coordinates of each nodes of the graph. The graph-based representation turned out to be clumsy when similarity search in large database is engaged.

Structural and hashed fingerprints representations are faster when we need to find similar structures or screen out the undesirable compounds that lack the specified feature. It is based on summarizing chemical structures into binary digits indicating the presence or absence of a particular chemical aspect (atom, molecule). We can classify the fingerprints into two groups based on their features – a *structural fingerprint* (or *structural key*) and *hashed fingerprint*. The former one is based on the predefined dictionary of the chosen features while the latter one is based on the number of particular feature in the compound such as all possible labeled paths up to a certain depth.

1.1 Graph-based Representations

It is essential for a cheminformatics application to represent a molecule using line notation (1D) or a connection table (2D and 3D). The former one is compact and employs alphanumeric characters and conventions for common molecular features such as bond types, ring system, aromaticity and chirality[2] [11], whereas the latter one is more outstretched in the mean of the storage space. Each line specify individual atoms and bonds, and can be created and parsed by computer, and is human-readable text file.

1.1.1 Linear Notations

The linear notation is compact and enables easy interpretation by computer programs. It is basically a single-line string of characters. The chemical line notation

[2] *Chirality* is derived from Greek term for *hand*. Chiral object is not identical to its mirror image and cannot be superposed onto it. It can be caused by central carbon atom, which is bonded to four different atoms. Two non-superposed images are often referred as right- and left-handed.

can be parsed by using string processing algorithms resulting in efficient cheminformatics applications [11]. Some well known notations are listed below.

WLN means *Wiswesser Line Notation* and was popular from 1950s until the introduction of SMILES in 1980s. It is named after William J. Wisswesser. The notation was restricted to using uppercase letters, the digits 0-9, and few special characters like '&'. The letters were reserved for functional groups and molecular features while an alphanumeric combination presented a fragment-base description of the molecules. For example, the acetone CH_3-CO-CH_3 in WLN representation is 1V1, where V is the character used for central carbonyl group and the digit 1 indicates the presence of sutured single-carbon atom chains on either side. Another example the methane CH_4 in WLN is 1H, the ethane CH_3-CH_3 is 2H, notation 3H means the propane CH_3-CH_2-CH_3 and the benzene ring C_6H_6 is simply denoted as R.

SMILES is a neat abbreviation for *Simplified Molecular Input Line Entry System*[3] using ASCII characters. SMILES strings can be converted back into two-dimensional models of the molecules. Extended SMILES strings can represent either the molecules themselves or reactions. It is true language with a simple vocabulary (atom, bonds, chains, aromaticity and some other symbols) and only with few grammar rules. The table 1.1 lists some examples of compounds in SMILES.

SMILES	**Name**
CC	ethane
O=C=O	carbon dioxide
C#N	hydrogen cyanide
C1CCCCC1	cyclohexane
c1ccccc1	benzene
N[C@H](C)C(=O)O	D-alanine

Reaction SMILES	**Name**
[I-].[Na+].C=CCBr>>[Na+].[Br-].C=CCI	displacement reaction
(C(=O)O).(OCC)>>(C(=O)OCC).(O)	intermolecular esterification

Table 1.1: Example of SMILES taken from Daylight website[4]

Canonical SMILES. While storing a chemical structure in a database, one should think of the redundancy of chemical structures, it means that one should check whether or not the same structures are presented in a database but in different ordering of atoms. In graph theory we refer this as an isomorph problem, see page 4. There is "a bypass" for a chemical nomenclature systems such as SMILES, these systems do so called *canonical labeling* of the atoms and bonds. This is generated whenever a new SMILES string is required. Algorithm can be found in [22] or in [11].

[3]The original specification was initiated by David Weininger in 1980s. The open version of the standard can be found on `http://www.opensmiles.org/`

InChI or International Chemical Identifier is the latest and the most modern of the line notations. It resolves many of the chemical ambiguities not addressed by SMILES, particularly with the respect to stereo centers, tautomers and other of the "valence model problems". It was developed at IUPAC and NIST starting in 2000[4][11]. Many cheminformatics databases provide searchable InChI Keys, it is basically a hashed InChIs. An example of InChI and its key for the benzene molecule is listed below, in the Table 1.2.

InChI	1S/C6H6/c1-2-4-6-5-3-1/h1-6H
Key	UHOVQNZJYSORNB-UHFFFAOYSA-N

Table 1.2: InChI and InChIKey of the benzene

1.1.2 Connection Table

A connection table or CTAB is a widely used representation of the molecular graph. A simple connection table contains string lines of the atoms and the bonds and the bond ordersr[5]. For instance, the benzene (from Figure 1.1) can be expressed as in Table 1.3, it is of the form where the hydrogen is suppressed. The first line contains two numbers: the first one indicates the number of atoms, n, and the following number of bonds, m. The next n lines comprise the atoms block, and list atomic coordinates and the atom types in the molecule. These are followed by the bonds block containing m lines with the atom numbers and the bond order as third number in a row. This example is indeed for 2D chemical diagram of the benzene, the connection table can be easily extended to represent 3D structures, in which case the z-coordinates are likely to have nonzero values.

```
6  6
  0.8660    0.5000    0.0000 C
  0.8660   -0.5000    0.0000 C
  0.0000    1.0000    0.0000 C
 -0.0000   -1.0000    0.0000 C
 -0.8660    0.5000    0.0000 C
 -0.8660   -0.5000    0.0000 C
1  2  2
1  3  1
2  4  1
3  5  2
4  6  2
5  6  1
```

Table 1.3: Connection table example of the benzene

Popular molecular file formats that are based on connection tables such as MOL, SDF (structure-data format)[6] and MOL2[7] format. These formats enrich

[4]Information and stable version is available on website `http://www.iupac.org/inchi`

[5]The bond number is number of chemical bonds between a pair of atoms

[6]Created by Symyx and now as part of Accelrys `http://accelrys.com/`

[7]MOL2 format from Tripos company, `http://www.tripos.com/`

connection tables with additional information such as the header, the format's version, the structure's weight, SMILES string, the chemical formula, the chemical name, the chiral center and etc. Most cheminformatics databases tailor file formats to their specific needs and they usually append addition information to the connection table such as a charge, the isotopes, and the stereochemistry[8]. Of course there are plenty of other molecular file formats, we can name CML[9] and HIN[10]. For visualization purposes there are some tools that enable to view the diagram of structures, for instance Marvin[6] or Bioclipse[11].

1.2 Structural Keys and Hashed Fingerprints

In the process of understanding the information and the knowledge from real-world data, it is important to define properties that differentiate certain objects from others. Therefore, an explicit definition of formal description of such objects is needed. These description vary on context of interests. A group of pharmacologists looking for a drug curing cancer might want different descriptions compared to dentists who want to invent whiter whitening toothpaste. For this reason, literally thousands of molecular descriptors have been proposed covering all properties of interest[11], many of important descriptors can be found in *Handbook of Molecular Descriptors* of the authors Todeschini and Consonni[**?**]. Here in this thesis we present only structural and hashed fingerprints.

One of the most popular methods for similarity search in large chemical compound databases is based on mapping the molecule to one or several numerical values. This popular method employs a set of substructures also known as *structural keys* and these are considered as descriptors. An alternative approach would use an algorithm that generates a set of substructures for every single molecule. In such case there is no need to explicitly define the substructure set and therefore a few interesting structural features may raise to the surface.

1.2.1 Structural Keys

In order to form *structural keys*, also known as *nonhashed fingerprints*, one decides which features are important, assigns a bit position for each feature, then generates a fixed-length bitmap for each molecule. It means that there is a unique mapping between a bit position and a specific feature. At the end of the 1970s, the MACCS[12] were born and have remained popular until today. MACCS keys [10] cover the most important structural keys.

When searching for a particular feature in a database a structural key is calculated for the feature and compared with each pre-computed key of molecules. If

[8] *Stereochemistry, a subdiscipline of chemistry, involves the study of the relative spatial arrangement of atoms that form the structure of molecules and their manipulation.* WIKIPEDIA

[9] CML stands for Chemical Markup Language, it is XML-based molecular file format

[10] From HyperChem

[11] Java based `http://www.bioclipse.net/`

[12] MACCS key is acronym for Molecular ACCess System key that was first of its kind and was part of a system which could store and retrieve molecules also as graphical structures.

Procedure 1 Get paths based on molecular graph M and search depth d

List $paths_d$ {list of unique paths of length d}
List $paths_{local}$ {temporary variable, it contains paths for each node in M}
for all atom $a \in M$ **do**
  $root$ = get all paths of length d via DFS from a
  $paths_{local}$ = get paths in the depth-first tree from the $root$ up to length d
  for all $path \in paths_{local}$ **do**
    {in case of cyclicity, check also the reverse sequence}
    if $(path \notin path_d)$ && $(path.reverse() \notin path_d)$ **then**
      $path_d.add(path)$
    end if
  end for
end for
return $paths_d$

any TRUE bit in the feature's key is not also a TRUE in the molecule's key, then we can say for sure that the molecule lacks the feature. This method of rejecting is called *screening*[13] and can vastly lower the time complexity of similarity or substructure search problem [4].

One can see that the choice of the features suffer from the lack of generality and literally depends on the queries to be made. The search speed across the database and reliability of similarity stands and falls on the effective choice of the key features. It is a product of the trade-off between an accuracy and a storage. The more bits in the structural keys, the better is the chance to screen out unwanted molecules and thus avoid full substructure search.

1.2.2 Hashed Fingerprints

Hashed Fingerprints or simply *fingerprints*[14] doe not suffer from this lack of generality by eliminating the idea of predefined patters. Unlike the structural keys with its predefined features, the features for molecular fingerprints are generated from the molecule itself. The fingerprinting algorithm examines the molecule and generates the following:

- a feature for each atom (the paths of the length 0),
- a feature representing each atom and its nearest neighbors (plus the bonds that join them) (the paths of the length 1),
- a feature representing each group of atoms and bonds connected by paths up to 2 bonds long,
- ... atoms and bonds connected by paths up to 3 bonds long,
- ... continuing, with paths up to 4, 5, 6, and 7 bonds long.

[13] Screening in this content is not the same as *virtual screening*.

[14] Reader should not be confused, fingerprints refer to hashed fingerprints and structural keys refer to nonhashed fingerprints.

Procedure 2 Get hashed path fingerprint from molecule M, size d and length L

Require: fixed number N
1: *fingerprint* = initialize bit map of length d
2: *paths* = get paths from Procedure 1 with M and path length L
3: **for all** atom $a \in M$ **do**
4: **for all** $path \in paths$ starting at a **do**
5: *seed* = generate an integer hash for *path*
6: *randomIntSet* = generate a set of size N of random integers with *seed*
7: **for all** $rInt \in randomIntSet$ **do**
8: $index = rInt \pmod d$
9: *fingerprint*[*index*]=TRUE
10: **end for**
11: **end for**
12: **end for**
13: **return** *fingerprint*

The algorithm finds all the unique labeled paths up to the length d in the molecule M. In which the algorithm builds a depth-first tree for each node (atom), in which all paths from the root to a leaf is of length d. The pseudo-code is presented in Procedure 1 and it was adapted from [11], in which depth-first traversal was used in the point 4 of the algorithm, we could substitute for the breadth-first traversal and the algorithm will still produce the same set of paths.

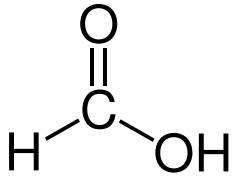


Figure 1.2: Diagram of formic acid

For example, the molecule of formic acid CH_2O_2 on Figure 1.2 would generate features in Table 1.4.

0-bond paths	H	C	O	
1-bond paths	HC	$C=0$	$C-O$	OH
2-bond paths	$HC=O$	$HC-O$	$O=C-O$	$C-OH$
3-bond paths	$HC-OH$	$O=C-OH$		

Table 1.4: Generated features for formic acid CH_2O_2

And then these features from Procedure 1 are hashed into binary numbers by a pseudo-random number generator and added (with logical OR) to produce the final fingerprint. The pseudo-code is shown in Procedure 2

Hashed fingerprints have several advantages over structural keys:

- Since hashed fingerprints have no pre-defined set of features, one fingerprinting system serves all databases and all types of queries.

- More effective use of the bitmap. Structural keys are usually "sparse" (mostly zeros) since a typical molecule has very few of the features that the structural key's bits represent.

- The features that go into a fingerprint are highly overlapped – except for "lone atoms", each feature shares portions of itself with at least one other feature. The result is that the more complex a molecule gets, the more accurately its fingerprint representation characterizes it.

1.3 Similarity Measures

In this section I will describe how the system can compute the similarity rank. Here comes a general question: "What the term rank means and how is it correlated to the similarity search problem?"

We often need to know how much similar are two molecules. This value is called *rank* and there are many approaches to measure such similarity, dependent both on the chosen measure and the molecular representation. These approaches might be diverse and it is upon us to choose what fits to our needs. One might rank the similarity according to their structural similarity (bonds, atoms), and others might rank the similarity according to the physical properties.

Now we will talk about bit-based methods. Our aim is to numerically measure such similarity and the best way we can do it, is to compare the bits in the fingerprints which are set TRUE. Then we can say that we want all the structures that have the similarity rank above the threshold constant. It is important to state that the similarity measures are independent of the molecular feature descriptors. There are plenty of distance measures also as similarity coefficients. I introduce here only basic idea.

If we describe our molecules by the presence or absence of features, then the binary association coefficients or similarity measures are based on the four terms a, b, c, d shown in Table 1.5.

		Object B		
		FALSE bits	TRUE bits	Totals
Object A	FALSE bits	d	b	$b+d$
	TRUE bits	a	c	$a+c=A$
	Totals	$a+d$	$b+c=B$	n

Table 1.5: Binary association coefficients

Where:
a is the count of bits TRUE in the object A but FALSE in the object B,
b is the count of bits TRUE in the object B but FALSE in the object A,
c is the count of the bits TRUE in both objects,
d is the count of the bits FALSE in both objects,

and the term $a+b+c+d$ is the total number of bits TRUE or FALSE in objects A or B.

Some commonly used similarity coefficients in cheminformatics:

Euclidian coefficient is defined as the $\sqrt{\frac{c+d}{n}}$

Total Difference coefficient, or Hamming distance is defined as $\frac{a+b}{n}$

Tanimoto-Jaccard coefficient of similarity is defined as $\frac{c}{a+b+c}$. It is independent of d, by other words, it is the proportion of the TRUE bits which are shared by both objects.

Dice (Czekanowski-Sorenson) coefficient is defined as $\frac{2c}{n}$.

... so forth, indeed the user can define his own similarity coefficient.

For further reading I encourage a reader to look at [11] for more references about this topic.

2. Third Party Applications

There are many tools that can calculate molecular descriptors but only few are listed below. Several well-known and popular tools are presented in this section, in the sense of not being exhaustive. Some of these tools can be integrated into our DEBRA application as plugins.

Chemistry Development Kit (CDK)[3][20][21] is an open source Java library for structural chemo- and bioinformatics originating in 2000. Many of other open source cheminformatics projects are based on CDK. Canonical SMILES created by CDK do not match Daylight canonical SMILES due to that Daylight treat aromaticity differently. The CDK itself does not contain any graphical user interface (GUI). Its small function portion with GUI simplifying the descriptor generation can be found in tools such as PaDEL, CDK Descriptor Calculator GUI, and BlueDesc.

CDK Descriptor Calculator GUI is an Java-based application, that has GUI and and CLI, can evaluate the fingerprints (hashed, MACCS (MDL), ESTate, CDK standard/extended fingerprints, Pubchem), and the descriptors. It can handle SDF and SMI formats as input.

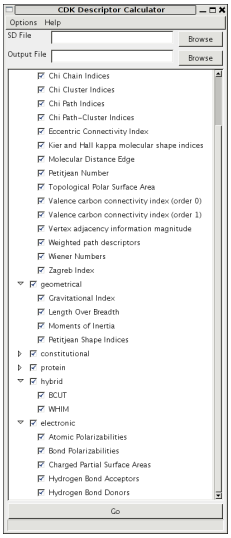


Figure 2.1: CDK Descriptor Calculator GUI

BlueDesc is a Java JAR command line application for easy generation of CDK and JOELib/JOELib2 descriptors (174 alltogether) from SDF file. Fingerprints do not seem to be included.

PaDEL-Descriptor [23] is a Java GUI, a command line, and an open-source software to calculate molecular descriptors and fingerprints from SDF file. *The software currently calculates 905 descriptors (770 1D and 2D descriptors, and 135 3D descriptors) and 10 types of fingerprints*[1]*. The descriptors*

[1]The list is available on `http://padel.nus.edu.sg/software/padeldescriptor/Descriptors.xls`

and fingerprints are calculated using CDK with some additional descriptors and fingerprints. These additions include atom type electrotopological state descriptors, Crippen's logP and MR, extended topochemical atom (ETA) descriptors, McGowan volume, molecular linear free energy relation descriptors, ring counts, count of chemical substructures identified by Laggner, and binary fingerprints and count of chemical substructures identified by Klekota and Roth.

Open Babel [17] is an open-source, collaborative software. It has bindings to Python. The integration with Python can be improved even more by using PyBel. Available fingerprints: MACCS (defined in MACCS.txt), FP2 (indexes linear fragments up to 7 atoms, length = 1024, similar to Daylight fingerprints), FP3 (SMARTS patterns specified in the file patterns.txt), FP4 (SMARTS patterns specified in the file SMARTS_InteLigand.txt). Can generate 3D coordinates (–gen3d option). However, I didn't find any information how this works.

jCompoundMapper in an open source Java library and a command-line tool for chemical fingerprints calculation, published in [13]. It provides popular fingerprinting algorithms for chemical graphs such as depth-first search fingerprints, shortest-path fingerprints, extended connectivity fingerprints, autocorrelation fingerprints (pharmacophore CATS2D fingerprints), radial fingerprints (e.g. Molprint2D), geometrical Molprint, atom pairs, and pharmacophore fingerprints. The inputs are MDL Molfiles.

These tools calculate many descriptors and fingerprints. Our goal is to implement the software that can wrap such tools as plugins. The user can also implement plugins which can call web services instead of command-line, as he wishes.

3. Problem Analysis

In this chapter I analyze the properties which this framework should have. In this thesis and in the source code, I often interchange the terms *plugins*, *tools* and *applications*. The reader should not be confused from the content.

3.1 Requirements

My analysis is based on requirements on a server, a client and plugins. For each requirement I try to propose an idea solving the problem.

3.1.1 Server

There are certain requirements that a server application must have, e.g. the ability of asynchronous serving, a session ID and the ability to run a plugin in its own thread environment.

The request workflow is shown in Figure 3.1.

Requirement 1. The calculation request is initialized by first sending a session ID creation request. This ID should be stored on the client's side and should be injected for requests that make use this ID, such as a situation when a client wants to check the process status or when it loses connection and reconnects with the same ID in order to get output data from the server.

Solution idea. My first attempts to implement the server were very spontaneous. I had very little knowledge about how to implement client-server communication, so I tried the built-in `ServerSocket` and RMI[1] – both seemed to be working well, but I wanted to program on a higher level with some nowadays popular tools based on Java. After a while I thought that it would be great to be able to access the server through HTTP and if someone deploy the server then someone else, even across the pond, could be able to take actions just with URL. And maybe in the future I may develop a web site through which users can make operations and get responses. So I was looking for a different way to implement the server. I hit web services, particularly SOAP messaging, but I did not overcome its complex XML schema output and parsing on the other side. Servlets[2] were presented to me and they got my interest. In defense, servlet can generate a session ID if requested and it can also create an asynchronous environment and send data to clients after a long calculation.

Requirement 2. The server must be able to respond to asynchronous requests, that means the server holds the session ID and when the client asks for the calculated data and the status, the server should be able to send files and the status of how the calculation have been finished or if plugins are still being processed. In case, errors are encountered, the server responds with an appropriate status. The

[1] RMI is remote method invocation is a Java API for calling methods on remote objects.

[2] University course Advanced Java, where RNDr. Petr Hnetynka, Ph.D. (Mr. Hnetynka's website is `http://d3s.mff.cuni.cz/~hnetynka/`) presented about servlets and JavaServer Pages.

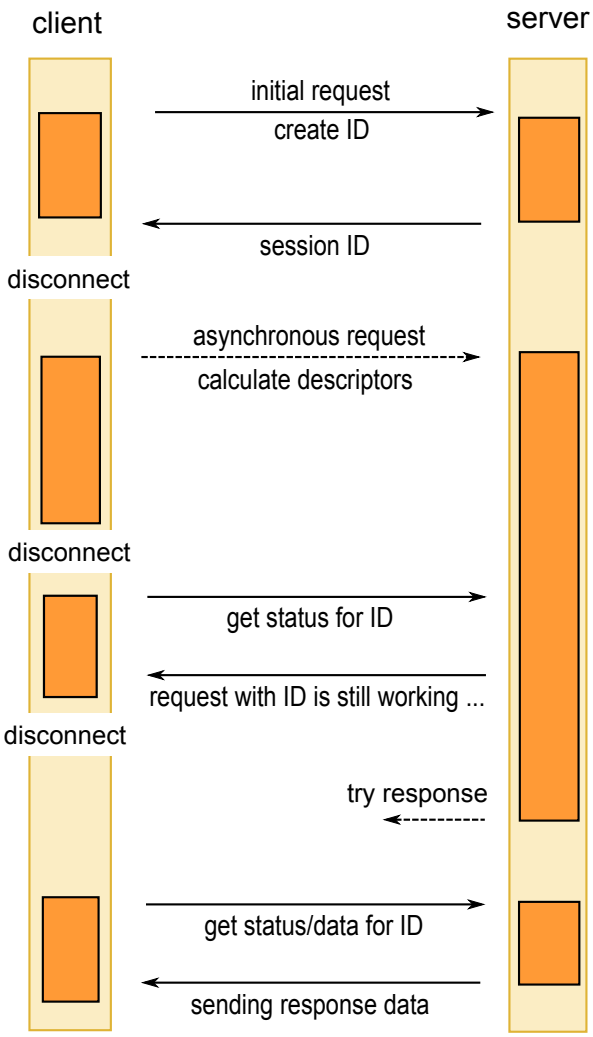


Figure 3.1: Connection workflow

client with a session ID should be able to retrieve data even if it loses connection and connects again with the same ID.

Solution idea. Create an asynchronous request context for each calculation request, put it together with an information of the calculation process, or an error status if a problem is encountered, as a value and the session ID as a key to a map. So, when all plugins notify about the finished task for the request. The output stream is called in order to send data to the client. If the client loses connection, its ID is in a map and it can be retrieved if needed.

Requirement 3. Plugins are Java byte classes that are loaded to the server dynamically by adding them to an appropriate directory. They are removed when the administrator removes the class from the directory and no request that interferes these "to be removed" plugins is working.

Solution idea. The Java classes are dynamically loaded by the class loader. For each get-plugins request we check the specified directory if any new class was added to the directory or removed from the directory. It is legitimate to add plugins very easily. There is no need to worry about adding plugins. On the other hand, removing plugins should be done after all calculations using those

plugins have finished. The problem of removing plugins can be prevented by setting listener for asynchronous context from Requirement 2. The methods of the listener are invoked when the asynchronous request completes, encounters an error, or when it starts. We can make use of these methods to check if any new plugins had been added or some have to be removed.

Requirement 4. Each calculation process works in its own environment as thread. When a calculation is finished an observer is notified and if all calculations for the particular session are finished, the server responds to client with outputs of the calculations.

Solution idea. For each request an asynchronous context is created and stored in a map (as mentioned in Solution idea of Requirement 2). When all tools finish processing, the output stream can be regained from the map.

Requirement 5. The server sends options (descriptor types), their values and a text with description of options on demand.

Solution idea. Each plugin implements an interface and a plugin holder class calls these methods in order to get plugin's structure and this stucture is then transformed into any string format and transmitted to a client. I choose JSON[3] format to represent an object structure.

The plugin's security manager is granted to manage (delete and create) files and lets third party applications take control over them, because some applications (such as BlueDesc – Molecular Descriptor Calculator[4]) which I have tried out, did not allow users to specify the output file name and just create an output file based on the input file name with an additional suffix. For each request the new directory is created initially with input files and then output files are put to the directory.

When the request is fully served, all output files are merged into one file with special delimiters for each file which is then sent to the client. I did not come up with any method how to visually let the user, on client a client side, map input files to output files, with tools take into account. Thanks to that, only one output file is permitted to the yet implemented client. If I could solve this problem, I could easily modify code to send multiple files to the client.

It is very important to keep in mind that the server administrator is responsible for the plugins which are added to enrich server's possibilities for descriptors and fingerprints calculations.

[3]JavaScript Object Notation (or JSON) [5] is a text-based open standard that is language independent. It is a lightweight data-interchange format and it is easy for humans to read and write and easy for machines to parse and generate. Another option is to use the Extensible Markup Language (or XML) format.

[4]The application is available on `http://www.ra.cs.uni-tuebingen.de/software/bluedesc/`

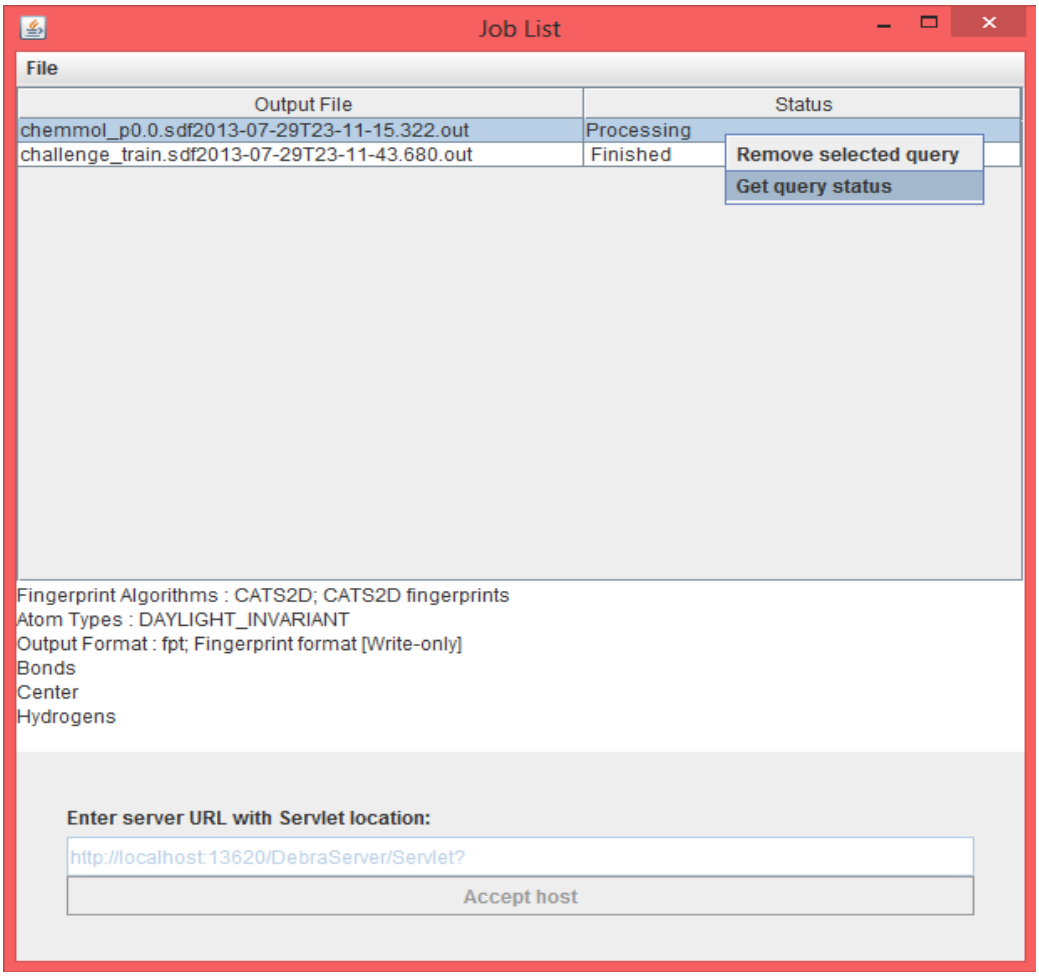


Figure 3.2: Job list view

3.1.2 Client

The client side application displays options of plugins acquired from the server. The user can thereafter select options (descriptors) that he/or she[5] wants to calculate and sends the request to the server.

Requirement 6. Firstly, the client application needs to know the class structure that the server passes to it so that it could parses the format (from Requirement 5) and displays corresponding components which epitomize options of plugins.

Requirement 7. Create a job-list where the queries and theirs statuses are listed. It should enable the user to export the queries to a file, and their import through the a menu of the window. This feature could be handy, when the user knows that the calculation task takes long time, he can quit the application and after a while he turns the application on, imports queries and clicks on the get-status button (Figure 3.2). If the server responds with a fact that all tools had finished, the data are transmitted and the client will display an output.

Requirement 8. The lengthy calculation tasks should not block a user from modifying window's content or performing actions on the buttons.

[5]In the latter paragraphs, I will mention the user in the masculine-form. I hope the reader does not mind, because I do not differentiate the sexes.

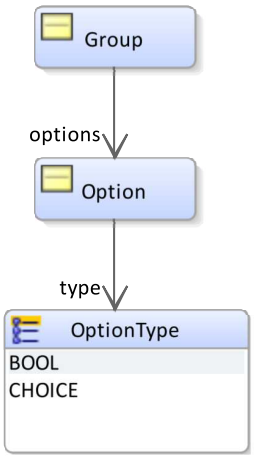


Figure 3.3: Class diagram of plugin options

Solution idea. The Java built-in class `SwingWorker` class can perform time-consuming task which runs in background and does not block *Event Dispatching Thread*[6] from updating views. The user then can run multiple queries which are listed in the job-list window from Requirement 7.

The client can only display what the server transmits. For instance, on each line of output there should be a formula, or SMILE notation of a molecule. This information cannot be provided by the client application, since the client does not know how the input files are treated on the server side. And even the server itself cannot provide such output. This is dependent on each plugin and it is up to plugin to enhance output with formulas, because some third party applications does not enrich computed molecular descriptors with molecular formulas. The plugin developer should add such information by an explicit piece of the code.

3.1.3 Plugins

Some of the requirements we have for plugins were described above and some are listed in this subsection again.

Requirement 9. Plugins invoke an interface of third party applications to run queries.

Requirement 10. Plugins implement a interface, and the methods of the interface are invoked by server application if needed.

Requirement 11. Plugins have a defined structure for representing options. This is invoked via `#addGroup()` and `#addOption()` methods. The former one adds a group which is then represented as a tab and the latter one add an option represented as a check box on the client side. The diagram is shown in Figure 4.11.

[6]Swing event handler runs on the event dispatching thread because most Swing object methods are not "thread safe". It is highly recommended for Swing applications to run on this thread.

Requirement 12. Plugins call a command line interface or any web service.

Solution idea. The plugin-developer is a coordinator of how the plugin acts. There is limitation for the developer, except for some questionable situations.

Requirement 13. Securing the server from being "tormented" by plugin.

Solution idea. Create a class loader for each plugin and set security manager to check suspicious tasks, such as exiting the Java virtual machine, or set the policy file of the web container, i.e. Tomcat in my case.

4. DEBRA Architecture and Implementation

In this chapter, I will present the concrete program implementation in details, its structure, and its class hierarchy. We will also discuss about selected options and decisions that have been made.

I often interchange the terms *plugin*, *application* and *tool*, this should be clear from the context.

The whole application contains two separated projects. The first one is called `DebraServer`, it could be deployed to the server and the other one is called `DebraClient` that sends requests to the server and receives an adequate response.

The Java 1.7 programming language was selected for both projects, due to its interoperability and platform-independence.

4.1 Applied Technologies

Google Gson is a Java library to covert JSON (or JavaScript Object Notation) to Java objects and vice versa [14]. It is used as a data exchange format for the server to "pack" the class object structures, and it requires no annotation in the Java code. The usage is very simple and for the most simplistic scenario it requires only two functions: `Gson#toJson` takes an object and produces the string representing this object in JSON, and `Gson#fromJson` returns an object instance.

Apache Tomcat. At the time of writing the thesis, Apache Tomcat released the version 7.0.27.0 and it was chosen as a web server and a servlet container. Tomcat implements servlet specification from former Sun Microsystem, and yet latest Java™ Servlet 3.0 Specification [19] also contains asynchronous context. Tomcat is a cross-platform application and was developed under open source Apache License 2.0 [7].

Java Servlet is a class that runs in the server in the Java web container and provide a dynamic content as a result of the query and since version 3.0 it can asynchronously respond to a request. Each request includes a state information and a session is managed for each request separately. Servlet itself can communicate over any client-server protocol including the popular HTTP protocol.

Apache HttpClient is a series of libraries that can make HTTP requests and receive responses. Its robustness was a key to my decision. The library is freely accessible from the Apache website.

FileUpload [1] package make it easy to upload files to servlets. It is depended on **Apache Commons IO**[2] which is a library of the input-output utilities.

Swing/AWT is a Java built-in GUI widget toolkit, with which the users can create functional graphical components.

Procedure 3 Servlet handling request

```
1: if ACTION = GETTOOLS then
2:    load plugins from directory to an array and convert to JSON
3: else if ACTION = GETSESSION then
4:    create a session on request object
5: else if ACTION = CALCULATE then
6:    if request content is multipart then
7:       get a session ID
8:       start asynchronous context and put it with session ID to a map
9:       parse multipart content and pass data to PluginHolder
10:      convert JSON query to object
11:      create an observer with session ID and number of plugins to be run as
         its parameters
12:      find appropriate plugins and run query in new thread
13:      if all plugins finished processing then
14:         observer ask SessionToFileMapper map to stream data back to client
15:      end if
16:   end if
17: else if ACTION = GETSTATUS then
18:   look for the status in the map and check if the process finished
19:   if finished then
20:      stream data to client
21:   else
22:      send only the status
23:   end if
24: end if
```

Apache Commons Exec is a library to execute external command line processes from within a Java application. In Chapter 5 we show an example of how the plugins can call third party applications through CLI.

4.2 Server Application

As mentioned above, server is written as servlet that is deployed to a web container such as Apache Tomcat. The request process procedure can be found in Procedure 3. It is a simplification of how the request is treated.

4.2.1 DebraServlet

Now we dive into the `DebraServlet` class implementation and see what happens under the hood and some issues I encountered while developing the DEBRA.

Servlet API is under `javax.servlet, javax.servlet.http` package with the interface `javax.servlet.Servlet`, every servlet must implement this interface. There are basically two possibilities of writing a servlet:

- a generic servlet `javax.servlet.GenericServlet` is protocol-independent

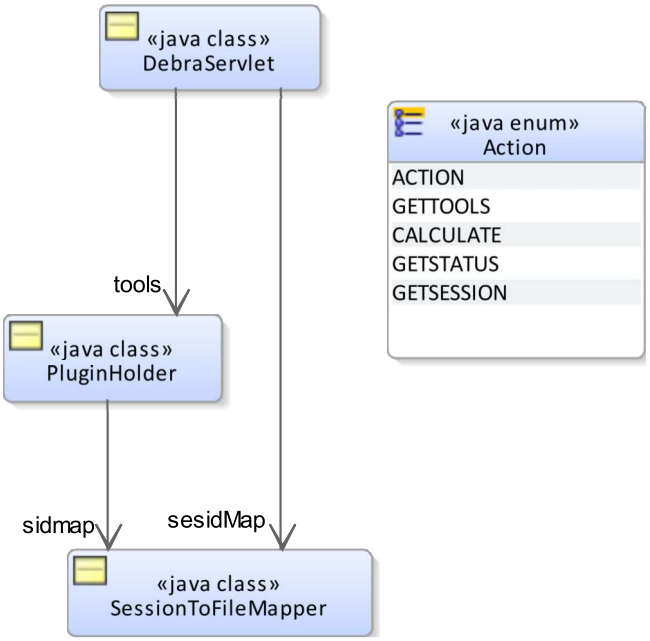


Figure 4.1: Functional Components

- and HTTP servlet `javax.servlet.http.HttpServlet` which is specific for HTTP client-server protocol.

Our `DebraServlet` class on figure 4.2 extends abstract `HttpServlet` class.

All servlets go through, a so called, servlet life-cycle from initialization to being destroyed by the container. Servlet's realization lies in the implementation of the following methods:

1. `init` method initialize servlet and is called only once when the servlet is deployed to the servlet container,
2. for serving the HTTP GET and the HTTP POST requests two methods are provided `doGet` and `doPost`,
3. `destroy` method is invoked only once when the container concludes that all requests for the servlet had been completed, or when the web application was un-deployed.

When `init` method is invoked, the base directory is set up to the WEB-INF/tmp temporary directory. So whenever a request that needs a session and will send data towards the server, the directory with a session ID as its name will be created within WEB-INF/tmp directory. Even though when the plug-in calculates fingerprints it will store the output right into this WEB-INF/tmp/<sessionID>/ directory and after everything for that concrete session is done, the method `getWriter` is called upon `HttpServletResponse` and it will send content to the client.

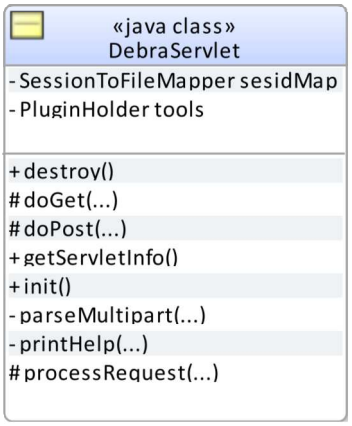


Figure 4.2: DebraServlet Class

When a request is proposed. For our purpose we don't differentiate between the POST and the GET HTTP methods. We serve them the same way by calling a common method `processRequest` with `HttpServletRequest` and `HttpServletResponse` objects as its parameters. We check the request parameters and the parameter "action" is expected. Actions could be one the following:

1. get plugins that `PluginHolder` manages and transform them into JSON format,
2. generate a session ID for the request,
3. receive files and user-selected representations, and perform calculations,
4. get a state of the calculation process.

Before calling a request to calculate, the client should send a request to create a session as it might want to be able retrieve calculated outputs even after it closes the connection and reconnects with the same session ID. The calculation task is a very complex request. Firstly, its content must be multipart that carries files against which calculation is processed, secondly, the asynchronous context is set up and started, this context is stored together with the session ID into a map so when a working computation thread finishes it could regain output writer (or stream) from the asynchronous context object and send data to the client. Servlet 3.0 is also equipped with multipart parser but instead, I use Apache Commons FileUpload library that parses file upload request which conforms RFC 1867[1] to an ordered list of items.

Problem 1. My first experiment was simple, which calls the plugins' `run` method to calculate descriptors. It worked well but blocked other requests until the previous was served, so I tried to create a new thread for each request and with only two assignments:

1. call plugin's `run` method and
2. when it finishes, it notifies the observer with the result.

This approach worked for two or three requests, and for more requests it threw exception while writing to response's output. I figured out, that the response objects had been flushed before the plugins' threads ended. It made me realize that I must start the asynchronous environment for requests. Fortunately, there is support for the asynchronous context since the release of the servlet version 3.0, and all it requires are following lines

```
@WebServlet(asyncSupported=true)
class ...

AsyncContext async = req.startAsync();
async.start( new Runnable{ ... } );
```

Even though it still throws an exception

```
java.lang.IllegalStateException: Not supported.
   at org.apache.catalina.connector.Request.startAsync
```

Somehow Tomcat 7.x has disabled the asynchronous context by default (in my case the asynchronous context was not enabled even by explicit setting neither with the class annotation nor in web.xml) and it must be set up explicitly to `true` by the following code

```
request.setAttribute("org.apache.catalina.ASYNC_SUPPORTED", true);
```

4.2.2 web.xml

In the DebraServer project the directory web/WEB-INF could be found. There is a web.xml file in this directory that defines the servlet mapping, i.e. each servlet could be mapped to a URL and whenever the client accesses this URL the respective servlet will be called.

In Fugure 4.3, I present you web.xml for our servlet. It says that the servlet class `cz.cuni.mff.debra.DebraServlet` is mapped to URL `/Servlet` and the class caters for a client when it enters the URL. Usually servlet container adds an extra project's name before the servlet's URL therefore in my case Tomcat was set up to

`http://localhost:8084/DebraServer/Servlet`

.

Now we describe functional classes on which the server is based on. The diagram is presented in Figure 4.1 on page 23.

4.2.3 PluginHolder

A singleton `PluginHolder` class lies in between servlets and plugins. It uses Google Gson library to convert `Objects` into the JSON string and vice versa. Each time the user asks for plugins, they are loaded from a specific folder. The plugins can be loaded and removed from the directory. The plugins are added as the compiled Java classes to this directory. Additionally, there are three significant inner classes:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
 xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee"
 http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
 <servlet>
    <servlet-name>DebraServlet</servlet-name>
    <servlet-class>
       cz.cuni.mff.debra.DebraServlet
    </servlet-class>
    <async-supported>true</async-supported>
  </servlet>
  <servlet-mapping>
    <servlet-name>DebraServlet</servlet-name>
    <url-pattern>/Servlet</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      2880 <!-- in minutes, i.e. two days -->
    </session-timeout>
  </session-config>
</web-app>
```

Figure 4.3: web.xml

- `CommandRunThread` class which implements the Runnable interface, it's main purpose is to call a plugin's `run` method and to notify the `CalculationDoneNotifier` class when it finishes or fails.

- `CalculationDoneNotifier` class object is created for each request and it holds a session ID for the request. When a thread finishes processing on a plugin, the class is notified via the `done` method or the `failed` method. The former one checks whether all plugins finished processing, in this case it asks to stream out calculated outputs back to the client, the latter one signals when something goes wrong while processing an operation, the error status is stored.

- `AsyncFinishedListener` class implements `AsyncListener` which listens to the asynchronous event whenever it could happen. The listeners' methods are invoked when the asynchronous context is completed or when it starts off or times out or an error occurrence. The first two events can help us dispose the plugins that are no longer of any use. More thoroughly, we have a map with the plugin's name as a key and a number of threads that employ the plugin as its value.

For the calculation action request, the JSON is parsed and the appropriate options are handed to the plugin with the input file paths.

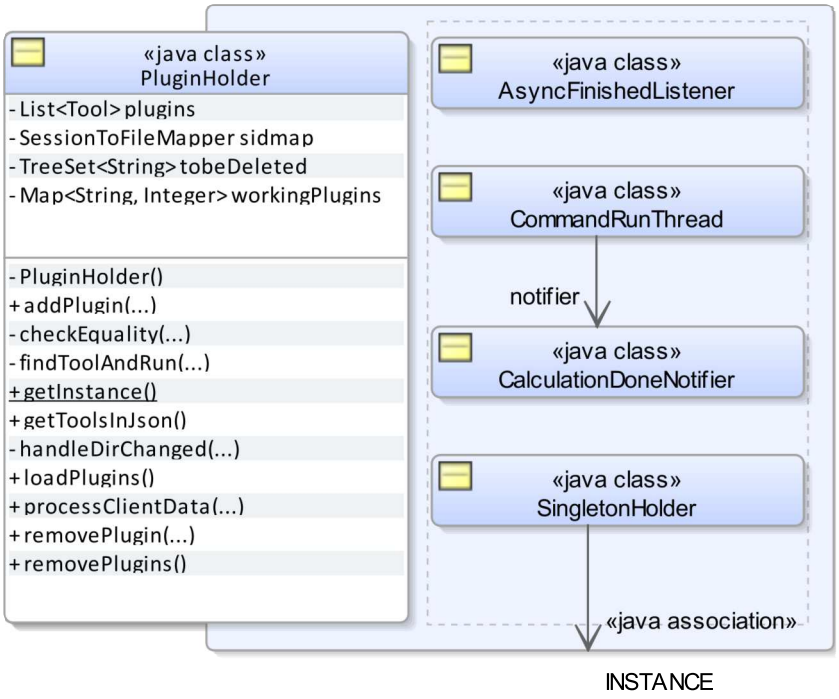


Figure 4.4: PluginHolder Class

4.2.4 SessioToFileMapper

It is a singleton class that maintains the sessions, the asynchronous context, and the files. It can create an empty output file on demand and stream data to the client. It has a map as an attribute, which stores a key – a session ID and a value – an information about the request, i.e. its error states, output files, asynchronous context and whether or not the request was served.

When a servlet is initialized, the base directory for requests' files is set up to WEB-INF/tmp. For each request the subfolder is created and <sessionID> as its name, due to separation of the requests from each others, in this case requests' files will not collide. This subfolder stores all inputs and outputs for a particular request. The folder WEB-INF/tmp which was chosen indeed can be any folder, where we want to store input and output files.

4.3 Plugin

The diagram of the plugin data structure is shown in Figure 4.6. A group of options is represented as tab on the client side and the options as check boxes. The options are indeed fingerprints. The option types are booleans and choices yet. For some applications the text should be entered. This can be easily modified on the server side by adding a value to the enum, and on the client side, the programmer only needs to simply modify the `MainFrameView#initialize()` method.

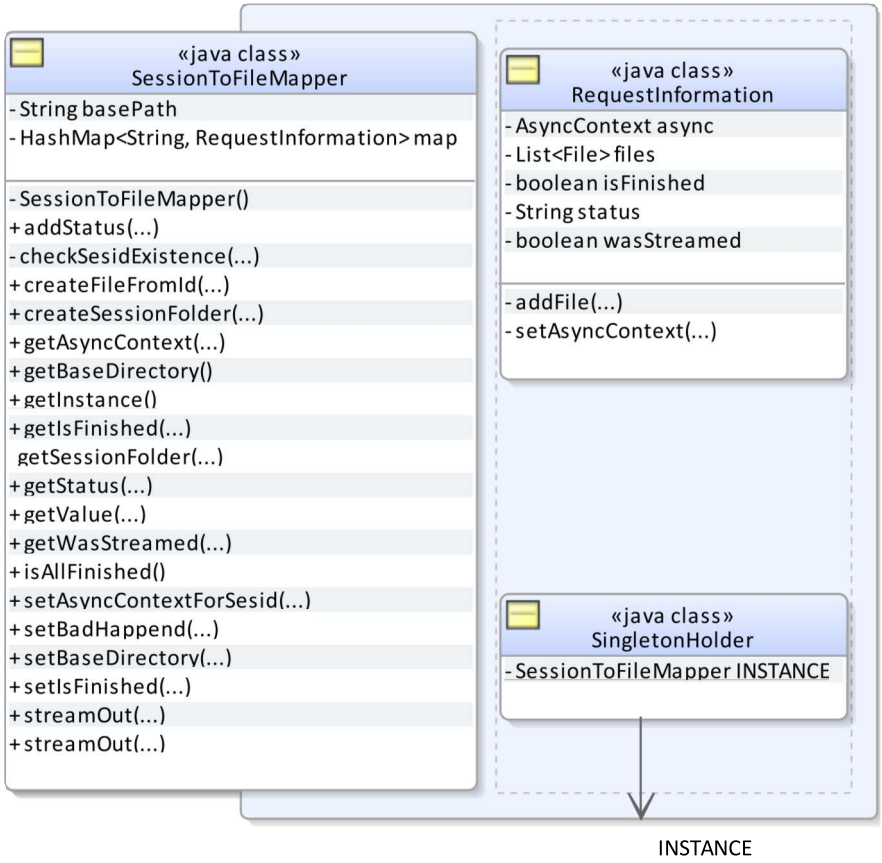


Figure 4.5: SessioToFileMapper Class

The content (attributes) of `BasicTool` class are transformed to JSON and in this form they are transmitted to the client when the get-tools action is invoked. The selected options, flagged by the client, are handed to the plugins to process the query.

All the plugins must implement interface `Tool` and must be put into specified directory. For simplifying plugin-development, the programmer can extend an abstract `AbstractTool` class, as shown in Figure 5.3. And it is up to programmer to initial data and implement `run` method.

Now we turn our focus to the client side application through which we can send queries and see results.

4.4 Client Application

The client application was written to conform the model-view-controller (MVC) pattern. The *model* contains data, the *view* displays the components with the *model* data and the *controller* manipulates the model and makes sense of the user interactions, in the figure 4.7.

There are two models and two views in the application. One pair model-view is for the Job List window and another pair for the main window (in which the user selects options and makes queries. The windows are shown in Figure 4.8.

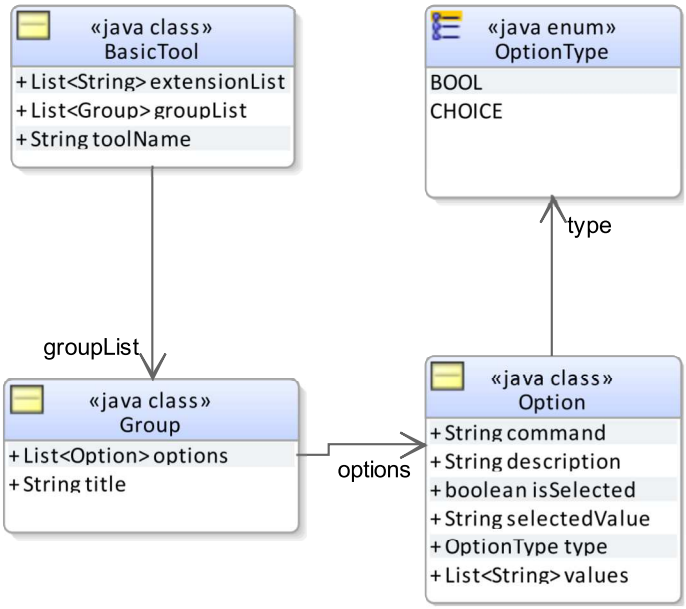


Figure 4.6: The plugin attribute structure

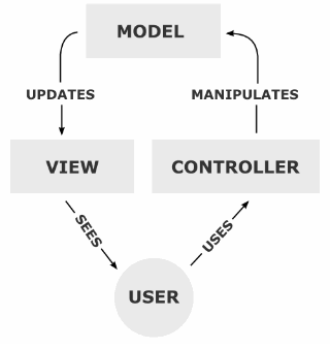


Figure 4.7: Model-View-Controller pattern

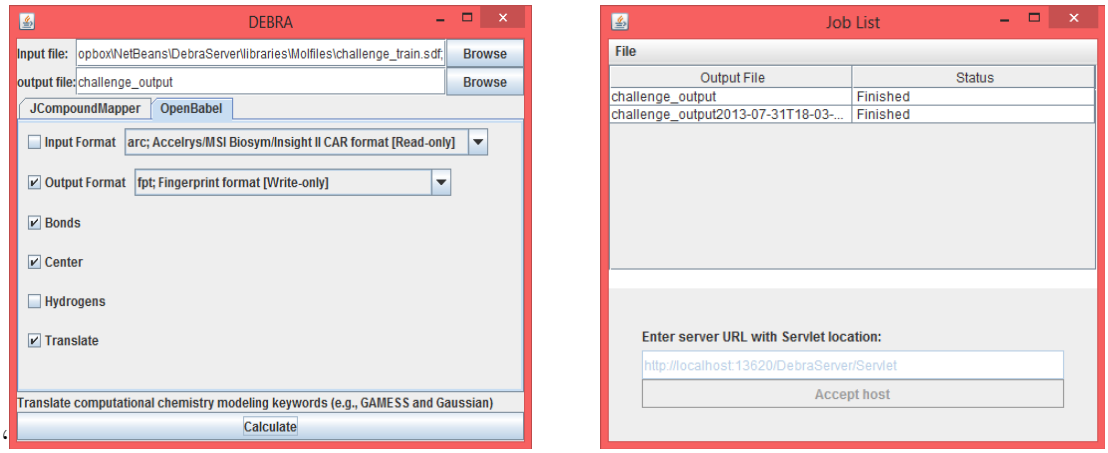


Figure 4.8: Windows: the *main* window (left) and the *Job List* window (right)

4.4.1 Controller

`MainFrameController` class (the diagram is in Figure 4.9) is the controller that manages the models and the views. When the user interacts with buttons or modifies the views, then suitable methods are called. It also mediates HTTP connection. Its logic is presented to the user as sequences of displayed windows. There are two significant classes – `StatusWorker` and `CalculationWorker`. Both of these run lengthy tasks in the background and when they finish, they notify the controller to change the status or display an output.

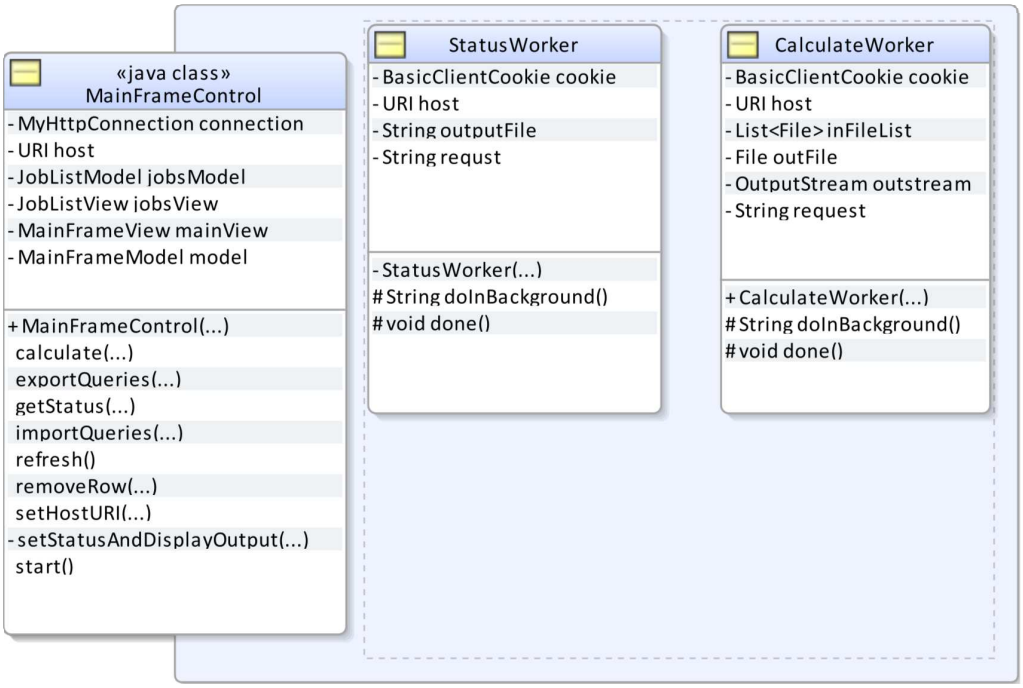


Figure 4.9: MainFrameController class and its inner classes

After the initial connection that checks the existence of the server, the main window is displayed.

4.4.2 Main Window

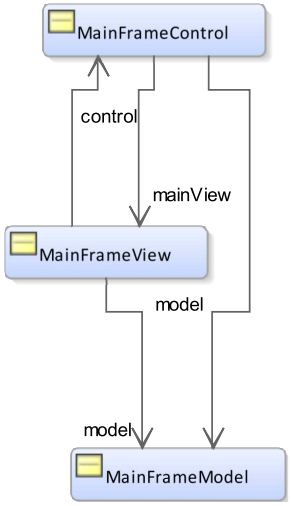


Figure 4.10: MVC diagram, its view is the main window

The middle region of the main window contains the tabs and the check boxes. The tabs are the groups (from Figure 3.3 and 4.8) and the check boxes are the options as shown in Figure 4.11.

The model contains data – list of tools (plugins) and the view displays these data. When the user selects an option, it is flagged and changed in the model. The data are transmitted from the server and transformed from JSON format to the objects representing the data. The object's structure is almost the same as on the server side in Figure 4.6.

Problem 2. Decide the structure of the plugins' options and if it can be as generic as possible, and the client application can parse it and "understand" its structure and being able to depict it to the user.

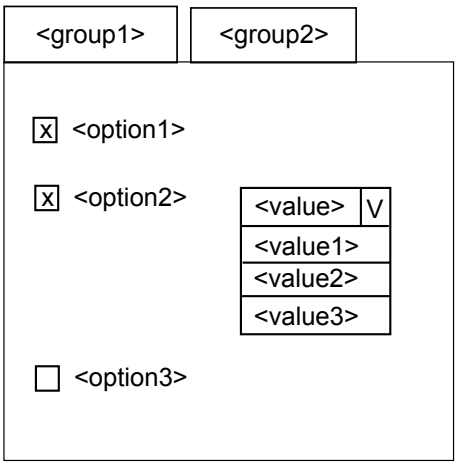


Figure 4.11: The middle region of main window with tabs and check boxes

Solution idea. I did not come up with any generic class structure that could be easily transformed to a string format and parsed. Another idea that I had was to load classes via URL class loader, but the client application still needs to "understand" its structure to be able to display the tabs and the check-boxes. Therefore I hard-coded this, by copying classes from the server application with a very little modification.

4.4.3 Job List Window

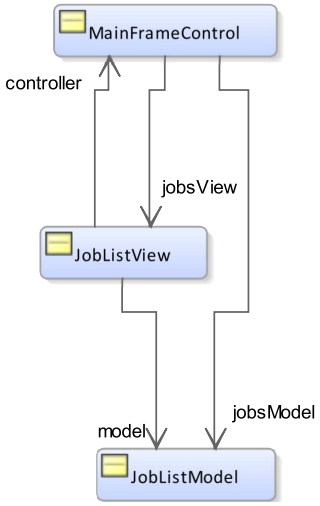


Figure 4.12: The model and the view for Job List window

The diagram of the classes that are responsible for displaying the Job List window is shown in Figure 4.12.

The model for the Job List window is basically only the table model (the table in the middle region of window). The table's rows contain following informations

1. which can reconstruct the connection – i.e. the session ID and the host address,
2. the output filename,
3. the simplified query, and
4. the complex query string in JSON format.

When the user exports the queries, these informations are transformed to JSON and stored in the file.

4.4.4 HTTP Connection

This class contains methods (in Figure 4.13) that can establish the connection. This class is called by controller. The connection can be established by both invoking `URLConnection#openConnection()` and invoking Apache's `HttpClient#execute()`

Figure 4.13: Class diagram of the HttpConnection class

methods. The second one can also make use of a multipart entity in order to send multiple files with just few lines of code.

The detailed documentation is attached as JavaDoc to the enclosed CD.

5. User Documentation

5.1 User Windows

The user interface is described in this chapter. The client capabilities are presented with descriptions together with screen shots. At the end of this chapter, the exemplary plugin implementation is shown.

The client application is packed in JAR file and Java 1.7 version is required for a correct program execution.

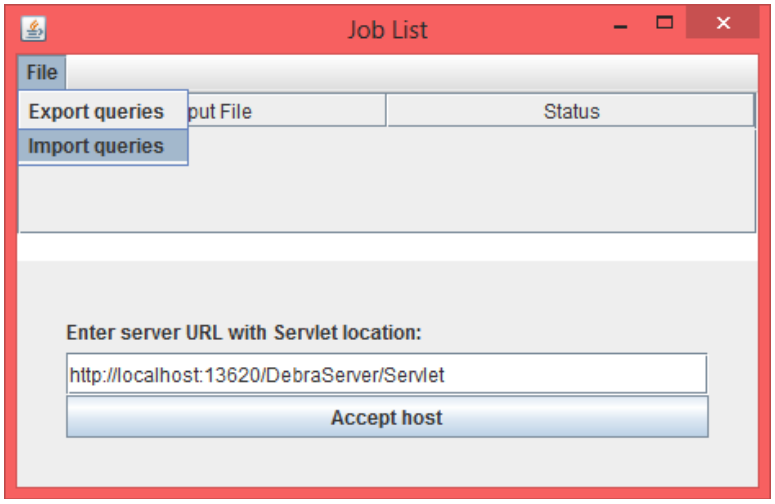


Figure 5.1: The Job list window

After the user launches the program, the windows are shown. First it displays the window Job List, and the bottom region of the window there is a field where the user enters host's address and after the correct input. The Job List window is divided into three regions (Figures 5.1 and 3.2):

1. The top region contains a menu with two items. The first one exports queries to a file. Its output is a file with a text in JSON format that contains information of queries. The second one imports queries from the exported file

2. In the middle region there are a table with queries – each query in a row – columns are the outputs' files name and the statuses of the process, and a selectable text area with a short listing of selected options.

3. On the lower region, there are text fields where the user enters the address of servlet and buttons that makes connection with servlet.

After insertion of the correct host URL, the main window appears, this is divided into three regions:

1. The top region contains a field for entering input file and output file with appropriate buttons for choosing file.

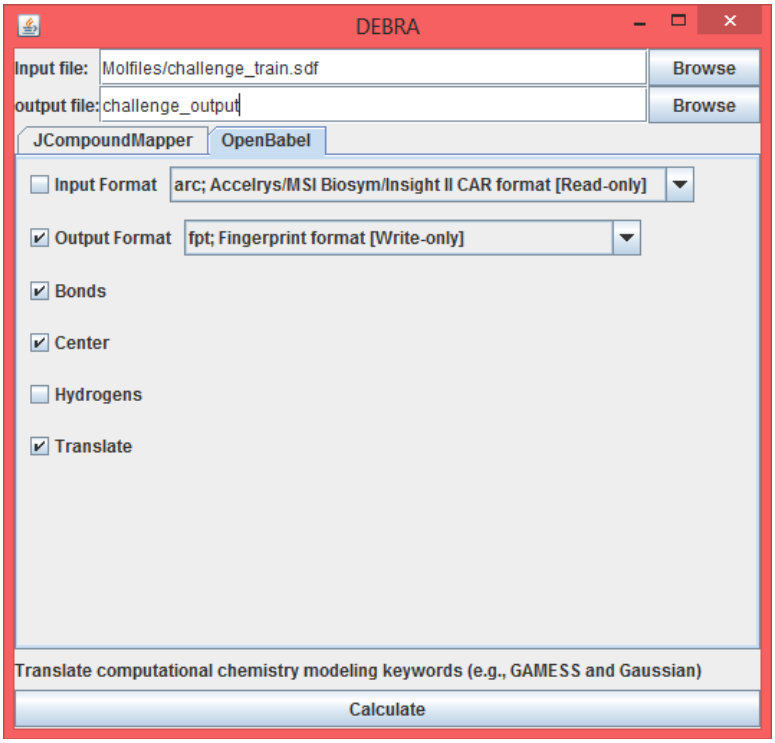


Figure 5.2: The main client window

2. The middle region displays options and tabs representing plugin's options. Options are selected by ticking the check box.

3. In the bottom region, if the user clicks on the calculate button, the request is sent to the server.

The user selects an input file and an output file and ticks options which he wants to be taken into an account and by clicking on the calculate button he submits the query and the query is transmitted to the server. The query is added as an entry to the table as Figure 3.2 shows. The user can check the query status (by right click) or explicitly exports the queries listed in the table by selecting item in the File menu as shown in Figure 5.2. If he closes the window, no information is implicitly stored, that is why he has to explicitly export queries. Next time when he runs the program again, he can import the exported queries and check the status of lengthy tasks. Checking status of the finished task gives the server an impulse to transfer data. If the user closes and re-run the program, and then imports the queries with the lengthy tasks are still in the progress, the data are not transmitted automatically, because the user lost the connection with the server, therefore the user has to explicitly check the statuses.

After the calculation is finished, the server sends data to the client and these data are written into the output file. In case, the user did not specify the output file, the output file name is generated from the input file and the current time is appended. Some errors might occur while the plugin processes the query, these are displayed in the status column.

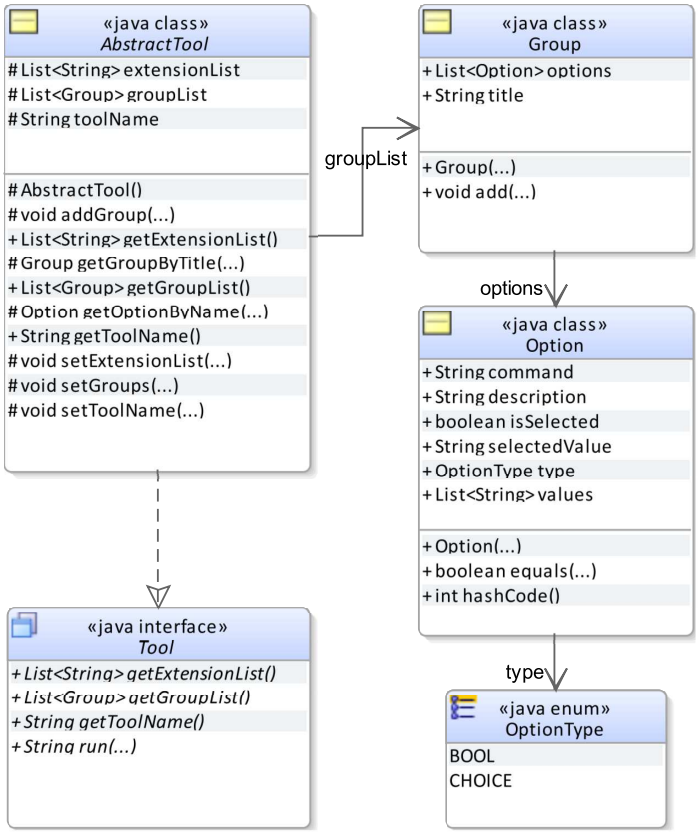


Figure 5.3: Plugin class hierarchy with the pre-implemented abstract class

5.2 Plugins Implementation

A plugin class hierarchy is shown in Figure 5.3. We can invoke methods of the `AbstractTool` class. The plugin-developer just need to implement two methods, one initializes model and second method runs third party applications.

As an example of the plugin-development, I chose the free OpenBabel software. I have mentioned about OpenBabel on the page 14. We will access it's functionality through the CLI.

The Code 5.1 shows important construct of the object initialization. We create the options that user can later select.

1. Firstly, we set the package to `plugins`.
2. Then, we import the `cz.cuni.mff.debra.common` package, it contains the model as the options.
3. We extend the `AbstractTool` class. This class contains some useful functions.
4. Then we set the name for the plugin. Usually by its class name.
5. We create a group and add this group to a list. Actually, we can create several groups, not just only one.
6. We add options to the group. Options that are only of the type `BOOL` has no value, therefore we just specify *null*. For the options of `CHOICE` we must specify a list of the values.

Code 5.1: Initialization the option's structure

```
// OpenBabelTool.java
package plugins;

import cz.cuni.mff.debra.common.*; // contains option's structure
...

public class OpenBabelTool extends AbstractTool
   public OpenBabelTool() {
      setToolName(OpenBabelTool.class.getSimpleName());
      Group OBGroup = new Group("OpenBabel");
      addGroup(OBGroup);

      OBGroup.add(new Option("Bonds", "Convert dative bonds",
          OptionType.BOOL, null));
      OBGroup.add(new Option("Center", "Center atomic coordinates at
          (0,0,0)", OptionType.BOOL, null));
      OBGroup.add(new Option("Hydrogens", "Add Hydrogens",
          OptionType.BOOL, null));
      ...

      List<String> outputFormats = new ArrayList<String>();
      outputFormats.add("fpt; Fingerprint format [Write-only]");
      outputFormats.add("mdl; MDL MOL format");
      ...
      OBGroup.add(new Option("Output Format", formatDescription,
          OptionType.CHOICE, outputFormats));
    }
    ...
}
```

Now we implement the `run` method (Code 5.2). Its return value is a string. String with value "0" stands for that the program does not encountered any error. On the other hand, any other value is treated as the error. This example uses the command line interface of the tool (plugin).

1. In order to be able to use CLI, I use the Apache Commons Exec library [?] to execute the program (tool), therefore it is required to import proper packages.

2. Locate the binary file of the Babel.

3. Find the group that we named in the initialization part.

4. For each option that is selected we add the proper argument to the command line.

Code 5.2: Initialization the option's structure

```
// OpenBabelTool.java
package plugins;

...
import org.apache.commons.exec.CommandLine;
import org.apache.commons.exec.DefaultExecutor;

public class OpenBabelTool extends AbstractTool
    ...

    @Override
    public String run(List<String> ins, String out, List<Group>
        selected) {
      File exeFile = new File("c:/Program Files
          (x86)/OpenBabel-2.3.2/babel.exe");
      CommandLine cmdline = new CommandLine(exeFile.getAbsolutePath());

      Group group = getGroupByTitle("OpenBabel", selected);
      if (group == null) {
         return "Failed to find OpenBabel group";
      }

      Option bondsOption = getOptionByName("Bonds", group.options);
      ...

      if (bondsOption != null && bondsOption.isSelected) {
         cmdline.addArgument("-b");
      }
      ...

      if (inputOption.isSelected) {
         /* We need only "fpt; Fingerprint format [Write-only]" the
            first part and we know that the returns in this form. */
         String opt = inputOption.selectedValue.split(";", 2)[0];
         cmdline.addArgument("-i");
         cmdline.addArgument(opt);
      }
      cmdline.addArgument(inputFilePaths.get(0));
```

On the following lines the command line program is executed, and in the case of no erroneous. The "0" is returned. In this case, plugin only runs first file as an input.

Code 5.3: The run

```
// OpenBabelTool.java
...

      DefaultExecutor executor = new DefaultExecutor();
      try {
          executor.execute(cmdline);
```

```
        } catch (IOException ex) {
            return "Execution error";
        }
        return "0";
    }
}
```

The entire example can be found in the Appendix B and also in the enclosed CD. The content of the CD is listed in the Appendix A.

Conclusion

In this thesis, the DEBRA Descriptor Barn framework was developed. It consists of the server part and the client part. The server makes use of the third-party applications and executes them through their interfaces, such as a command line interface, or through the web services. These functionalities are integrated to the server as plugins.

I used the fact that quite a lot of applications calculate representations of the molecules. In addition, I do not know about any of them to be freely available, open-source, and can integrate other applications very simply as plugins.

A client application displays options, which the server provides. The user then can choose among these representations and send the calculation requests to the server together with the initial representation of the molecules. The client can store these requests for later usage.

I managed to implement all functionality which I stated in the introduction. But still, the framework has some flaws that are in my list of future improvements. Such as,

- The user can specify only one file as an output.
- Display the outputs in a reasonable form.
- There is no limitation for the plugins leading to security issues.
- The client-server communication is not secured.
- It misses the pipeline of the plugins – OpenBabel can be used to transform between formats and its result can be passed to another plugin.
- The files are not compressed, therefore it could lead to network congestion.
- Now it is not convenient for a user to connect to multiple servers. The bypass for him, now, is to make the requests on the server, store query, refresh the application, connect to another server and import the query.

Thanks to this thesis and my supervisor, I find out that cheminformatics is very interesting field of research and I want to study more about this field, more thoroughly, in order to create an applications that can make use of the DEBRA framework and being able to predict the activity of compounds from their structures. Unfortunately, this exceeds my ability, now. I think it is worth to investigate this latter, in my diploma thesis.

Bibliography

[1] Apache commons fileupload. [online]. http://commons.apache.org/proper/commons-fileupload/. Accessed: 2013-27-07.

[2] Apache commons io library. [online]. http://commons.apache.org/proper/commons-io/index.html. Accessed: 2013-27-07.

[3] Chemistry development kit (cdk). [online]. http://cdk.sourceforge.net. Accessed: 2013-27-07.

[4] Daylight theory manual. [online]. http://www.daylight.com/dayhtml/doc/theory/index.html. Accessed: 2013-27-07.

[5] Javascript object notion (JSON). [online]. http://json.org/. Accessed: 2013-27-07.

[6] Marvin, a collection of tools to draw and visualize chemistry. [online]. http://www.chemaxon.com/products/marvin/.

[7] Apache license 2.0. [online]. http://www.apache.org/licenses/LICENSE-2.0, 2004. Accessed: 2013-27-07.

[8] Bolton, E. E., Wang, Y., Thiessen, P. A., and Bryant, S. H. Chapter 12 pubchem: Integrated platform of small molecules and biological activities. *Annual Reports in Computational Chemistry 4* (2008), 217 – 241.

[9] Brown, F. K. Chapter 35. chemoinformatics: What is it and how does it impact drug discovery. In *Annual Reports in Medicinal Chemistry*, J. A. Bristol, Ed., vol. 33. Academic Press, 1998, pp. 375 – 384.

[10] Durant, J. L., Leland, B. A., Henry, D. R., and Nourse, J. G. Reoptimization of MDL keys for use in drug discovery. *J Chem Inf Comput Sci 42*, 6 (2002), 1273–1280.

[11] Faulon, J.-L., and Bender, A. *Handbook of chemoinformatics algorithms*. Chapman & Hall/CRC, 2010.

[12] Gaulton, A., Bellis, L. J., Bento, A. P., Chambers, J., Davies, M., Hersey, A., Light, Y., McGlinchey, S., Michalovich, D., Al-Lazikani, B., and Overington, J. P. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic Acids Research 40*, D1 (2012), D1100–D1107.

[13] Georg, H., Lars, R., Andreas, J., Nikolas, F., and Andreas, Z. jCompoundMapper: An open source java library and command-line tool for chemical fingerprints. [online]. http://jcompoundmapper.sourceforge.net/. Accessed: 2013-27-07.

[14] Inderjeet, S., Joel, L., and Jesse, W. Googlegson library. [online]. https://code.google.com/p/google-gson/. Accessed: 2013-27-07.

[15] Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. Zinc: A free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling 52*, 7 (2012), 1757–1768.

[16] Johnson, M. *Concepts and applications of molecular similarity*. Wiley, New York, 1990.

[17] O'Boyle, N., Banck, M., James, C., Morley, C., Vandermeersch, T., and Hutchison, G. Open babel: An open chemical toolbox. *Journal of Cheminformatics 3*, 1 (Oct. 2011), 33.

[18] Petr, S., and David, H. Chemical space visualization using viframe. [online]. `http://siret.ms.mff.cuni.cz/node/1064`. Accessed: 2013-27-07.

[19] Rajiv, M., and Amy, R. Servlet 3.0 specification. [online]. `http://jcp.org/en/jsr/detail?id=315`. Accessed: 2013-27-07.

[20] Steinbeck, C., Han, Y., Kuhn, S., Horlacher, O., Luttmann, E., and Willighagen, E. The chemistry development kit (cdk). *Journal of Chemical Information and Modeling vol. 43*, issue 2 (2003-03-24), 493–500.

[21] Steinbeck, C., Hoppe, C., Kuhn, S., Floris, M., Guha, R., and Willighagen, E. Recent developments of the chemistry development kit (cdk) - an open-source java library for chemo- and bioinformatics. *Current Pharmaceutical Design vol. 12*, issue 17 (2006-06-01), 2111–2120.

[22] Weininger, D. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Modeling vol. 28*, issue 1 (1988-02-01), 31–36.

[23] Yap, C. W., Hoppe, C., Kuhn, S., Floris, M., Guha, R., and Willighagen, E. Padel-descriptor. *Journal of Computational Chemistry vol. 32*, issue 7 (2011), 1466–1474.

List of Tables

List of Figures

List of Abbreviations

API	Application Programming Interface
CDK	Chemistry Development Kit
CLI	Command Line Interface
CTAB	Connection Table
DEBRA	Descriptor Barn framework
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
InChI	International Chemical Identifier
JAR	Java Archive
JSON	JavaScript Object Notation
MACCS	Molecular Access System
MOL	MDL Molfile
MOL	MDL Molfile
MVC	Model View Controller pattern
RMI	Remote Method Invocation
SDF	Structure Data File
SMILES	Simplified Molecular Input Line Entry System
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAR	Web Archive
WLN	Wiswesser Line Notation
XML	Extensible Markup Language

A. Content of CD

The enclosed CD contains the documentation, the source codes, the program binaries, the external libraries, the exemplary molecules' files, the exemplary plugin implementation and the electronic version of this thesis. The CD is organized as follows:

DebraClient and **DebraServer** contain the source codes, binaries, and JavaDocs. The binaries and JavaDocs are under dist/ folder. The DebraClient.jar is an executable program, which requires additional libraries under lib/ folder. Remember to read the README.txt file. The DebraServlet.war is the web archive to be deployed into web container.

Libraries is the folder that contains used libraries.

Molfiles is the folder that constains the exemplary files with molecules.

JCompounderMapperTool.class and **OpenBabelTool.class** are the compiled Java classes which can be add to servers plugins' directory.

OpenBabel2.3.2a_Windows_Installer.exe and **jCMapperCLI.jar** are the third party applications, and they can be integrated as the plugins to the server application.

JCompounderMapperTool.java and **OpenBabelTool.java** are the exemplary code of the plugins, adapters to CLI.

Installation Guide.pdf is an informal guide that shows the installation of the Tomcat step by step.

B. Example of Plugin Code

The listed example can be compiled by following command

```
javac -cp commons-exec-1.1.jar;WEB-INF/classes/ OpenBabelTool.java
```

The first part of the class-path is the path to the JAR file that contains the Apache Commons Exec library, and the second is a path to the compiled classes of the DebraServer project that is available after deployment in the folder $CATALINA_HOME$/webapps/DebraServer/

```
// OpenBabelTool.java
package plugins;

import cz.cuni.mff.debra.common.*;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.commons.exec.CommandLine;
import org.apache.commons.exec.DefaultExecutor;

public class OpenBabelTool extends AbstractTool {

    public OpenBabelTool() {
        setToolName(OpenBabelTool.class.getSimpleName());

        Group OBGroup = new Group("OpenBabel");

        List<String> inputFormats = new ArrayList<String>();
        inputFormats.add("arc; Accelrys/MSI Biosym/Insight II CAR
            format [Read-only]");
        inputFormats.add("bgf; MSI BGF format");
        inputFormats.add("box; Dock 3.5 Box format");
        inputFormats.add("bs; Ball and Stick format");
        inputFormats.add("c3d1; Chem3D Cartesian 1 format");
        inputFormats.add("c3d2; Chem3D Cartesian 2 format");
        inputFormats.add("caccrt; Cacao Cartesian format");
        inputFormats.add("can; Canonical SMILES format");
        inputFormats.add("car; Accelrys/MSI Biosym/Insight II CAR
            format [Read-only]");
        inputFormats.add("ccc; CCC format [Read-only]");
        inputFormats.add("cdx; ChemDraw binary format [Read-only]");
        inputFormats.add("cdxml; ChemDraw CDXML format");
        inputFormats.add("cif; Crystallographic Information File");
        inputFormats.add("cml; Chemical Markup Language");
        inputFormats.add("cmlr; CML Reaction format");
```

```
        inputFormats.add("crk2d; Chemical Resource Kit 2D diagram
            format");
        inputFormats.add("crk3d; Chemical Resource Kit 3D format");
        inputFormats.add("ct; ChemDraw Connection Table format");
        inputFormats.add("dmol; DMol3 coordinates format");
        inputFormats.add("ent; Protein Data Bank format");
        inputFormats.add("fch; Gaussian formatted checkpoint file
            format [Read-only]");
        inputFormats.add("fchk; Gaussian formatted checkpoint file
            format [Read-only]");
        inputFormats.add("fck; Gaussian formatted checkpoint file
            format [Read-only]");
        inputFormats.add("feat; Feature format");
        inputFormats.add("fract; Free Form Fractional format");
        inputFormats.add("fs; Open Babel FastSearching database");
        inputFormats.add("g03; Gaussian 98/03 Output [Read-only]");
        inputFormats.add("g98; Gaussian 98/03 Output [Read-only]");
        inputFormats.add("gam; GAMESS Output [Read-only]");
        inputFormats.add("gamout; GAMESS Output [Read-only]");
        inputFormats.add("gpr; Ghemical format");
        inputFormats.add("hin; HyperChem HIN format");
        inputFormats.add("ins; ShelX format [Read-only]");
        inputFormats.add("jout; Jaguar output format [Read-only]");
        inputFormats.add("mdl; MDL MOL format");
        inputFormats.add("mmd; MacroModel format");
        inputFormats.add("mmod; MacroModel format");
        inputFormats.add("mol; MDL MOL format");
        inputFormats.add("mol2; Sybyl Mol2 format");
        inputFormats.add("moo; MOPAC Output format [Read-only]");
        inputFormats.add("mop; MOPAC Cartesian format");
        inputFormats.add("mopcrt; MOPAC Cartesian format");
        inputFormats.add("mopin; MOPAC Internal");
        inputFormats.add("mopout; MOPAC Output format [Read-only]");
        inputFormats.add("mpc; MOPAC Cartesian format");
        inputFormats.add("mpqc; MPQC output format [Read-only]");
        inputFormats.add("nwo; NWChem output format [Read-only]");
        inputFormats.add("pc; PubChem format [Read-only]");
        inputFormats.add("pcm; PCModel format");
        inputFormats.add("pdb; Protein Data Bank format");
        inputFormats.add("pqs; Parallel Quantum Solutions format");
        inputFormats.add("prep; Amber Prep format [Read-only]");
        inputFormats.add("qcout; Q-Chem output format [Read-only]");
        inputFormats.add("res; ShelX format [Read-only]");
        inputFormats.add("rxn; MDL RXN format");
        inputFormats.add("sd; MDL MOL format");
        inputFormats.add("sdf; MDL MOL format");
        inputFormats.add("smi; SMILES format");
        inputFormats.add("sy2; Sybyl Mol2 format");
        inputFormats.add("tdd; Thermo format");
        inputFormats.add("therm; Thermo format");
        inputFormats.add("tmol; TurboMole Coordinate format");
```

```
        inputFormats.add("unixyz; UniChem XYZ format");
        inputFormats.add("vmol; ViewMol format");
        inputFormats.add("xml; General XML format [Read-only]");
        inputFormats.add("xyz; XYZ cartesian coordinates format");
        inputFormats.add("yob; YASARA.org YOB format");

        String formatDescription = "This should be explicitly selected";
        OBGroup.add(new Option("Input Format", formatDescription,
            OptionType.CHOICE, inputFormats));

        List<String> outputFormats = new ArrayList<String>();
        outputFormats.add("acr; Carine ASCI Crystal");
        outputFormats.add("alc; Alchemy format");
        outputFormats.add("bgf; MSI BGF format");
        outputFormats.add("box; Dock 3.5 Box format");
        outputFormats.add("bs; Ball and Stick format");
        outputFormats.add("c3d1; Chem3D Cartesian 1 format");
        outputFormats.add("c3d2; Chem3D Cartesian 2 format");
        outputFormats.add("caccrt; Cacao Cartesian format");
        outputFormats.add("cache; CAChe MolStruct format [Write-only]");
        outputFormats.add("cacint; Cacao Internal format [Write-only]");
        outputFormats.add("can; Canonical SMILES format");
        outputFormats.add("cdxml; ChemDraw CDXML format");
        outputFormats.add("cht; Chemtool format [Write-only]");
        outputFormats.add("cif; Crystallographic Information File");
        outputFormats.add("cml; Chemical Markup Language");
        outputFormats.add("cmlr; CML Reaction format");
        outputFormats.add("com; Gaussian 98/03 Cartesian output
            [Write-only]");
        outputFormats.add("copy; Copies raw text [Write-only]");
        outputFormats.add("crk2d; Chemical Resource Kit 2D diagram
            format");
        outputFormats.add("crk3d; Chemical Resource Kit 3D format");
        outputFormats.add("csr; Accelrys/MSI Quanta CSR format
            [Write-only]");
        outputFormats.add("cssr; CSD CSSR format [Write-only]");
        outputFormats.add("ct; ChemDraw Connection Table format");
        outputFormats.add("dmol; DMol3 coordinates format");
        outputFormats.add("ent; Protein Data Bank format");
        outputFormats.add("fa; FASTA format [Write-only]");
        outputFormats.add("fasta; FASTA format [Write-only]");
        outputFormats.add("feat; Feature format");
        outputFormats.add("fh; Fenske-Hall Z-Matrix format
            [Write-only]");
        outputFormats.add("fix; SMILES FIX format [Write-only]");
        outputFormats.add("fpt; Fingerprint format [Write-only]");
        outputFormats.add("fract; Free Form Fractional format");
        outputFormats.add("fs; Open Babel FastSearching database");
        outputFormats.add("fsa; FASTA format [Write-only]");
        outputFormats.add("gamin; GAMESS output [Write-only]");
        outputFormats.add("gamout; GAMESS Output [Read-only]");
```

```
        outputFormats.add("gau; Gaussian 98/03 Cartesian output
            [Write-only]");
        outputFormats.add("gjc; Gaussian 98/03 Cartesian output
            [Write-only]");
        outputFormats.add("gjf; Gaussian 98/03 Cartesian output
            [Write-only]");
        outputFormats.add("gpr; Ghemical format");
        outputFormats.add("gr96; GROMOS96 format [Write-only]");
        outputFormats.add("hin; HyperChem HIN format");
        outputFormats.add("inchi; IUPAC InChI [Write-only]");
        outputFormats.add("inp; GAMESS output [Write-only]");
        outputFormats.add("ins; ShelX format [Read-only]");
        outputFormats.add("jin; Jaguar output format [Write-only]");
        outputFormats.add("mdl; MDL MOL format");
        outputFormats.add("mmd; MacroModel format");
        outputFormats.add("mmod; MacroModel format");
        outputFormats.add("mol; MDL MOL format");
        outputFormats.add("mol2; Sybyl Mol2 format");
        outputFormats.add("molreport; Open Babel molecule report
            [Write-only]");
        outputFormats.add("mop; MOPAC Cartesian format");
        outputFormats.add("mopcrt; MOPAC Cartesian format");
        outputFormats.add("mopin; MOPAC Internal");
        outputFormats.add("mpc; MOPAC Cartesian format");
        outputFormats.add("mpd; Sybyl descriptor format [Write-only]");
        outputFormats.add("mpqcin; MPQC simplified output format
            [Write-only]");
        outputFormats.add("nw; NWChem output format [Write-only]");
        outputFormats.add("pcm; PCModel format");
        outputFormats.add("pdb; Protein Data Bank format");
        outputFormats.add("pov; POV-Ray output format [Write-only]");
        outputFormats.add("pqs; Parallel Quantum Solutions format");
        outputFormats.add("qcin; Q-Chem output format [Write-only]");
        outputFormats.add("report; Open Babel report format
            [Write-only]");
        outputFormats.add("rxn; MDL RXN format");
        outputFormats.add("sd; MDL MOL format");
        outputFormats.add("sdf; MDL MOL format");
        outputFormats.add("smi; SMILES format");
        outputFormats.add("sy2; Sybyl Mol2 format");
        outputFormats.add("tdd; Thermo format");
        outputFormats.add("test; Test format [Write-only]");
        outputFormats.add("therm; Thermo format");
        outputFormats.add("tmol; TurboMole Coordinate format");
        outputFormats.add("txyz; Tinker MM2 format [Write-only]");
        outputFormats.add("unixyz; UniChem XYZ format");
        outputFormats.add("vmol; ViewMol format");
        outputFormats.add("xed; XED format [Write-only]");
        outputFormats.add("xyz; XYZ cartesian coordinates format");
        outputFormats.add("yob; YASARA.org YOB format");
        outputFormats.add("zin; ZINDO output format [Write-only]");
```

```
        OBGroup.add(new Option("Output Format", formatDescription,
            OptionType.CHOICE, outputFormats));

        OBGroup.add(new Option("Bonds", "Convert dative bonds",
            OptionType.BOOL, null));
        OBGroup.add(new Option("Center", "Center atomic coordinates at
            (0,0,0)", OptionType.BOOL, null));
        OBGroup.add(new Option("Hydrogens", "Add Hydrogens",
            OptionType.BOOL, null));
        OBGroup.add(new Option("Translate", "Translate computational
            chemistry modeling keywords (e.g., GAMESS and Gaussian)",
            OptionType.BOOL, null));

        addGroup(OBGroup);
    }

    @Override
    public String run(List<String> inputFilePaths, String
        outputFilePath, List<Group> selected) {
        System.out.println("inputFilePath = " + inputFilePaths);
        System.out.println("outputFilePath = " + outputFilePath);
        //new File(outputFilePath).mkdirs();


        File exeFile = new File("c:/Program Files
            (x86)/OpenBabel-2.3.2/babel.exe");
        CommandLine cmdline = new
            CommandLine(exeFile.getAbsolutePath());

        Group group = getGroupByTitle("OpenBabel", selected);
        if (group == null) {
            return "Failed to find OpenBabel group";
        }

        Option bondsOption = getOptionByName("Bonds", group.options);
        Option centerOption = getOptionByName("Center", group.options);
        Option hydroOption = getOptionByName("Hydrogens",
            group.options);
        Option transOption = getOptionByName("Translate",
            group.options);
        Option inputOption = getOptionByName("Input Format",
            group.options);
        Option outputOption = getOptionByName("Output Format",
            group.options);

        if (bondsOption != null && bondsOption.isSelected) {
            cmdline.addArgument("-b");
        }

        if (centerOption != null && centerOption.isSelected) {
```

```
            cmdline.addArgument("-c");
        }

        if (hydroOption != null && hydroOption.isSelected) {
            cmdline.addArgument("-h");
        }

        if (transOption != null && transOption.isSelected) {
            cmdline.addArgument("-k");
        }

        if (inputOption == null) {
            return "Failed to find Input Format options";
        }
        if (inputOption.isSelected) {
            String opt = inputOption.selectedValue.split(";", 2)[0];
            cmdline.addArgument("-i");
            cmdline.addArgument(opt);
        }
        cmdline.addArgument(inputFilePaths.get(0));

        if (outputOption == null) {
            return "Failed to find ouput format option";
        }
        if (outputOption.isSelected) {
            String opt = outputOption.selectedValue.split(";", 2)[0];
            cmdline.addArgument("-o");
            cmdline.addArgument(opt);
        }
        cmdline.addArgument(outputFilePath);

        DefaultExecutor executor = new DefaultExecutor();
        System.out.println("Executing cmdline = " + cmdline.toString());
        try {
            System.out.println("Executor = " +
                executor.execute(cmdline));
            System.err.println("Good, output file is in " +
                outputFilePath);
        } catch (IOException ex) {
            return "Execution error";
        }
        return "0";
    }
}
```