Internship Report

Institut Universitaire de Technologie

Development of a GUI for a SAXS data analysis program for modelling proteins in solution

Alejandro de Maria Antolinos

December 1, 2009

Contents

Chapter 1

About the internship

This project has been carried out in the context of a work placement agreement between ESRF [1] and the Universit Joseph Fourier.[2] Concerning to the university I was enrolled in the final year of professional bachelor degree in computer networks and telecomunications specialising in wireless networks and and security.

As french student of last year I must do a work placement in a profesional area. I took this opportunity to do my final project. Doing a work, or project not only challenges in the fact of living in other country, other languages etc... but the real challenge is the cross-communication between people of different countries working with the same aim. Cultural gap between all us can create difficult situations and only open-minded people could solved this situations.

The ESRF not only gave me the opportunity to develop myself in my professional area but as well as in a personal area. I worked for the Experimental Division who is in charge of supporting scientifics who are doing their experiments inside of the beamlines.

Besides of the opportunity of working with people of more than 20 different nacionalities creating a special environment where the fact of working seems different.

[1] European Synchroton Radiation Facility

[2] Specially,A University Institute of Technology

Chapter 2

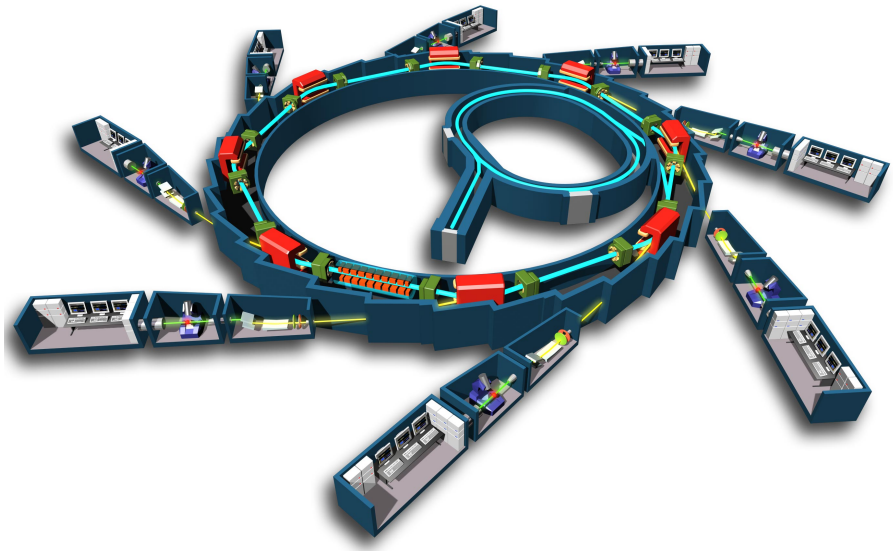
Introduction

2.1 Understanding ESRF

ESRF is a company created to carry out advanced scientific research. It represents a very real technological, scientific and human challenge. Budget estimated for 2007 year was around 80 millions. With 600 people of staff members and around 6000 researchers visitors coming each year. They create about 1500 scientific papers per year.

It is situated at Grenoble, a small city at the foot of the French Alps where the Drac joins the Isere River. Population is estimated about 560 thousand inhabitants. After Paris is the second city with more amount of technological enterprises of France

"A synchrotron is a particular type of cyclic particle accelerator in which the magnetic field (to turn the particles so they circulate) and the electric field (to accelerate the particles) are carefully synchronized with the travelling particle beam."[1]



Scientifics worldwide come to ESRF for executing their experiments. The most of the times they have developped themselves their software tools to carry out their purposes. Thus, a wide varity of sofware can be found developped using languages as Fortran, c, c++, python, java, visual basic, etc...

[1] source: www.wikipedia.com

Sometimes this software has not got appropiate manuals and Graphic User Intarface. In fact, most of them haven't got a Graphic User Interface and people inside of the experimental Division is in charge to create an appropiate GUI.

The wide variety of software has favoured the utilisation of script languages to create GUI's as, for example, python. Combining Python with some linux shell scripting is posible to give a solution to the major part of the software that require an user interface.

Of course, a cross-platform environment is required. We can not know which operating system the final user will use. Actually they are working in their unirversities and for a few weeks they come to execute their experiment. So, a cross-platform environment is expected and our programs need to be portables.

Concerning to the scientific software the most of the them need a very high calculation level. Experiments estimating the weight of the proteines is just an example where millions of mathematical operations will be carried out. Likeliwise not only we will need a good processor but a good compiler able to run fast applications and carry out millions of mathematical operations per second.

2.2 Fortran Language

Fortran is a language created by IBM in the 1950s for scientific and engineering applications. We could say that Fortran is the grandfather of all scientific computer languages. The name Fortran is derived from FORmula and TRANslation, indicating that the language was intended to translate scientific equation into computer code.

It is an imperative programming language that is especially suited to numeric computation and scientific computing. We can remark as well as it is one of the most important languages used in the area of high-performance computing and is the language used for programs that benchmark and rank the world's fastest supercomputers.

2.2.1 Portability

Portability was a big problem. Software companies sold incompatibles extensions in order to differenciate their products. In addition, any fortran code standard was published until 1977 when the National Bureau of Standards published FIPS PUB 69, that processors purchased by the U.S. Government were required to diagnose extensions of the standard. Rather than offer two processors, essentially every compiler eventually had at least an option to diagnose extensions.

Similarily for numerical, it is important to take account of the characteristics of the arithmetic.

2.2.2 The evolution of Fortran

Fortran is a dynamic language that is constantly evolving to keep up on advances in programming practice. Fortran Working Group belongings to ISO has the responsibility for developing new versions.

Preparing a new version is a huge job starting by asking for suggestions to the users for new features, deciding which suggestions are feasible to implement, writing and circulating drafts until a general agreement is reached.

Designers of new Fortran version have to decide between backward compatibility and the introduction of desirable new features. Although new good programming structures have been introduced, many undesirable structures from earlier version of Fortran have been retained from backward compatibility.

2.2.3 Example of Fortran Code

```
C
C     PARTS 1 AND 2, DEFINE SEQUENCE A SECONDS AND CLOUD NA DURING A
C
      KNA=0
      K1=0
  230 CALL RANDOM_NUMBER(X1)
      A=-DELTA * LOG(X1)
      IF(A.LE.ALIM) GO TO 250
      IF (K1.GE.KT2) GO TO 240
      K1=K1+1
      GO TO 230
  240 A=ALIM/2.0
      X1=0.0
  250 K2=0
  260 CALL RANDOM_NUMBER(X2)
      IF (JW.GT.1) GO TO 280
  270 UX=R*X2
      GO TO 310
  280 CALL RANDOM_NUMBER(TEMP)
      IF (TEMP.GE.0.5) GO TO 290
      UX=UPR + R * (1.0-SQRT(X2))
      GO TO 300
  290 UX=UPR - R * ( 1.0-SQRT(X2))
  300 IF ((UX.GE.0.0).AND.(UX.LE.R)) GO TO 310
      IF (K2.GE.KT2) GO TO 270
      K2=K2+1
      GO TO 260
  310 U=UX
      DA=V3 * EXP(U)
```

2.3 XUL

2.3.1 What is XUL?

XUL means XML User Interface Language and XUL is Mozilla's XML-based language that lets you build feature-rich cross platform applications that can run connected or disconnected from the Internet. XUL was created to make development of Mozilla browser easier and faster. As XML language all features envolving XML are also avalaible to XUL. For example, we can use DOM or Xpath in order to explore a XUL node.

2.3.2 How can we use XUL?

There are several ways for executing a XUL application:

- Firefox extension – an extension adds functionality to the browser itself, often in the form of extra toolbars, context menus, or customizations to the browser's user interface. This is done using a feature of XUL called an overlay, which allows the UI provided from one source, in this case, the Firefox browser, to be merged together with the UI from the extension. Extensions may also be applied to other Mozilla based products such as Thunderbird.

- XUL package – in between the other two are applications which are created in the same way as an extension, but they act like a separate application in a separate window. This is used when you don't want to have the larger size of a complete XULRunner application, but don't mind requiring a Mozilla browser to be installed to be able to run the application.

- Standalone XULRunner application – XULRunner is a packaged version of the Mozilla platform which allows you to create standalone XUL applications. A browser isn't required to run these applications, as they have their own executable file.

- Remote XUL application – you can also just place XUL code on a web server and open it in a browser, as you would any other web page. This method is limited however, as there are security concerns that will limit the kinds of things you will be able to do, such as opening other windows.

Chapter 3

Analysis

3.1 Background

Genfit is a Fortran application dealing with the weight and form of the proteines. It simulates proteine forms comparing with severals models from an internal data base. Genfit has been developed from ten years ago. It contains more than 40000 lines of code and it is using functions and code from Fortran 77 and Fortran 90. Now, it is beeing compiled using gFortran that it is a g95 compiler. As well as Genfit is a SAXS application.

"Small-angle X-ray scattering (SAXS) is a small-angle scattering (SAS) technique where the elastic scattering of X-rays (wavelength 0.1 ... 0.2 nm) by a sample which has inhomogeneities in the nm-range, is recorded at very low angles (typically 0.1 - 10). This angular range contains information about the shape and size of macromolecules, characteristic distances of partially ordered materials, pore sizes, and other data. SAXS is capable of delivering structural information of macromolecules between 5 and 25 nm, of repeat distances in partially ordered systems of up to 150 nm.[1] USAXS (ultra-small angle X-ray scattering) can resolve even larger dimensions."[1]

3.2 Structure of Genfit

3.2.1 Folders and files structure

In this section we are going to describe the folders and files that genfit needs for executing. One of the most difficult parts of this project was to understand this structure. Lack of documentation has been a big probleme in order to modify source code and create the GUI.

We have to remember that Genfit has been developed to be executed on Windows and Linux. Then non-portable code will need be duplicated for Windows and Linux. I'll just assume a tree structure like

- /SAS

  This folder is the root folder of the Genfit application. I contains all the folders and several script files. These script files will be described in next sections.

- /SAS/GENSTORE

[1] Wikipedia

Genstore folder contains all the source code of the application. Besides, Genfit needs to recompile each time you want to execute it. There are many files that they are created during the execution of the scripts. All this process will be explained later.

- /SAS/GENSTORE/PC

  This folder contains all source files and object file for executing on Windows

- /SAS/GENSTORE/LX

  The same as PC but for executing on Linux

3.2.2 Output Folder

In order to execute Genfit and getting the results, an output folder must be created inside SAS folder. This folder will contain not only the output files but the input files as well.

We will describe with more details later which are the input files and and output files that genfit will generate. We can already remark that it is not a good practice to have got in the same folder input and output files.

We will try to arrange in order to have got input files in a folder and output files in another one.

3.3 Input Files

Generally, Genfit need three input files. The first one is a data base with all the models and with the experiments we will want to predict the molecular weight. The second one is a numerical file with the scattering curve and the last one will be another parameter file where we will can correct the error margin.

For convention, the first one will be called from here genABCD.dat.

3.3.1 genABCD.dat File

The genABCD.dat is the main input file. ABCD can be replaced for four numbers that will be called output prefix number. For example, an experiment with the poutput prefix number 0008 will need the gen0008.dat file for executing.

In this file we have got several parts.

General experiments parameters section

This part concerns to the description of the minimization routines used in the fitting procedure. In this first step, we expect to fix the parameters concerning the minimization method. In the next step, we could provide a different input, giving the possibility to choose which minimization method to use.

Here, we have got explained all the parameters:

1. General Description of Fit
   Write details concerning the procedure
2. Output Prefix Name (if not gen//code)
   Choose a prefix different from gen for your output files
3. Maximum MONKEY iterations number
   Quasi-Newton minimization, n. iterations
4. Maximum SIMPLEX iterations number
   simplex minimization, n. iterations
5. Number of Cycles for SIMANN
   simulated annealing, n. cycles
6. Number of Subrun for SIMANN
   simulated annealing, n. subrun
7. Starting Temperature for SIMANN
   simulated annealing, starting temperature
8. Final Temperature for SIMANN
   simulated annealing, final temperature
9. Maximum ZXMIN iterations number
   Zxmin minimization, n. iterations
10. MONKEY minimization flag (1/0)
    if 1,this minimization will be used, otherwise no
11. SIMANN minimization flag (1/0)
    if 1,this minimization will be used, otherwise no
12. SIMPLEX minimization flag (1/0)
    if 1,this minimization will be used, otherwise no
13. ZXMIN minimization flag (1/0)
    if 1,this minimization will be used, otherwise no
14. Iterations number for error analysis
    iterations number inside the error bar

Single Experiment section

The second part of the genABCD.dat file concerns the description of the experiment to be analyzed. In this part only few fields have a particular relevance.

1. Experimental scattering curve
   In the experimental scattering curve the data file has to be indicated together with the folder where it is, considering as you are sitting in the genfit folder. For what concerns the parameters that characterize the single experimental curve, they should be entered corresponding to the window —Par. Symb.— and the name of the parameters follows fortran rules. It means that if you call a parameter iconc and indicate the value 10.5, the software will read iconc=10, because it reads it as an integer. Consequently you can follow my suggestions present in this genABCD.dat file.

2. *Q-range*
   The part concerning the Q-range, the number of Q-points to be averaged and the number of lines to skip at the beginning of each data file, it will follow from the features of the data file resulting from the beamline setting. In any case, I think that each parameter will be fixed

3. Part dedicated to the model simply links the weight of the model to the parameters introduced as characterizing the experimental curve

4. Part concerning the experimental curve and the model to be used, can be extended in order to fit different curves and with different models. For example: if you want to analyze one experimental curve with three different model, you have to copy the three lines concerning the model for three times

Similarly, if you want to fit different curves, you have to copy the SINGLE EXPERIMENT SECTION together with the models to be used, for each experimental curve (of course in each experimental scattering curve window you will indicate different data files).

Models section

The final part of the genABCD.dat file concerns the different models which can be used, each one titled with a precise number. We underline that it does not matter what it is written in the parts dedicated to the models which are not used. Starting from the indications in the SINGLE EXPERIMENT SECTION, the software will go directly to read the part concerning the model to be used.

At the moment, we have got forty eight models but list can get bigger. In fact, adding new models is a requeriment of the future application.

First line of this part begins with the number and description of the model. After that, depending of the model there are some parameters. We will analyze all the possible cases we have got for each model.

We can see here an example of a simple model:

4- Worm-Like Model wihtout excluded volume effect and finite cross section

KhunLength.....:45.43 5. 100. 0
PersistenceLength.....:300. 10. 500. 0
CrossSectionRadius.....:21.47 5. 50.

In this case the example correponds to model number 4 which name is Worm-Like Model wihtout excluded volume effect and finite cross section. As well as it has three parameters called KhunLength, PersistenceLength and CrossSectionRadius.For each parameter we have got several numbers correspondig to starting value, lower value, upper value, and so on....

Sometimes model can need PDB files to fit the experiment. A pdb file is a protein data bank file with information about proteins. So, for few models we will have got another different format of model. We have got at the moment five differents format of models:

1. Format number 1
   4- Worm-Like Model wihtout excluded volume effect and finite cross section
   PDB File (without .pdb) 1 .....:6LYZ
   PDB File (without .pdb) 2 .....:
   PDB File (without .pdb) 3 .....:
   PDB File (without .pdb) 4 .....:
   Maximum Rank of Harmonics (max 30) ...:30
   Max. Q of Partial Amplitudes (Angs.) .:2.0
   Numb. of Q of Partial Amplitudes .....:101
   TCP Packing Distance (Angs.) .....:2.8
   X-ray wavelength 1 (Angs.) .....:0.00
   X-ray wavelength 2 (Angs.) .....:0.00
   Number of Hydration Shells .....:1

2. Format number 2
   PDB File (without .pdb) 1 .....:./lyso/2LYZ
   PDB File (without .pdb) 2 .....:
   PDB File (without .pdb) 3 .....:
   PDB File (without .pdb) 4 .....:
   Random Points Number .....:20000
   Radial grid amplitude (Angs.) .....:1.0

3. Format number 3
   Max. Q of Partial Amplitudes (Angs.) .:0.15
   Numb. of Q of Partial Amplitudes .....:64

4. Format number 4
   PDB File (without .pdb) 1 .....:blgmod
   PDB File (without .pdb) 2 .....:blgmod
   PDB File (without .pdb) 3 .....:
   PDB File (without .pdb) 4 .....:
   Maximum Rank of Harmonics (max 30) ...:30
   Max. Q of Partial Amplitudes (Angs.) .:2.0
   Numb. of Q of Partial Amplitudes .....:101
   FCC Packing Distance (Angs.) .....:3.0
   X-ray wavelength 1 (Angs.) .....:0.00
   X-ray wavelength 2 (Angs.) .....:0.00
   Number of Hydration Shells .....:1
   Reference Temperature (Celsius) .....:0.0
   Reference Pressure (bar) .....:1.0
   Order g(r) Density Expansion (1/0) ...:1
   Number of Intervals for gij(r) .....:20

5. Format number 5
   Point Group Symmetry .....:D5d
   Sampling Sphere Radius (Ang.) .....:400.
   Number of Spheres .....:7
   Random Points Number .....:10000
   Radial grid amplitude (Angs.) .....:3.0

3.3.2 Parameter file

This file enumerates the parameters to be fitted and the range to be used. First time Genfit is executed it creates this file. Later the user can modify this file and Genfit will take it as a input file.

In the field [...] there are the minimum and the maximum value corresponding to the parameter to be fitted. This file will be read from the software in order to start to fit. At the end of the fitting procedure, the software will rewrite this file, entering the fitted value (4.3075312E-05, in this case) and the error bar (5.8303811E-07).

Example of parameter file

1 scal.1= 4.3075312E-05 5.8303811E-07 [1.0000000E-06 - 1.0000000E-03]

3.3.3 Scattering curve file

This file contains the results of the experiments. Generally it has got three columns X,Y and an error margin. This file is needed for Genfit to fit the experiment.

Example

0.1900000E-01 0.3482176E+01 0.1866059E+01
0.1967224E-01 0.3539469E+01 0.1881348E+01
0.2034448E-01 0.3448353E+01 0.1856974E+01
0.2101672E-01 0.3494449E+01 0.1869345E+01
0.2168896E-01 0.3555331E+01 0.1885559E+01
0.2236120E-01 0.3507842E+01 0.1872923E+01
0.2303344E-01 0.3442698E+01 0.1855451E+01
0.2370569E-01 0.3502810E+01 0.1871580E+01
0.2908361E-01 0.3161165E+01 0.1777967E+01
0.2975585E-01 0.3113212E+01 0.1764430E+01
0.3042809E-01 0.3010831E+01 0.1735175E+01
0.3110033E-01 0.2943150E+01 0.1715561E+01
0.3177258E-01 0.2923023E+01 0.1709685E+01
0.3244482E-01 0.2855408E+01 0.1689795E+01
0.3311706E-01 0.2813714E+01 0.1677413E+01
0.3378930E-01 0.2773319E+01 0.1665329E+01
0.3446154E-01 0.2735078E+01 0.1653807E+01
0.3513378E-01 0.2681530E+01 0.1637538E+01
0.3580602E-01 0.2599799E+01 0.1612389E+01
0.3647826E-01 0.2598789E+01 0.1612076E+01
0.3715050E-01 0.2647779E+01 0.1627200E+01
0.3782274E-01 0.2546956E+01 0.1595918E+01
0.3849498E-01 0.2527685E+01 0.1589869E+01
0.3916722E-01 0.2416698E+01 0.1554573E+01
0.3983946E-01 0.2482120E+01 0.1575474E+01
0.4051171E-01 0.2351255E+01 0.1533380E+01
0.4118395E-01 0.2329867E+01 0.1526390E+01
0.4185619E-01 0.2302213E+01 0.1517305E+01
0.4252843E-01 0.2258577E+01 0.1502856E+01
0.4320067E-01 0.2216767E+01 0.1488881E+01
0.4387291E-01 0.2175419E+01 0.1474930E+01

3.4 Output Files

Once we have executed Genfit application we can see that some files have been generated. We are going to have got some words about these files.

3.4.1 genABCD.out File

It contains the most complete description of fit results. It contains the minimization procedures used, the final functional value, the resulting parameters. It is the more user friendly output file.

Example

SIMPLEX Calls To Routine CHI2 ......: 0
ZXMIN Calls To Routine CHI2 ....: 0
Achieved Convergence; Error Flag .....: 1492212136
Final Functional Value ....: 5.153427E+47
Functional Gradient Norm ....: 0.00000

Final set of optimized parameters: raw form

1) corrcpro = 0.000000 0.000000
2) cpro = 0.000000 0.000000
3) peso = 0.000000 0.000000
4) x11 = 0.000000 0.000000
5) y11 = 0.000000 0.000000

Final set of optimized parameters: scaled form

1) corrcpro = 9.9995493E+07 0.000000 [0.000000 - 1.9999099E+08]
2) cpro = -9.9990941E+07 0.000000 [-1.9998188E+08 - 0.000000]
3) peso = -6282.010 0.000000 [-1.9998987E+08 - 1.9997731E+08]
4) x11 = 1586.487 0.000000 [-1.9999064E+08 - 1.9999382E+08]
5) y11 = 1871.970 0.000000 [-1.9999489E+08 - 1.9999864E+08]

Experiment ....: 1
==========

File Name ....:mio7cin01.ass
Description of Experiment ....:Experiment

Final Functional Value ....: 5.153427E+47
Scale Factor kappa ....: 1.00000 0.00000
Flat Background B ....: -3.619365E+18 0.00000

3.4.2 genABCD01.fit File

It contains the experimental curve and the theoretical one. The numbers following the code genABCD are referring to the number of fitted curves. If we are analyzing just one curve, we will obtain only a genABCD01.fit output. Otherwise, we will obtain a number of *.fit files equal to the number of fitted curves. The numbers correspond to the series of the experimental data written in SINGLE EXPERIMENT SECTION in the genABCD.dat file. Each *.fit file contains 6 columns: Q values, experimental Intensity I, experimental log I, theoretical Intensity I, Theoretical log I, experimental error for the Intensity I. This file has to be used to create a

graph. The easier way to proceed is to create a graph with experimental logI in function of Q2 and the corresponding theoretical logI. genABCD.log It contains the complete list of fitted parameters. It could be used to read the resulting parameters to show as result

3.4.3 genABCD.log File

It contains the complete list of fitted parameters. It could be used to read the resulting parameters to show as result.

3.5 Genfit Scripts

There are several scripts inside Genfit that It's worth keeping in mind. One of them is called Gallo and his mission is to execute the application Genfit. The other one is lxgen who is in charge of compiling.

3.5.1 Gallo Script

Gallo is in charge of the execution of the application. Gallo calls to lxgen because a pre-compilation is need before executing genfit.

The command to execute Gallo is:

gallo 0002 last 1 1

where 0002 is the output prefix name.

last is the folder where the input files are and where the output files will be.

Thrid parameter is 1 if we are using fourier transform files

4th parameters is 1 if we have modified the parameter file.

Basically, Gallo copy the input files (see Input files chapter) inside the root of the application. It executes lxgen with parameter 2 and after with parameter 4. After that all output files are copied to the output directory.

3.6 Fortran reading code

Genfit is a huge Fortran application developed for ten years ago. Concerning to this project just the few first thousands of lines are interesting because it is the code where the input file is been read.

All this code concerning to the reading of the input file is in genfit.f. ".f" is the natural extension of the fortran files. ".f90" and ".f95" are used as well when a fortran 90 or 95 is been used.

3.6.1 Reading routines

Genfit is reading input file line per line. Fortran uses a format statement to indicate wich type of value it is going to read. They are called format edit descriptors.

So, fortran code will need to have got a format edit descript for each line that it is going to read. This point in addition to the fact that fortran reads in a sequecial way we can realize that it will be very easy to make a mistake in the input file that it will provoque the failure on the execution of the application.

We can see some examples of reading setences:
write(*, 900) i, x
900 format (I4,F8.3)

In this case the format label 900 is chosen somewhat arbitrarily, but it is common practice to number format statements with higher numbers than the control flow labels. After the keyword format follows the format codes enclosed in parenthesis. The code I4 stands for an integer with width four, while F8.3 means that the number should be printed using fixed point notation with field width 8 and 3 decimal places.

3.6.2 Common format codes

The most common format code letters are:

1. A - text string
2. D - double precision numbers, exponent notation
3. E - real numbers, exponent notation
4. F - real numbers, fixed point format
5. I - integer

3.7 Main difficulties

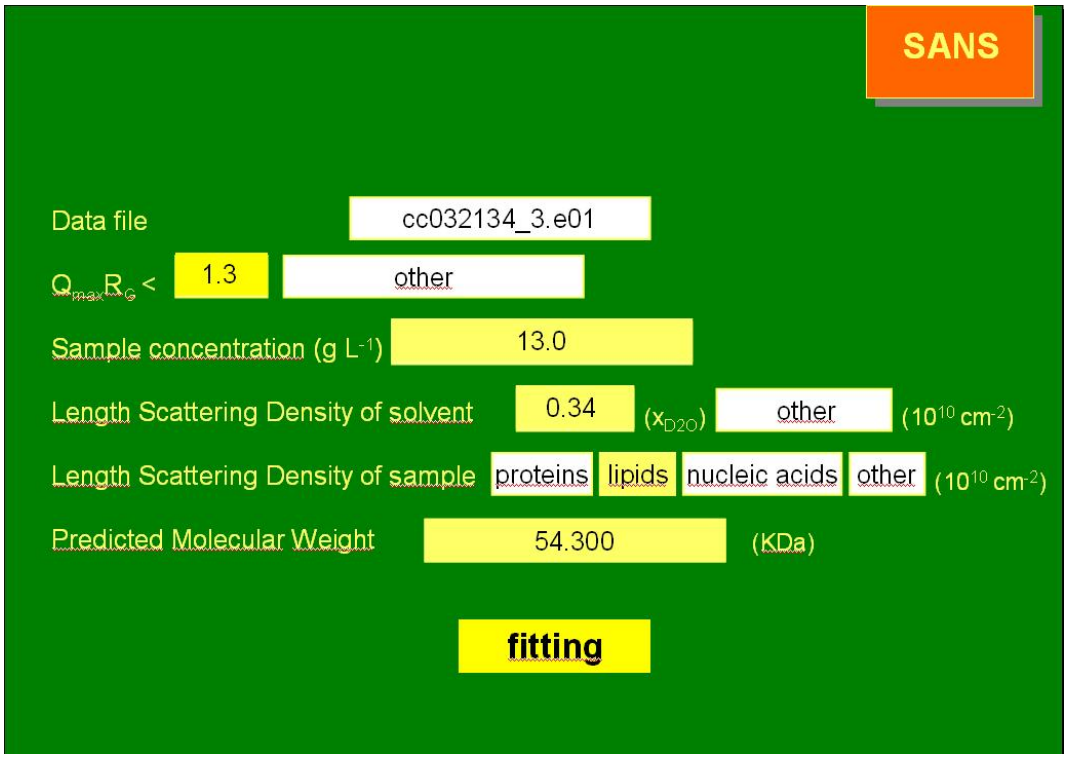
1. Mainly, user needs to know the correct format of the input files. Specially on genABCD.dat, this file is very sensitive to mistakes. If a number is not well placed the fortran program will not be able to read the value and will failure.
2. User needs to know how to execute the application, I mean, which parameters Gallo need and to deal with the linux terminal or with Windows command line.
3. User needs to know how to prepare the output folder in order to Genfit be able to read it. User have to create a outptu folder where genABCD.dat and other input files will be placed in.
4. The fact of users facing with a enormous file with more than 30 thousands parameters and at the momento 50 models and getting bigger represents a too big challenge for users trying to modify the input parameters.

3.8 Proposal from the user

At the beginning of the project the client proposes a schema or draft about a possible solution. This solution was given in PowerPoint format and it's showing how the user wants the application looks like.

In our first analysis we saw that this solution was a too much simple solution. It didn't support all models and even for just one model, It didn't take account all possible cases that we can have got.

Even of been a not realistic solution, it was good due to we can see what the user was expecting more or less. How he was expeting type the values and we could get an idea of ow the user wants to interact with the application.



Chapter 4

Design

4.1 Introduction

To add a Graphic user interface to make it easier for users is the aim of any GUI. End-users face to software via GUI, it means that by creating a very well-done software is not the unique condition for our software to success. It will need an adequate interface between users and software. Sometimes software developers invest not enough time in the GUI provoking in most of the cases a non-acceptation of the software created.

Before starting the description of our design we want to focus on the major problems in the software engineering. Some of them, we can say, are critics. Next lines I will write some of them because I consider important to keep in mind.

Major problems developing software:

1. Too much code. Either tens of thousands, or hundreds of thousands, or millions of lines of code for an "Application".
2. No one really understands exactly what the code is doing.
3. There is insufficient documentation. In fact, it is so unusual to have documentation, it is ignored when it does exist.
4. We have little chance of ever understanding all of the interactions between different parts of a large system.
5. The people who do have a chance of understanding the code, very often don't understand the application.
6. People do not reuse existing software either because they don't trust it or because it is not general enough.

Specifically in my case these problems are bigger because application deals with a very complex data analysis calculations. So, it is required to have got a very high level in physics to understand the program is doing.

In addition, lack of documentation is present in this situation and obvious things for scientific can provoke a bad understanding for programmers who don't have got enough knowledge in mechanical, psychical and biological fields.

4.2 Methodology

The aim of this project is to build an user interface for a single application but the problem we try to solve is the same in most of the cases of fortran applications. We mean that users facing to huge and very complex input files is a trouble at the ESRF and programmers have to deal with.

Then, if we are able to give a general solution for this problem or if we are able to give them a new or better solution our project will be more valuable. Given a better solution means for programmers to be able to build users interfaces in a easier and faster way.

Furthermore, the quality of the software is even more important than getting it as fast as possible. If we are able to create a software by using a easy and fast way but with a bad quality will provoke a more complex supporting work later. and probably project will be abandoned some months later.

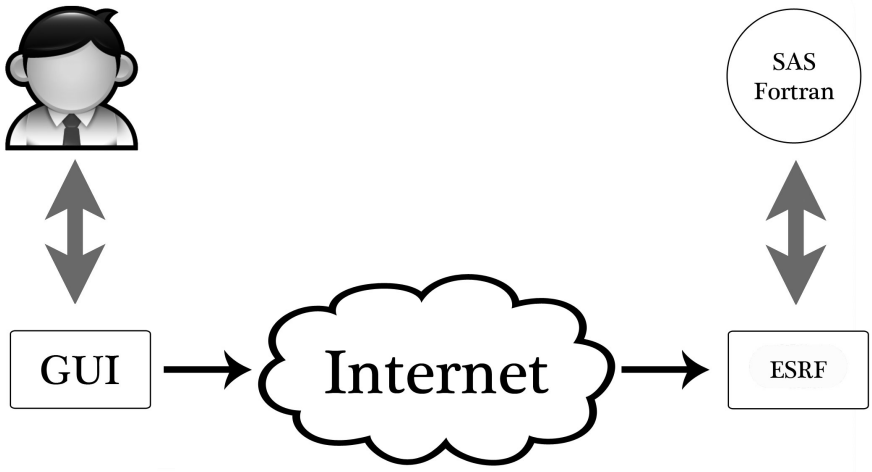
Then, our solution will be designed with these priorities and in this order:

1. Quality.
2. Easy to do.
3. Fast to do.

4.2.1 User interface as isolated system

An interesting thing we would like to do is to isolate the user interface. It means to be able to execute the application both with user interface and without user interface depending of the user preferences. Advanced users or users having the habit to use his application without user interface will follow using his application in a traditional way.

Besides each more time popular remote technologies can provide us abilities to execute user interfaces remotely without security restrictions. We mean, we can have got the fortran program in our server or in a main frame inside of the intranet and users can execute the user interface in remote mode from his home or university.



In fact, this last point is very interesting due to two important points: first one is that fortran application require compilers and dependencies that they can not be solved on local machines as well as users can require execute fortran application on a main frame with high calculation performance instead of their home computers. Second one is that users can execute experiments from their home or university in the same way as if they were at the ESRF.

We can go further and say that is possible to create a generic user interface system dealing with all fortran applications.

4.2.2 Technologies involved

Usually, combining different technologies in the same project could make us to think that the project will be more complex. A developer will need first to learn all languages and later to manage them for programming systems.

However, technologies involved in this project are created specifically to make it easier for developers. So we are using several languages and all of them have been created to be easier than predecesors. And by combining all of them we can expect not to increase the complexity of the project if not to reduce it.

As programming language we have chosen Javascript, XUL (XML User interface Language) as interface language, XML for storing data and Fortran 95 for our main program.

Javascript

We have chosen Javascript because is very similar to c++ and java. So the number of new concepts required to learn javascript is very small.

Javascript is both procedural and object-oriented language. Then we can create classes or we can develop as a procedural language or we can combine both options. That is really good because programmers who aren't very familiar with oriented-object programming can develop their software in procedural manner or vice-verse.

In fact, Javascript is called first-class function. That means we declare a function and it acts as a procedural function but as well as we can declare another functions inside of this function. Then this first function will become automatically a class. And we will can instantiate objects of this class.

XML User interface Language

XUL is a XML language created to be easy and faster. Besides we can use behind XUL another languages like javascript, python etc. Developers of XUL were very fanatical of the portability. Then XUL applications can be executed on Linux, Windows, Mac and as Mozilla Firefox Extension without changing any line of code.

As XML language we can insert any XML language inside of XUL. For example we can insert a piece of code of HTML, SVG (Scalable Vector Graphics) or MathML (Mathematical Markup Language).

We will use XUL to build the user interface. So we will create a .xul file for each window that we need.

XML for storing data

XML is a standard and portable language for storing data. User interface need load and store information from user or another programs in anywhere. XML gets us the possibility to access information in a standard way.

Another advantage of XUL is that practically all programming languages implement libraries to read and store data in XML format. Most of text editors are able to read XML formats and XML has been placed as a standard in facto.

We will see later than XML offers as well as the possibility to be used as the input file of the Fortran program just adding a new library in charge of read and write XML for Fortran. This approach will be seen in next chapters.

Fortran 95

Of course, we must not forget that this user interface we are thinking in is to be used on a fortran application. This fortran application will receive a input file and we will deal with it. We have got several options or our user interface creates a input file transforming XML data in expected input file or we modify the fortran program in order to get it able to read XML files.

This point will discuss with much more details in next chapters.

4.3 Users' cases

Once we have decided which technologies we will use we need to have a look with a details a what this user interface is expected to do. The way to do that will be describing users' cases.

Before starting describing all users cases we are going to organize them in two classes. The fist one will be users cases which only administrator or advanced users can do. And the second one will be theses users cases which everybody can do even administrator.

By separating in two different modes is required because there are some tasks that users can not do by themselves. We will describe here with detail which they are. Then this chapter will be divided in two different subchapters: Users cases in administration mode and in user mode.

Of course, administration mode extends all features of user mode.

4.3.1 User mode

Users can add new calculations and of course to execute the SAS application. This last point can be done in two different ways. In the first one user will can to execute a single input file and in the second one user will can execute several input files. This last one way was though as a kind of batch mode. Scientific can execute a huge experiment and go home to sleep expecting to have results one day later or perhaps two or more.

Case number 1. Adding new experiment

Then user will be able to insert a new experiment in the application. Managing this experiments is related in case number 2.

Case number 2. Editing experiments

Once user have added a new experiment it must be able to:

1. Modify all parameters concerning to this experiment. This parameters are static and doesn't change for each experiment. Some examples of these parameters are: scaling factor, wavelength band, source-sample distance, etc....
2. Modify models inside of the experiment. It means user can add, remove and modify models of the experiments. Besides each model has got some parameters which user will need deal with.
3. Modify parameters of the experiment. These parameters are not static. It means user can add as more parameters as he want. Depending of the model these parameters will change. User can choose if he adds them or not. Some examples are: weight, concentration, etc...

Case number 3. Executing a single experiment

The user interface will need to able to execute the Fortran application. It means that one of its tasks will be to call the Fortran application and wait for its results.

We have chosen an isolated user interface it means that the fortran application is not inside of the user interface. So a parameters we need will be the path to the Fortran application. Another one will be the path to the file with the scattering curve and of course the path where we want to have got the output files after execution of the program.

1. Path to the folder where GENFIT application is. Perhaps it is in another machine we will need the complete path to it.
2. Path to the folder where we can find the file with the scattering curve.
3. Path to the folder where user wants to have got all the output files.

Two more parameters are required. One of them is to know if we are using Fourier transform in our experiment. The second one is to know if we are using error margin file. In positive case we will need the path to the error file we are using. Son, new parameters:

1. Fourier transform parameters
2. Path to margin error file
3. Margin error parameter

Case number 4. Executing multiple experiments

In this case user wants to execute a very long experiment or perhaps two or more experiments. Then application will execute one experiments after another one.

For executing each experiment we will need a complete input file. Then a parameter of this users case will be the path to the XML file and of course, all parameters required for executing a single experiment explained in the users case number three.

These parameters will be:

1. Path to all XML files containing input files.
2. Path to the folder where GENFIT application is. Perhaps it is in another machine we will need the complete path to it.
3. Path to the folder where we can find the file with the scattering curve.
4. Path to the folder where user wants to have got all the output files.

4.3.2 Administration mode

In this section we will describe what are the tasks that the graphic user interface must be able to do in administration mode.

Furthermore we will need to plan a system for the user interface for user to switch from user mode and administration mode and vice-verse

Case number 1. Adding a new model

Administrator can to extend the number of models of the application. It is not a common task and it is reserved to experimented users.

Case number 2. Modifying a model

Administrator can modify all the settings of a model already created. These settings are values as for example: random point numbers, radial grid, maximum rank harmonics, point of symmetry, etc....

Case number 3. Adding a new parameter

For each model we will can have got more than one parameter. For example for the model called "Gaussian chain with finite cross section" we have got two parameters called: "Gyration radius" and "Cross section radius". User will be able to add new parameters.

Case number 4. Modifying parameters

Each parameters has a more or less static structure with the next values: starting, lower, upper, flag, kind, grid, lower integer, upper integer, etc...

For each model and inside of each model for each parameters the user in administration mode will be able to manipulate all this information about the parameters.

4.4 Implementation

This chapter will deals with how this application has been made. At the beginning of the project we had no previous knowledge about XUL, Fortran and user interfaces. So, we can say that this project has been developed completely from scratch. That means a few libraries have been created for us to help the development apart of all things concerning to the user interface: windows, windows code and Fortran code.

4.4.1 Two main approaches

F rom the beginning we thought in two main approaches, one of them more sophisticated but both of them with identical results. In this chapter we will describe these two approaches:

- To modify the fortran code of the SAXS application in order to make it to read XML files using a specific library to do that.
- Create a translator between XML and the expected input file.

4.4.2 Using XML-Fortran reader

Using xml-Fortran library we are able to read from Fortran to XML files. This library provided Fortran programmers have an all-Fortran solution for reading and writing XML files. We could use this library for:

- Reading the files in much the same way as an ordinary text file - a stream-oriented method
-
- Directly reading the contents into a data structure that reflects the structure of the XML file
- Constructing a tree of data from the XML file with essentially the same structure
- Processing the contents using "events", in much the same way as the original Expat library.

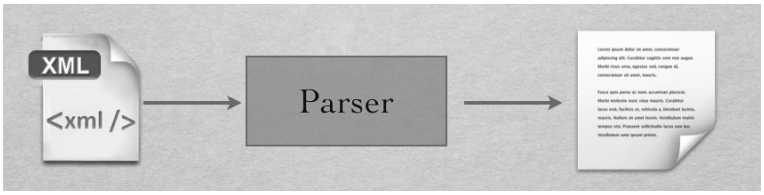
The most difficult challenge with this approach was to modify the Fortran code adding this library.

Why was it a challenge?. Because:

- No documentation about the code.
- More than 40.000 thousand lines in the same code file
- Code not structured at all.
- Development started ten years ago up to our days using functions of the sixties.
- and so on...

XML-fortran works in the same way. You must build a schema with the data structure you want to use. Once you have created this data structure we need to run the XM-fortran parser which will convert automatically this schema to a fortran 90 class.

This generated class will have got all functions to read and write a XML file with the structure expected. We can see in the next picture the process.



We will need to modify all the routines in the fortran program concerning to the reading of the data by our new functions. This step is quite hard because of the number of lines we need to modify and because we will new to understand all the fortran code before executing any changes.

In the next listing we can see how the XML schema looks like:

```
<?xml version="1.0"?>
<genfit>
   <comment>
   This is an example of a template for the xmlreader program used by GenFit
   </comment>

   <options strict="yes" globaltype="no" rootname="genfit"/>

   <placeholder tag="fitparameters" optional="no" strict="yes">
      <variable name="GeneralDescription" type="line" length="70"></variable>
      <variable name="OutputPrefixName" type="line" length="40"></variable>
      <variable name="MaximumMONKEYIterations" type="integer" length="40"></variable>
      <variable name="MaximumSIMPLEXIterations" type="integer" length="40"></variable>
      <variable name="NumberSIMANNCycles" type="integer" length="40"></variable>
      <variable name="NumberSIMANNSubRuns" type="integer" length="40"></variable>
      <variable name="SIMANNStartingTemp" type="real" length="8"></variable>
      <variable name="SIMANNFinalTemp" type="real" length="8"></variable>
      <variable name="MaxZXMINIterations" type="integer" length="40"></variable>
      <variable name="MinimizationFlagMONKEY" type="integer" length="40"></variable>
      <variable name="MinimizationFlagSIMANN" type="integer" length="40"></variable>
      <variable name="MinimizationFlagZXMIN" type="integer" length="40"></variable>
      <variable name="MinimizationFlagSIMPLEX" type="integer" length="40"></variable>
       <variable name="ErrorIterations" type="integer" length="40"></variable>

    </placeholder>
   <placeholder tag="experimentparameters"  strict="yes">
      <variable name="ExperimentalScatteringCurve" type="line" length="40"></variable>
      <variable name="ExperimentDescription" type="line" length="40"></variable>
      <variable name="DataFileMinQAngs" type="real" length="8"></variable>
      <variable name="DataFileMaxQAngs" type="real" length="8"></variable>
      <variable name="PoissionErrorKValue" type="real" length="8"></variable>
      <variable name="DataFileSkipLines" type="integer"></variable>
      <variable name="QPointsToAverage" type="integer" ></variable>
      <variable name="NumberofModel" type="integer"></variable>
      <variable name="NumberPointsConvolution" type="integer" ></variable>
      <variable name="RadiusSourceSlit" type="real" length="8"></variable>
      <variable name="RadiusSampleSlit" type="real" length="8"></variable>
      <variable name="SourceSampleDistance" type="real" length="8"></variable>
      <variable name="SampleDetectorDistance" type="real" length="8"></variable>
      <variable name="Wavelength" type="real" length="8"></variable>
      <variable name="WavelengthBand" type="real" length="8"></variable>
      <variable name="SpacialResDetector" type="real" length="8"></variable>
      <variable name="SamplingRadialAverage" type="real" length="8"></variable>
      <variable name="QVerticalSlit" type="real" length="8"></variable>
      <variable name="QMaxVerticalSlit" type="real" length="8"></variable>
      <variable name="QHorizontalSlit" type="real" length="8"></variable>
      <variable name="QMaxHorizontalSlit" type="real" length="8"></variable>
      <variable name="flagCollimation" type="integer"></variable>
      <variable name="flagDetector" type="integer"></variable>
      <variable name="flagRadialAverage" type="integer"></variable>
      <variable name="flagVerticalSlit" type="integer"></variable>
      <variable name="flagHorizontalSlit" type="integer" ></variable>
      <variable name="WavelengthBandFlag" type="integer" ></variable>
      <variable name="numberModels" type="integer"></component>
   </placeholder>

    <typedef name="modelweights_t" strict="yes">
      <component name="modelNumber" type="integer"></component>
      <component name="starting" type="real" length="8"></component>
      <component name="lower" type="real" length="8"></component>
      <component name="upper" type="real" length="8"></component>
      <component name="flag"  type="real" length="8"></component>
      <component name="link"  type="line" length="300"></component>
   </typedef>

   <typedef name="model_t" strict="yes">
      <component name="description" type="line" length="40"></component>
      <component name="starting" type="line" length="40"></component>
      <component name="lower" type="line" length="40"></component>
      <component name="upper" type="line" length="40"></component>
      <component name="flag"  type="line" length="40"></component>
      <component name="kind" type="line" length="40"></component>
      <component name="lowerInt" type="line" length="40"></component>
      <component name="UpperInt"  type="line" length="40"></component>
      <component name="lowerParOne" type="line" length="40"></component>
      <component name="upperParOne"  type="line" length="40"></component>
      <component name="lowerParTwo" type="line" length="40"></component>
      <component name="upperParTwo"  type="line" length="40"></component>
   </typedef>

   <typedef name="genfitmodel" strict="yes">
      <component name="ModelNo" type="line" length="40"></component>
      <component name="ModelType" type="line" length="100"></component>
      <component name="ModelDesc" type="line" length="100"></component>
      <component name="model"  type="model_t" dimension="1"></component>
   </typedef>

   <variable name="modelweights" type="modelweights_t"  dimension="1"></variable>
   <variable name="genfitmodels" type="genfitmodel" dimension="1"></variable>
</genfit>
```

Listing 4.1: schema.xml

We can see how xml-fortran parser has generated the fortran class with this small piece of code:

```
module xml_data_xchema
    use READ_XML_PRIMITIVES
    use XMLPARSE
    implicit none
    integer, private :: lurep_
    logical, private :: strict_
    character(len=70)                                  :: GeneralDescription
    character(len=40)                                  :: OutputPrefixName
    integer                                            :: MaximumMONKEYIterations
    integer                                            :: MaximumSIMPLEXIterations
    integer                                            :: NumberSIMANNCycles
    integer                                            :: NumberSIMANNSubRuns
    real                                               :: SIMANNStartingTemp
    real                                               :: SIMANNFinalTemp
    integer                                            :: MaxZXMINIterations
    integer                                            :: MinimizationFlagMONKEY
    integer                                            :: MinimizationFlagSIMANN
    integer                                            :: MinimizationFlagZXMIN
    integer                                            :: MinimizationFlagSIMPLEX
    integer                                            :: ErrorIterations
    character(len=40)                                  :: ExperimentalScatteringCurve
    character(len=70)                                  :: ExperimentDescription
    real                                               :: DataFileMinQAngs
    real                                               :: DataFileMaxQAngs
    real                                               :: PoissionErrorKValue
    integer                                            :: DataFileSkipLines
    integer                                            :: QPointsToAverage
    integer                                            :: QPointsErrorBarMove
    real                                               :: PrMaxDistance
    integer                                            :: ScalingFactor
    integer                                            :: BackgroundFlag
    integer                                            :: NumberofModel
    integer                                            :: NumberPointsConvolution
    real                                               :: RadiusSourceSlit
    real                                               :: RadiusSampleSlit
    real                                               :: SourceSampleDistance
    real                                               :: SampleDetectorDistance
    real                                               :: Wavelength
    real                                               :: WavelengthBand
    real                                               :: SpacialResDetector
    real                                               :: SamplingRadialAverage
    real                                               :: QVerticalSlit
    real                                               :: QMaxVerticalSlit
    real                                               :: QHorizontalSlit
    real                                               :: QMaxHorizontalSlit
    integer                                            :: flagCollimation
    integer                                            :: flagDetector
    integer                                            :: flagRadialAverage
    integer                                            :: flagVerticalSlit
    integer                                            :: flagHorizontalSlit
    integer                                            :: WavelengthBandFlag
    integer                                            :: UsingExponentAlpha
    integer                                            :: ExponentAlpha
    integer                                            :: numberModels
    integer                                            :: numberParameters
    integer                                            :: numberSingleSections
```

Listing 4.2: Fragment of schema.f90

And a function to read an array:

```
subroutine read_xml_type_weight_t_array( &
      info, tag, endtag, attribs, noattribs, data, nodata, &
      dvar, has_dvar )
    type(XML_PARSE)                                    :: info
    character(len=*), intent(inout)                    :: tag
    logical, intent(inout)                             :: endtag
    character(len=*), dimension(:,:), intent(inout)    :: attribs
    integer, intent(inout)                             :: noattribs
    character(len=*), dimension(:), intent(inout)      :: data
    integer, intent(inout)                             :: nodata
    type(weight_t), dimension(:), pointer :: dvar
    logical, intent(inout)                          :: has_dvar

    integer                                         :: newsize
    type(weight_t), dimension(:), pointer :: newvar

    newsize = size(dvar) + 1
    allocate( newvar(1:newsize) )
    newvar(1:newsize-1) = dvar
    deallocate( dvar )
    dvar => newvar

    call read_xml_type_weight_t( info, tag, endtag, attribs, noattribs, data, nodata, &
                dvar(newsize), has_dvar )
end subroutine read_xml_type_weight_t_array
```

Listing 4.3: Fragment of schema.f90

We could test that everything was working correctly and after the modification of the SAXS program our modification were reading correct values. However, and speaking with fortran

experts at the ESRF we saw that, probably, there was a incompatibility between the code of the SAXS application which was written in fortran 77 and migrated to fortran 95 later with the fortran 95 code of the library.

Problem was that until scientific didn't test the program and saw the results we couldn't realize that the results generated are not corrects. It was hard after working with the fortran code during months to change completely the approach.

Anyway, this approach is very interesting and programmers here at the ESRF are already seen how to use this XML fortran for future projects.

4.4.3 Creating a translator

A translator means to create a software tool which is in charge to carry out the translation between our XML file and the expected input file with the appropriate format. This solution is quite good because no changes in the fortran code are required and we can isolated at maximum the dependencies between the SAXS application ant the user interface.

Furthermore we can be able to carry out the addition of new models and new parameters. That means that we have to create a pseudo-inteligent generator of file. Besides the not very easy format of the expected file brings us the problem of the generation itself, we mean, what is the best way to generate a input file? via generating from scracth or replacing values?.

This solution has been much faster to develop than the first one. It is not very sophisticated from the programming point of view but it is really good because we have got exactly the same advantages, like editing/adding new models, than in the first one and easier and faster.

Templates

Firstly we have studied the different type of format that we have got in the input file for each model. We have realized that we have got more than five types of formats. Then we have created a template for each different format. As well as we have got associated each model with his corresponding format.

The process of translation has been carry out by iterating each model and replacing the values in the template. Later, by software of course, it will copy and paste each fragment forming the complete input file.

We can see here an example of template:

```
1  ** Input File for GENFIT program 3.00 generated by GUI Genfit **
2
3   GENERAL SECTION ########################################
4
5    General Description of Fit ..........:$GeneralDescription
6   Output Prefix Name (if not gen//code) :$OutputPrefixName
7   Maximum MONKEY iterations number ....:$MaximumMONKEYIterations
8   Maximum SIMPLEX iterations number ....:$MaximumSIMPLEXIterations
9   Number of Cycles for SIMANN ..........:$NumberSIMANNCycles
10  Number of Subrun for SIMANN ..........:$NumberSIMANNSubRuns
11  Starting Temperature for SIMANN ......:$SIMANNStartingTemp
12  Final Temperature for SIMANN .........:$SIMANNFinalTemp
13  Maximum ZXMIN   iterations number ....:$MaxZXMINIterations
14  MONKEY minimization flag (1/0) ......:$MinimizationFlagMONKEY
15  SIMANN minimization flag (1/0) ......:$MinimizationFlagSIMANN
16  SIMPLEX minimization flag (1/0) ......:$MinimizationFlagSIMPLEX
17  ZXMIN   minimization flag (1/0) ......:$MinimizationFlagZXMIN
18  Iterations number for error analysis .:$ErrorIterations
```

Listing 4.4: Model Template

Translation tags

Tags marked with the symbol dollar are recognized by the software and translate automatically by its value at the XML. All this process is carried out each time we want to execute an experiment.

Furthermore we give the user the possibility to export the file as a ASCII file. Because it has been an aim of this project to give the user the opportunity to execute his SAXS application with or without user interface. So exporting the input file can help the user but he has got the choice to execute SAXS with the user interface or just to create the input file with it.

4.5 Structure of code and windows

AS we told before we have done this application from scratch and we have had to define some libraries that we have reuse in the application. Reusing code is one of the quality factors of the programming engineering.

4.5.1 XUL Folder Structure

XUL uses a generic folder structure. We need to have the same structure if we want our application running on linux, windows and inside of Firefox. The folder structure looks like:

```
helloworld/
  chrome.manifest
  install.rdf
  content/
    overlay.js
    overlay.xul
    hello.xul
  locale/
    en-US/
      overlay.dtd
      hello.dtd
  skin/
      overlay.css
```

where the purpose of each file is:

Filename	Purpose
chrome.manifest	Tells Firefox where your chrome files are and what to do with them
install.rdf	The description file for your extension ("Install manifest")
overlay.xul	The file describing UI elements to add to the browser window
overlay.js	The file with scripts to run in the browser window
overlay.dtd	Contains translation for text string codes in overlay.xul
hello.dtd	Contains translation for the strings in hello.xul
overlay.css	Lets you adjust appearance of UI elements with CSS
hello.xul	The file describing the UI of the new window
helloworld@mozilla.doslash.org	A pointer to your extension files

4.5.2 Code Folder Structure

As a XUL application all windows code are in the /content directory. This is the unique constraint we have got with XUL. We need to arrange a good structure to manage the code of our application. We thought next structure:

- /content: This folder contains all our code and files
- /content/JS: containing all .JS files that have got a general use.
- /content/JSWindows: containing all the code of the Windows. For example if we have got a window called main.xul we will have got a main.js file insidi of JSWindows directory. This is the way for us to separate the code specific for each window.
- /content/NewModels: Folder containing all files to add new models
- /content/Templates: Containing all files to generate the input file.

4.6 Dealing with XML from XUL

We have seen that XUL is a XML language and sometimes we will need to read and save information in a XML format. Specially if we have translated a input text file to a XML file. In this section we would like to clarify from XUL what are the possibilities to deal with XML information.

There are several ways to manage data in XML format from XUL. I have been using two: DOM and Xpath. Why have I used these two technologies?. Because they are standards.

4.6.1 Xpath

XPath is a language for finding information in an XML document. XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system.

XPath became a W3C Recommendation 16. November 1999.

4.6.2 DOM

The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. This is an overview of DOM-related materials here at W3C and around the web.

4.7 Classes

The use of classes makes the development easier for programmers. In this case they are dummy classes because they have not functionality, no methods are implemented inside of the classes.

They are used to contain the data of the XML in a more human readable way.

Next list shows some specification of the classes: model, parameter and single

```
/*
        THIS FILE GENFIT OBJECTS ARE DEFINED: MODEL, PARAMETERS AND SINGLE SECTION

        Classes:
                        Model(vartype, vardescription, varnumber)
                        Parameter(desc,Starting,Lower,Upper,Flag,Kind,Grid, lowerInt,
                                        upperInt, lowerPar1,upperPar1,Link1,lowerPar2,upperPar2,Link2)
                        Single(filePath, desc)

        Functions
                        CountParameterModel(modelNumber)
                        CountTotalModel()
                        CountTotalSingle()

                        returnSingleExperiments()
                        -- return array of Single objects
                        returnModels()
                        -- return array of model objects
                        returnParameters(modelNumber, parameterNumber)
                        --- return array of parameter objects


*/

//Define the object Model
function Model(vartype, vardescription, varnumber)
{

        this.tipo=vartype;
        this.desc=vardescription;
        this.number=varnumber;

}
//Define the object parameter
function Parameter(desc,Starting,Lower,Upper,Flag,Kind,Grid, lowerInt,
                                    upperInt, lowerPar1,upperPar1,Link1,lowerPar2,upperPar2,Link2)
{
        this.desc=desc;
        this.starting=Starting;
        this.lower=Lower;
        this.upper=Upper;
        this.flag=Flag;
        this.kind=Kind;
        this.grid=Grid;
        this.lowerInt=lowerInt;
        this.upperInt=upperInt;
        this.lowerPar1=lowerPar1;
        this.upperPar1=upperPar1;
        this.link1=Link1;
        this.upperPar2=upperPar2;
        this.lowerPar2=lowerPar2;
        this.link2=Link2;

}
//Define the single experiment Section to show on the three
function Single(filePath, desc, models)
{

        this.FilePath=filePath;
        this.Desc=desc;
        this.Models=models;

}
```

Listing 4.5: Genfit Classes

Chapter 5

Results

5.1 Study of complexity of our application

There are too many opinions about how to measure the complexity of software. In fact it is not very simple to evaluate the complexity of a software because there are too many kind of software different. For example we can not compare easily a Fortran application with a web page. Both of these cases are software but it is quite difficult to compare them. Anyway as far as computing programmer concerns if the complexity of the software is low we will can modify the software easily for fixing bugs or to extend it.

The aim of this report and project is not to enter in discussions about how to measure software complexity. So, I will use a system by counting lines of code. Why are we using this way?. Because it is easy to apply and it gives us an idea about the size of the software. Besides we can see clearly how many lines of code we are reusing. It is the way to see if in the future we want to build another user interface knowing how many package, libraries etc... we can reuse we will know how much time we will save developing the software.

In our case, we can think the application as two subsystems of code. One of them is our specific application itself, its windows, its code and the other one is a set of general libraries that we will reuse in the future.

So, we will see in this section how many files we have got a how many lines inside of each file. Then we will separate the files of our application and the files of the libraries.

5.1.1 Application: windows and code

Firstly we are going to see a list with all files referring to Windows code. It means the files which perform task for windows. For example: load the xml file, load and save data, process the input file and answer to event like mouse or keyboard

```
 1 133 ExperimentParameters.js
 2   219 MultipleProcess.js
 3   139 ParameterSingleSection.js
 4    55 Parameters.js
 5   179 Process.js
 6   264 SingleSectionEdit.js
 7   326 SingleSectionModels.js
 8   139 SingleSectionParameter.js
 9   256 editModelMultiple.js
10   146 editMoldeWindow.js
11    67 fileEditor.js
12   291 main.js
13   158 models.js
14
15
16  2372 total
```

Listing 5.1: wc -l *.* in /content/JSWindows

We can observe we have got many small pieces of code no more than four hundred lines for each file. It gives us the opportunity to manage our code more comfortable. The total of these lines of code (more than 2000) gives us an idea of the size. We have to remark that all these files have been written in javascript language.

We will focus now in the xml code of the windows in our application:

```
1
2    171 ExperimentParameters.xul
3    105 MultipleProcess.xul
4     51 ParameterSingleSection.xul
5    140 Parameters.xul
6    124 Process.xul
7    191 SingleSectionEdit.xul
8    133 SingleSectionModels.xul
9     52 SingleSectionParameter.xul
10    24 XMLWindow.xul
11    78 editModelMultiple.xul
12    93 fileEditor.xul
13   153 main.xul
14   144 models.xul
15     7 sample.xul
16
17  1466 total
```

Listing 5.2: wc -l *.xul in /content

No more than 1500 lines of xml code to build all our application. We have to say that it has around thirteen windows. Advantages of building application using XUL is that we can not make mistakes in the XML code. For example in languages as Visual Basic, Java or c sharp we will need to create object elements to create buttons, windows and so on... With this way of creating interfaces the number of errors is drastically reduced.

5.1.2 Libraries

About our files composing the library we have got:

```
1
2    167 AddNewModels.js
3    576 CreateInputFile.js
4    120 CreateJavaData.js
5    392 Execute.js
6    311 FileUtilities.js
7    182 GenfitUtilities.js
8    261 Init.js
9      7 InputDirectoryProcess.js
10   217 OldXMLReader.js
11   136 Parameters.js
12   114 Process.js
13   325 WindowsEventsFunction.js
14   148 XMLUtilities.js
15   260 XMLWriter.js
16   121 countWords.js
17
18  3337 total
```

Listing 5.3: wc -l *.* in /content/JS

We have to say, to be honest, that all this files will not be reused for future applications because they are very dependent of the GENFIT structure. For example, GenfitUtilities.js has the classes that out application uses to describe the models of the proteins. However, the most of them are being reused in several parts of our software and the most of them will be reused in the future.

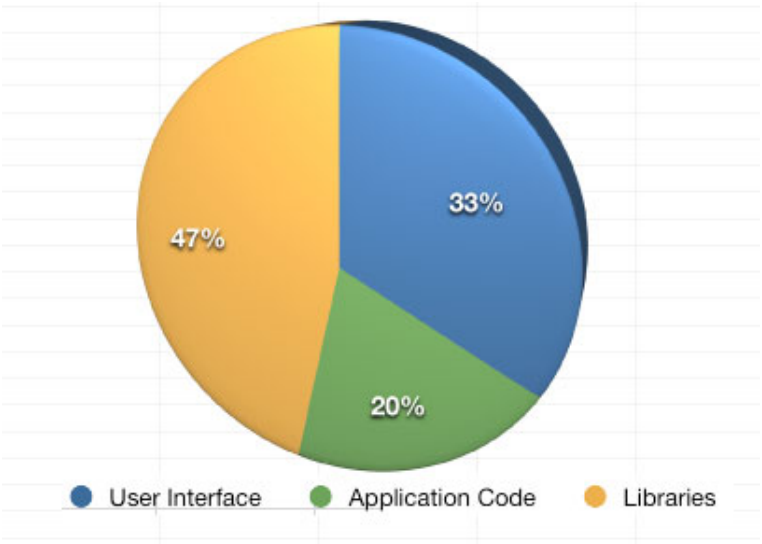
For example, FileUilities.js has methods and functions in charge of open, save and execute files as well as open new dialog file windows to open and save files and directories.

5.2 Reusability

In software engineering, reusability is the likelihood a piece of source code that can be used again to add new functionalities with small or no modification. In this way we can reduce

implementation time, increase the likelihood that prior testing and use has eliminated bugs and localizes code modifications when a change in implementation is required.

Next graphic shows us how the code in our application is distributed. That means for example that new developments will be able to reuse the code of the libraries. Then we think is a good signal of quality in the software that the most of the code can be reused.



Chapter 6

User's Guide

6.1 Installation

In this section we will explain different ways to install Genfit's user interface. This application has been conceived to be a very portable application and will can be run on linux, windows, mac and as Firefox extension.

6.1.1 On linux

If we want to execute Genfit user interface on linux we will need an addition software called XULRunner. XULRunner is a Mozilla runtime package that can be used to bootstrap XUL+XPCOM applications that are as rich as Firefox or Thunderbird. It will provide mechanisms for installing, upgrading and uninstalling these applications.

To make it easier this software has already been package together with genfit. Then end-users will not need to download again from internet.

Steps to install the application:

1. Download the package genfit.zip or genfit.tar.bz from the website.
2. Go to the folder where genfit has been uncompressed.
3. Execute the script called "start".

6.1.2 On Windows

If we want to execute Genfit user interface on linux we will need an addition software called XULRunner. XULRunner is a Mozilla runtime package that can be used to bootstrap XUL+XPCOM applications that are as rich as Firefox or Thunderbird. It will provide mechanisms for installing, upgrading and uninstalling these applications.

To make it easier this software has already been package together with genfit. Then end-users will not need to download again from internet.

Steps to install the application:

1. Download the package genfit.zip or genfit.tar.bz from the website.
2. Go to the folder where genfit has been uncompressed.
3. Execute the script called "start.bat".

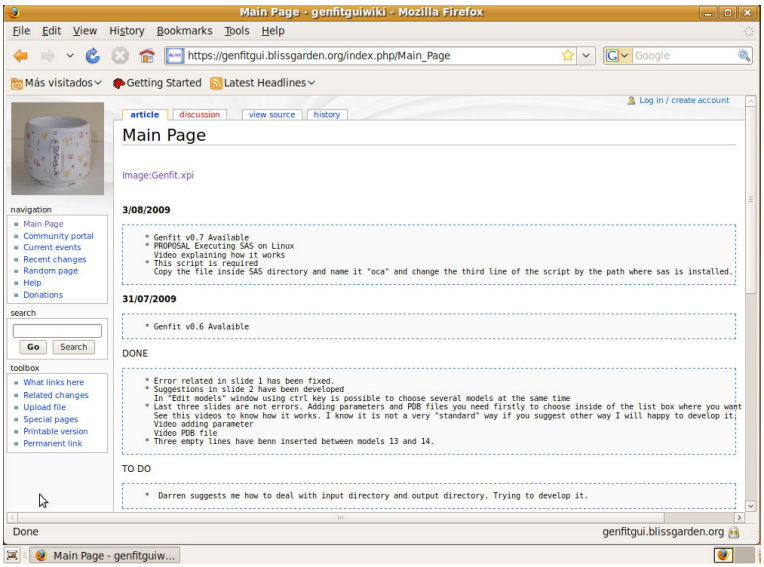
6.1.3 On Mozilla Firefox

Another interesting way to install Genfit user interface is inside of Mozilla Firefox. It makes possible to execute Genfit making an installation easier.

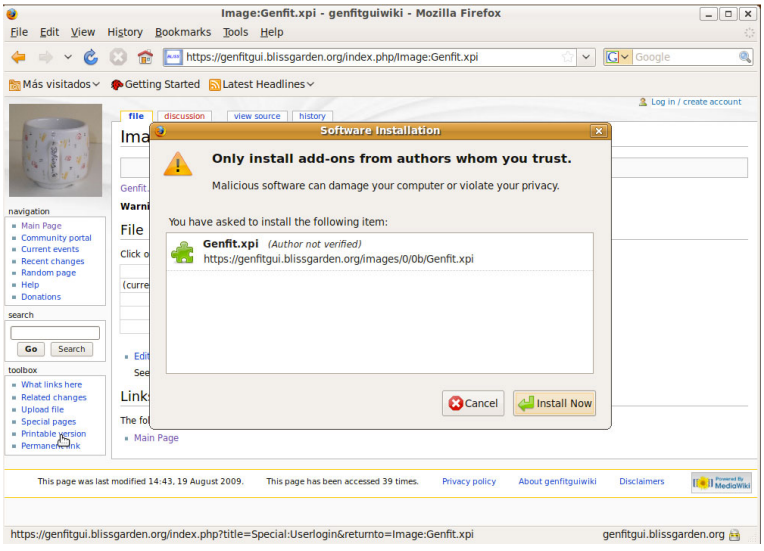
With this methods the number of dependencies is reduced and you just will need to have got the browser Firefox installed. File .xpi are package extension of firefox. When Mozilla firefox tries to open a file with this extension it will try to install it automatically. Then we will need to open this package from internet or from local files and it will be installed.

Steps to install the application:

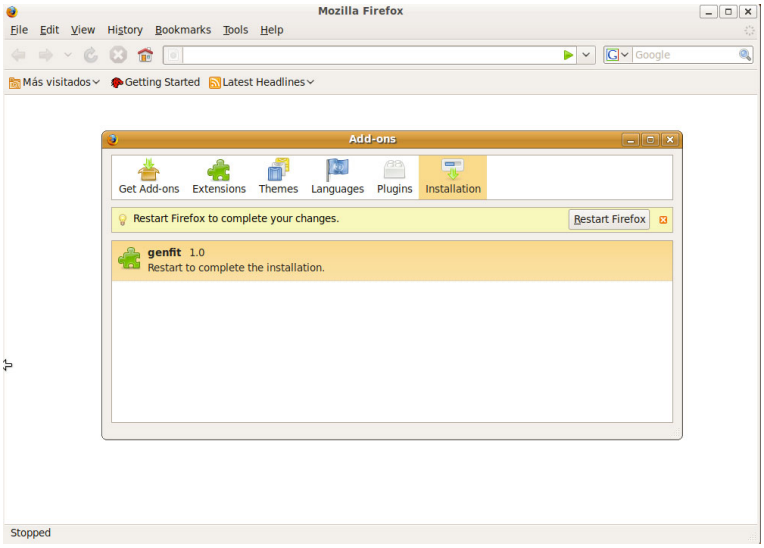
1. Open Mozilla Firefox.
2. Go to https://genfitgui.blissgarden.org



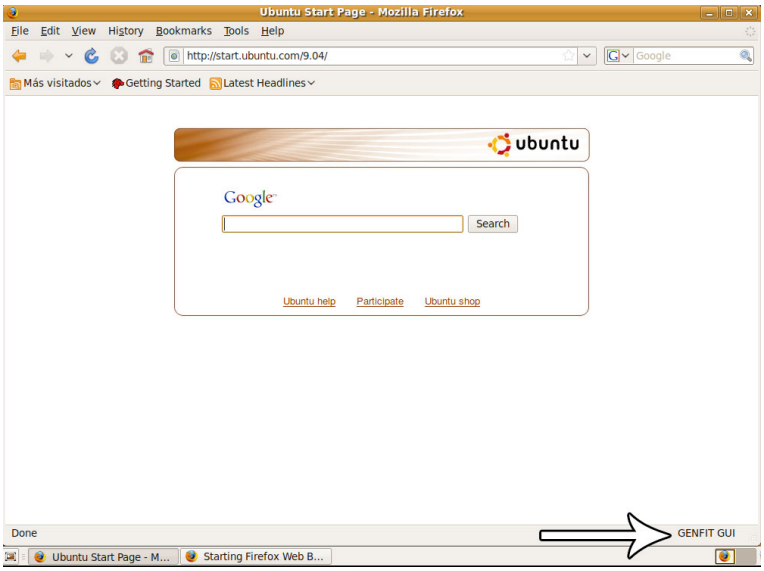
3. Click on Genfit.xpi and click "Install now" button to install



4. Restart Mozilla Firefox



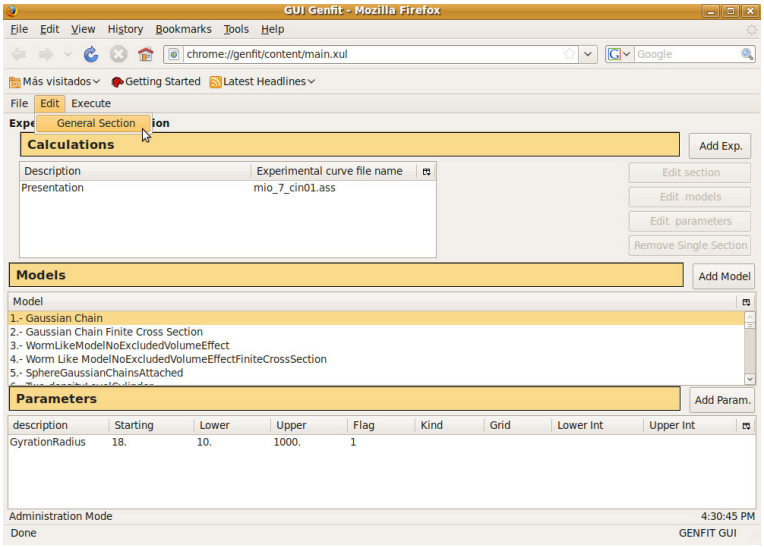
5. Check if application is already installed



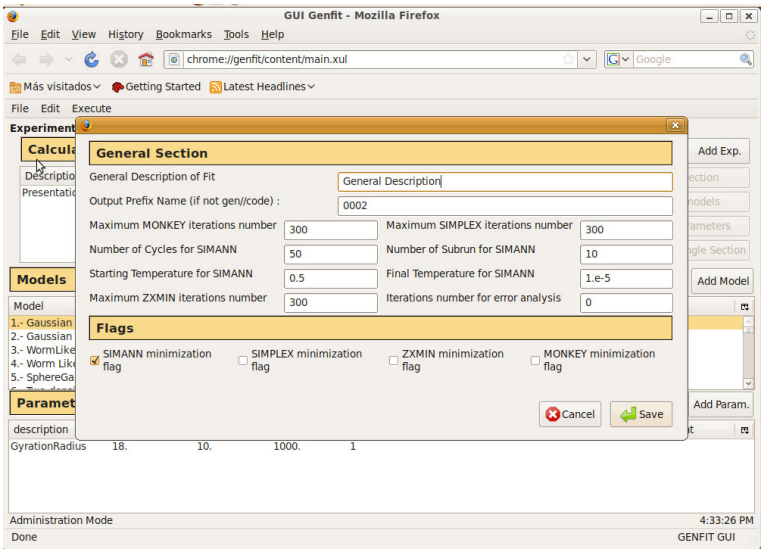
6.2 Modification of general parameters of the experiment

To modify general parameters of the experiment:

1. Go to the menu Edit
2. Click on General Section



3. Modify parameters and click "save" button to save.

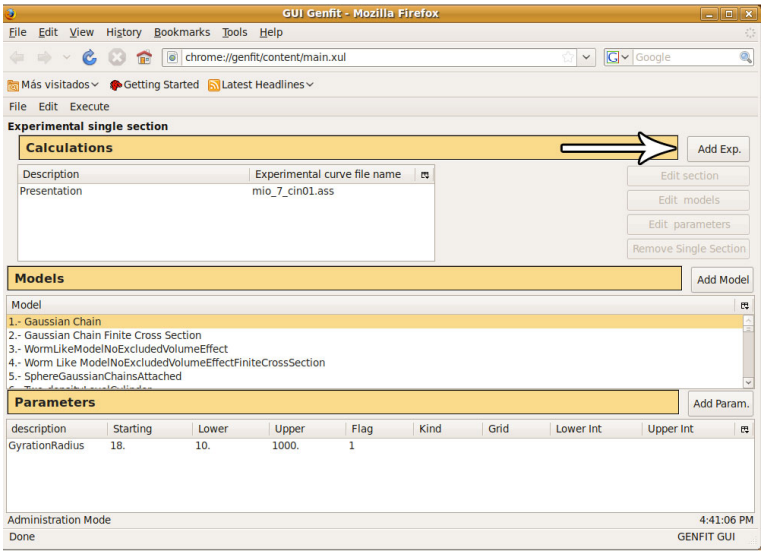


6.3 Calculations

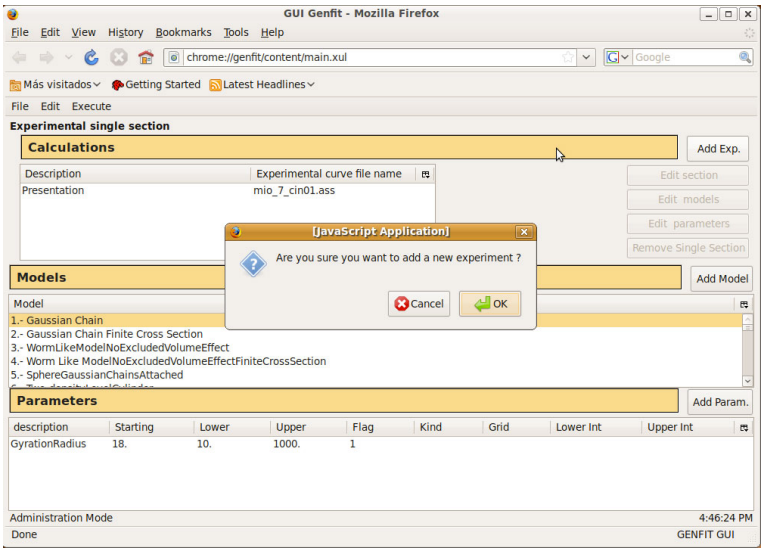
6.3.1 Adding a new calculation

To add a new calculation:

1. Go to the initial window and on Experimental single section click on button called "Add Exp."



2. After clicking application will ask if you are sure you want to add a new experiment. Answer yes/not depending of what you want to do.



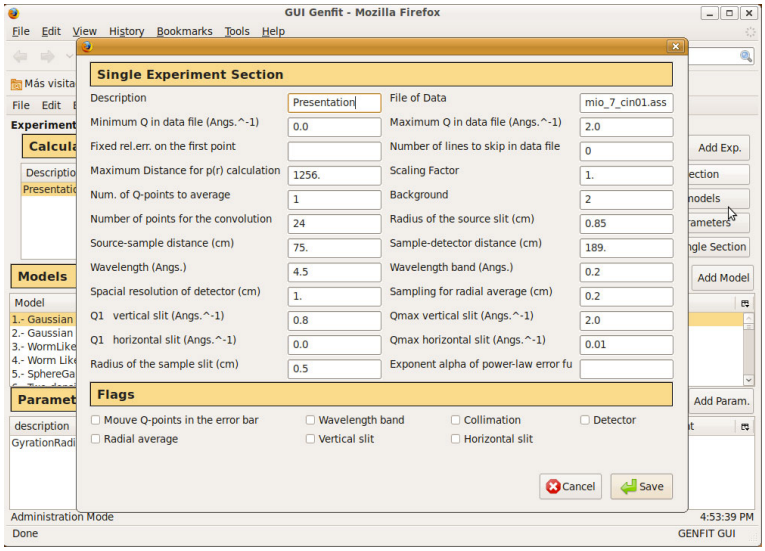
6.3.2 Static parameters

There are two ways for editing a calculation.

1. doing double click on the calculation.

2. Clicking on the button called ”Edit section”.

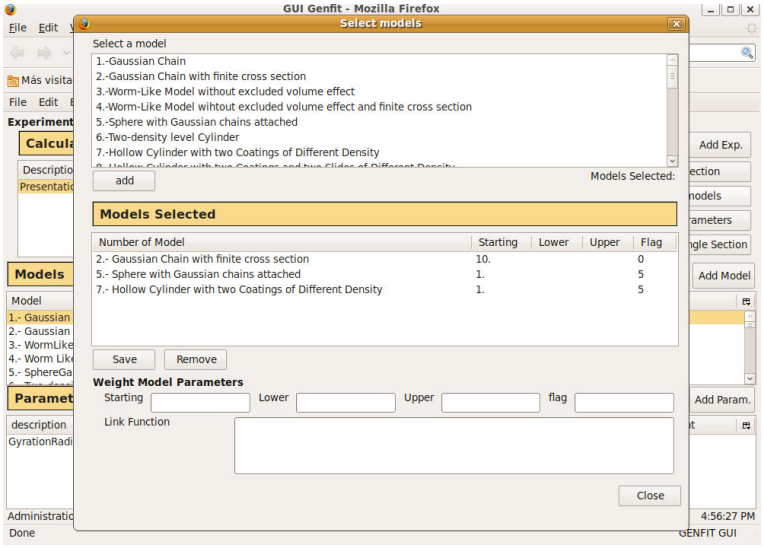
Once the windows has been open we edit the values and click ”Save” to save. We can see this action on the next picture.



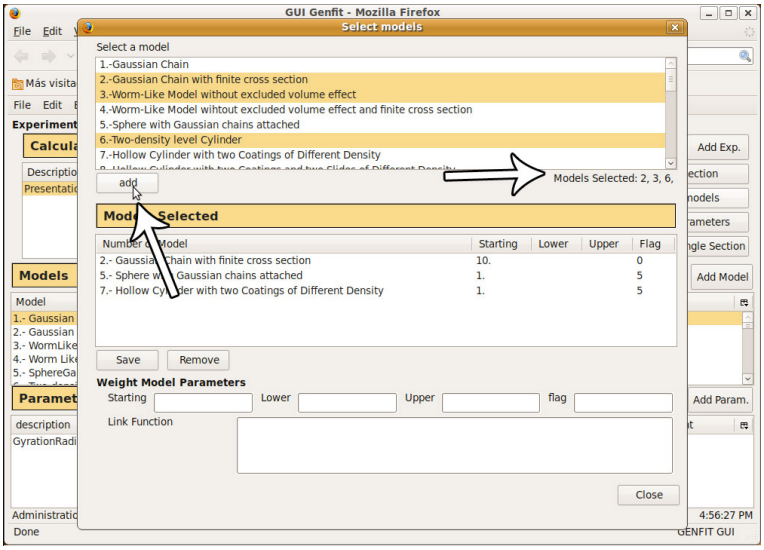
6.3.3 Models

We can edit models of the calculation.This means we can add new models, remove models and change parameters of the models.

1. Select the calculation we want to modify and after by clicking "Edit models" button the window will appear.

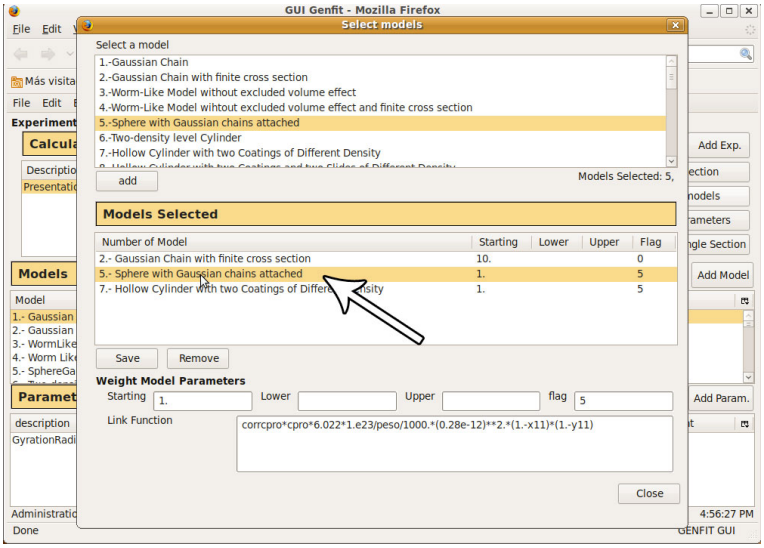


2. Click on the text box called "Select a model" to select a model to insert into the calculation. e can select several models using control key. After selecting all models click on button "Add".

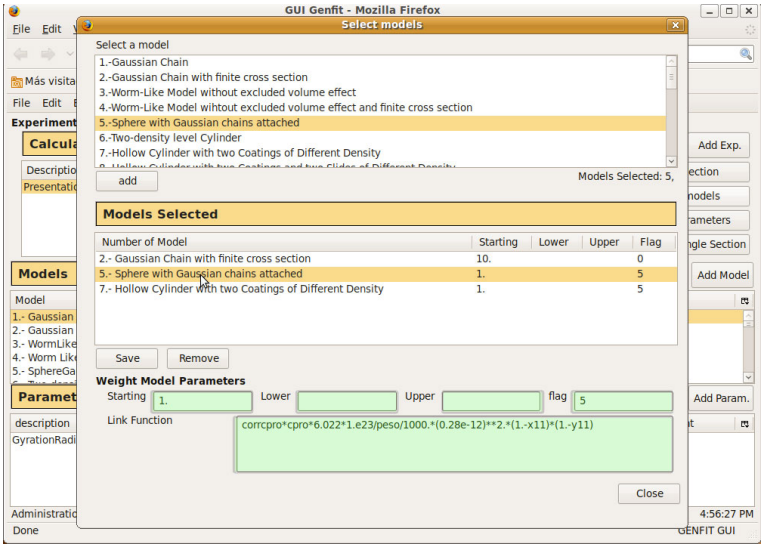


Once we have selected models that take part of the calculation we will need to change the values of the experiment.

1. Select the model inside of the second text box in the middle of the window.



2. Modify all the weight parameters (in green) of the experiment.

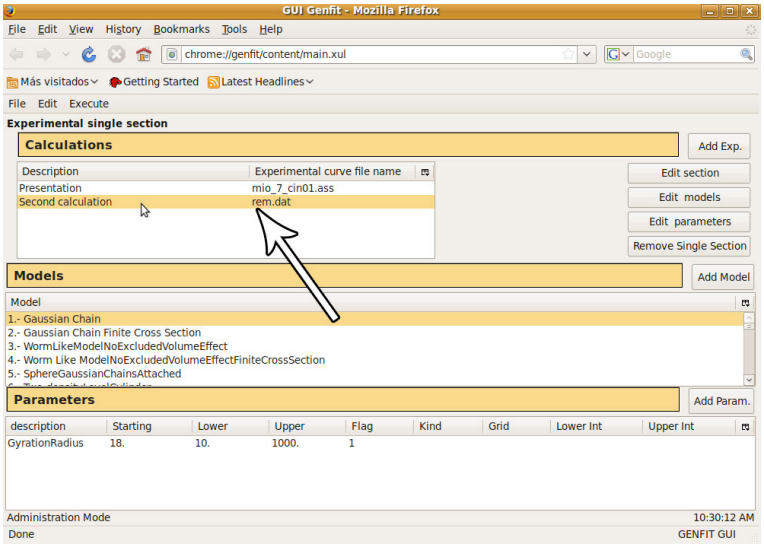


3. Click ”save” button to save all changes.

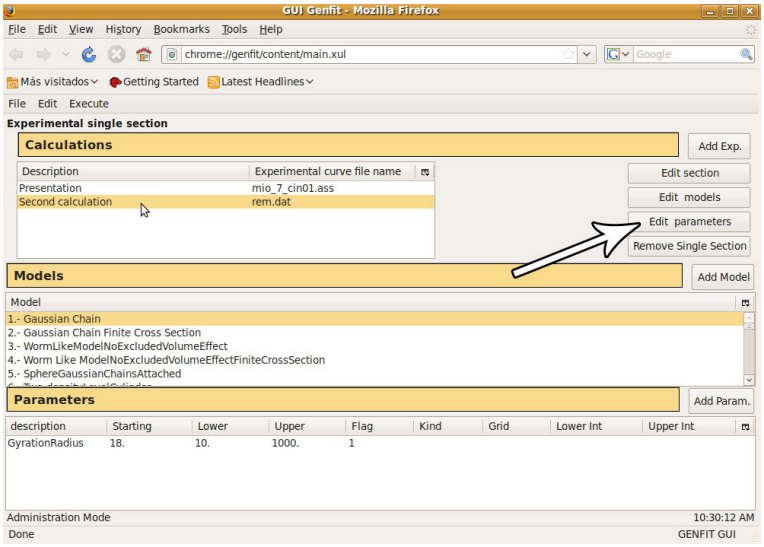
6.3.4 Dynamic parameters

We told there are some dynamic parameters for each calculation which are introduced by users.

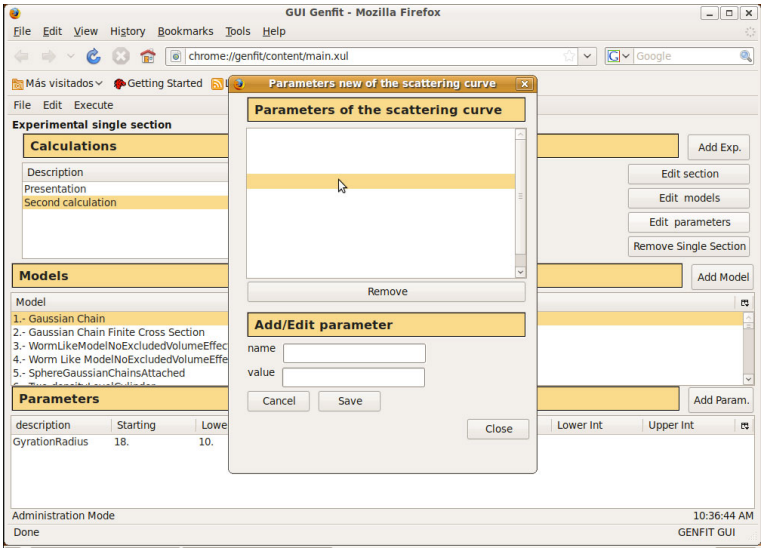
1. Select the calculation we want to modify.



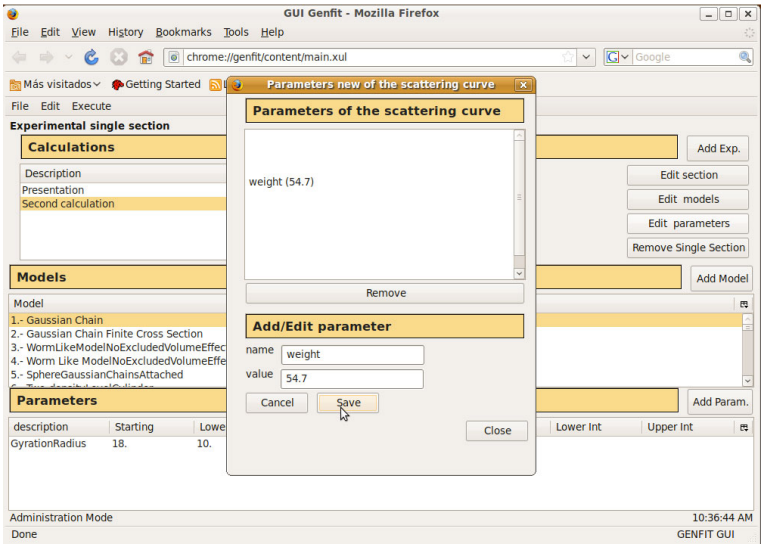
2. Click on "Edit models" button the window will appear.



3. Once window has been open. We will have got a list box at the top side. By selecting a line we will can add a new parameter. We can select whatever line we want. For example, if we want to add a new parameters in the position number four, we will select the line number four. In the example:



4. After selecting the line. At the bottom part of the window we have got two text boxes (name and value). We will write the name and values there and will save.

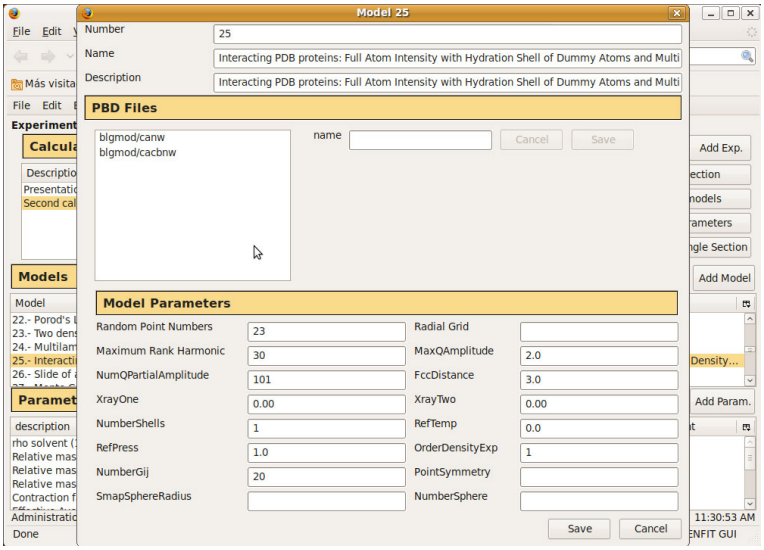


6.4 Models of the experiment

Only administrators can modify these models but users are able to read. At the middle of the main window we have got a tree box with a list with all the models. Next sections will explain how to switch from administration mode to user mode.

6.4.1 Edit a model

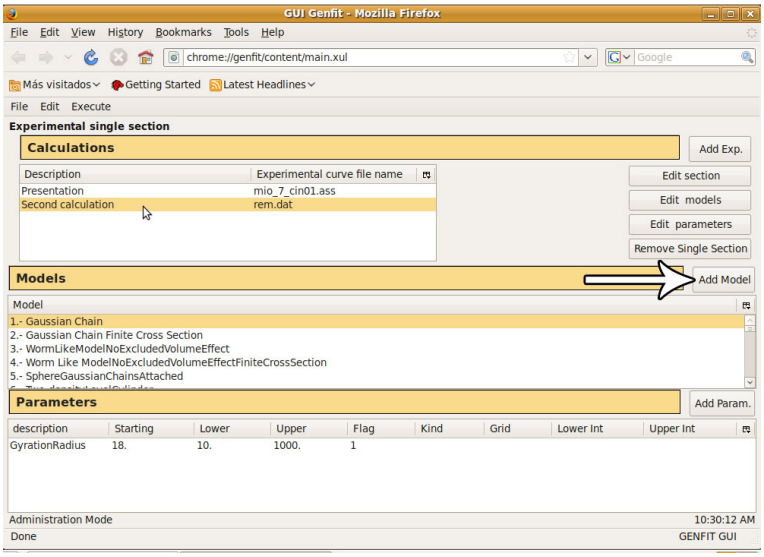
1. Click on the model you would like to display and then double click on it.
2. A window with all the information about the model will be shown.



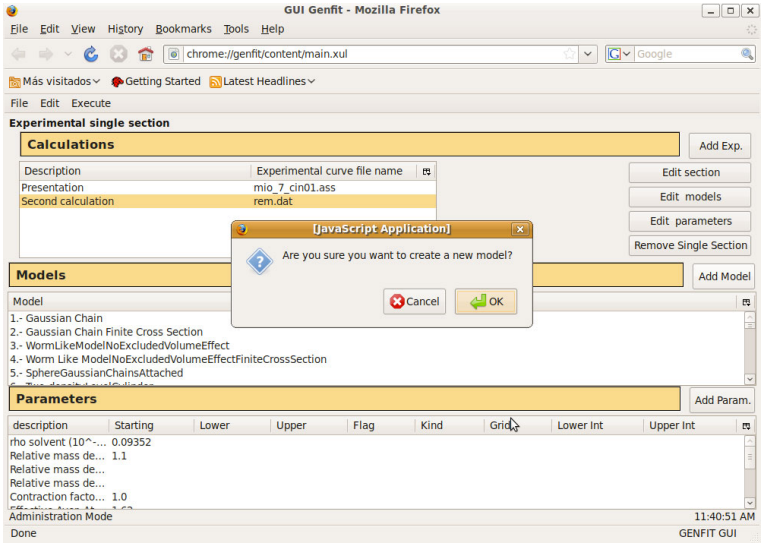
3. If you are in administration mode you can modify the model and click save. In other case you can just click cancel to come back to the main window.

6.4.2 Add a model

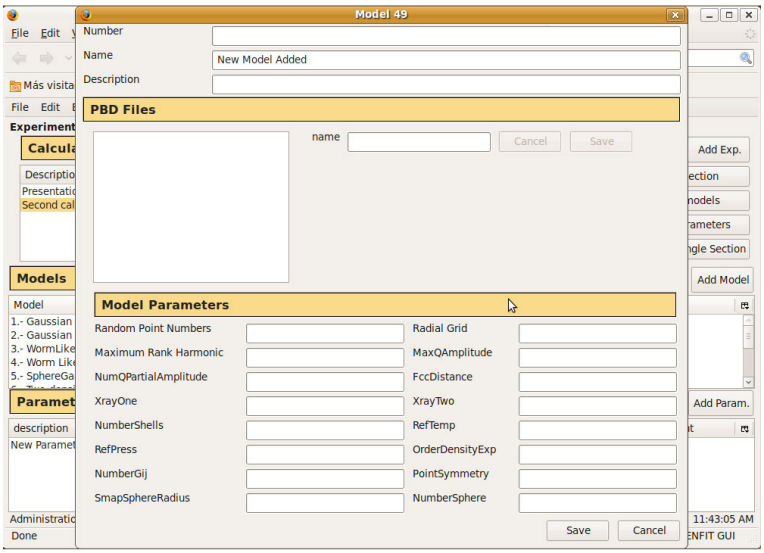
1. Click on the "Add model" button.



2. A window asking if you are sure you want to add a new model will appear.



3. Once you click "Ok" a new model has been inserted at the end of the list and a new window editing this model is shown.



6.5 Parameters of the model

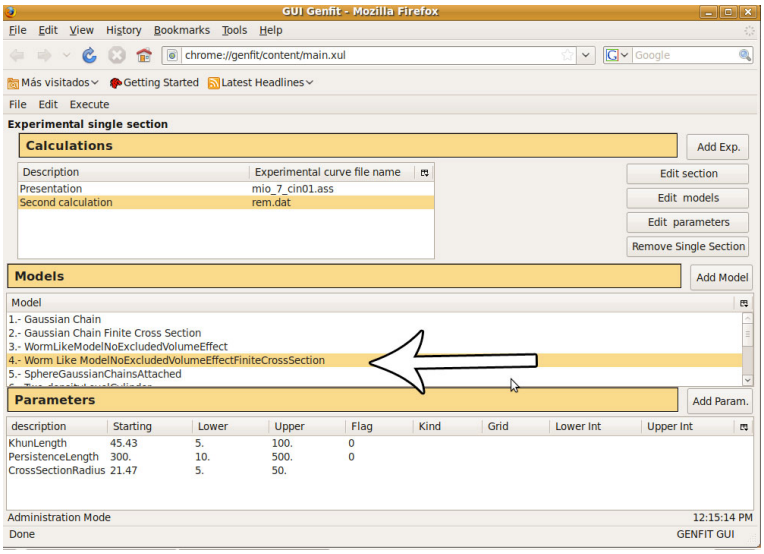
We know that for each model we can have got one or more than one parameters. In this section we will explain how to modify this parameters and in case of administrator mode how to add new parameters.

6.5.1 Edit a parameter

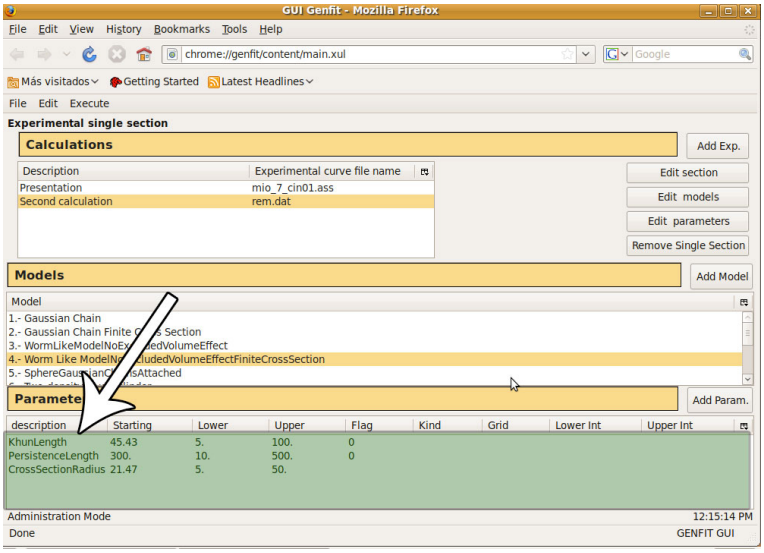
1. Double click on the parameter to be edited.
2. A new window with the parameter appears. Change the values as you want.
3. Click "Save" button to save.

6.5.2 Add a parameter

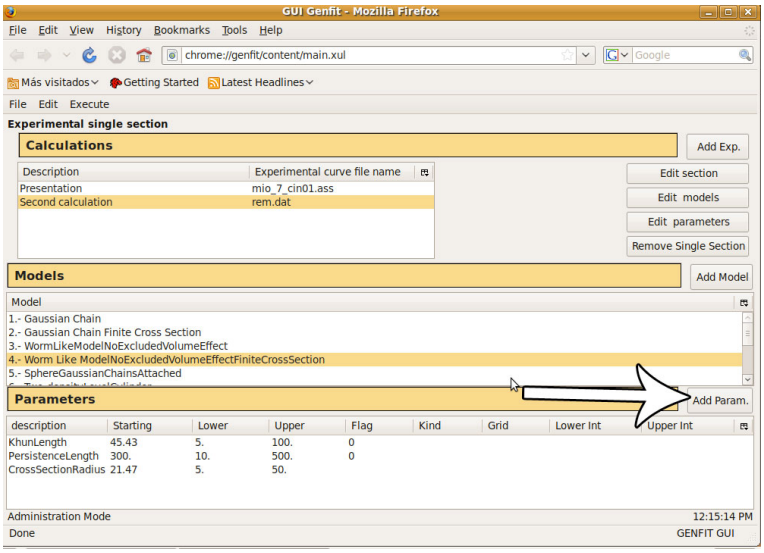
1. Choose the model which you want to insert a new parameter



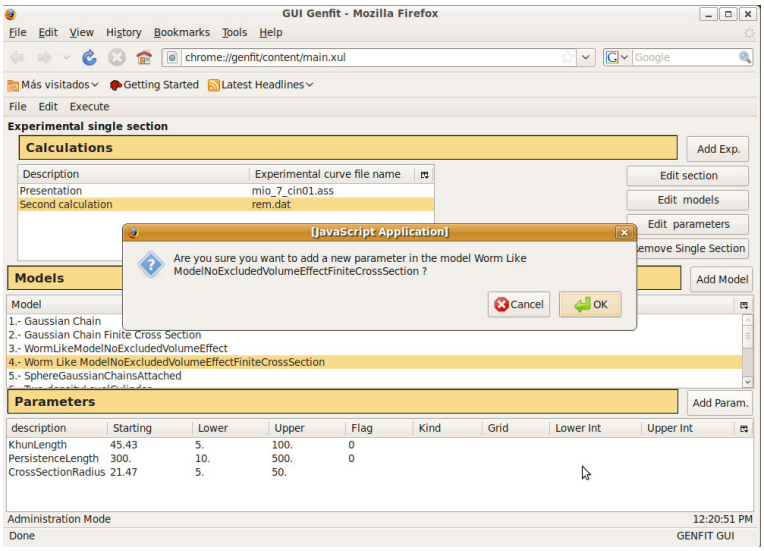
2. A list with parameters at the bottom part of the main window will display all the parameters of this model.



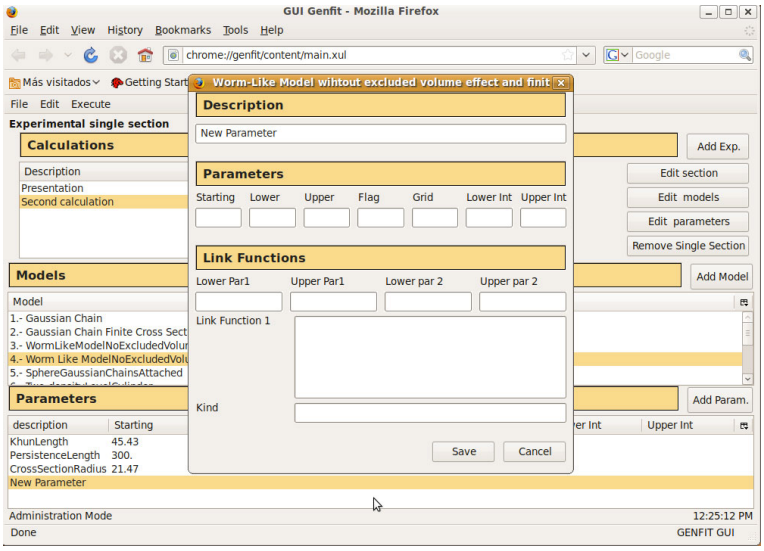
3. Click on "Add Parameter" button



4. A window asking if you are sure you want to add a new parameter will appear.



5. By clicking Ok a new parameter will be added and the edit window will be shown.



6.6 Executing SAXS from Genfit GUI

This section describes the steps required to execute SAXS application from the GUI.

6.6.1 Overview

Basically GUI will create a temporal folder inside the sas folder and will copy the input files. After GUI will call to Gallo for executing the experiment. Once the experiment has been finished all generated files will be copy to the output directory user has configure on the executing window in the application.

6.7 Executing a single experiment

In this modality user will execute a single experiment. All the parameters of the experiment loaded in the application will be executed. User will need to establish next paths:

1. Choose the input folder where the input files will be copied

2. Automatically a "Processed" folder is suggested for the application inside of the input directory. Anyway user can will change this path to another one where the output files of the SAXS application will be moved there after of the execution.

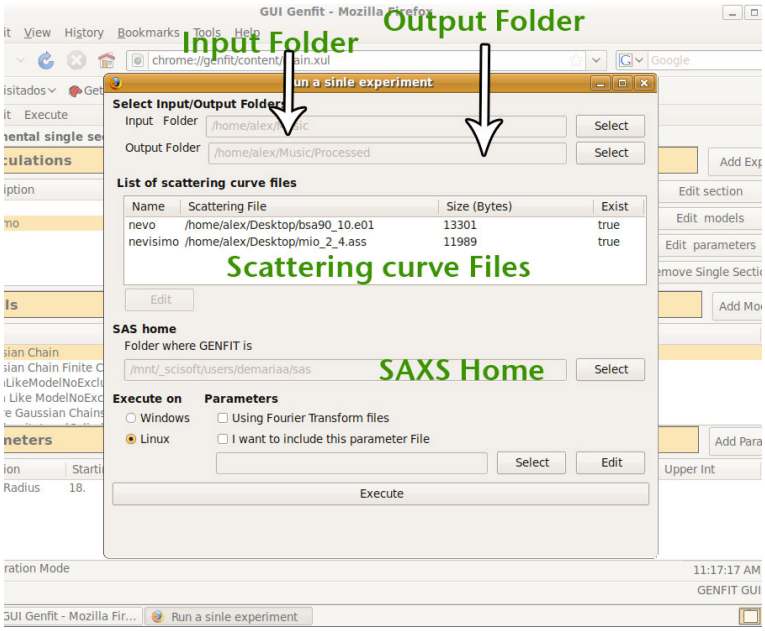
3. We can edit the files of the scattering curve, we can modify if we consider necessary or we can read the values. As well as we can see easily if the file exists and which is the size of the file.

4. We will need to tell to the application what is the directory of the SAXS application. It can be a local path or a network path. This is very interesting because we don't need any more to have got the SAXS application stored in our local hard disk. However if the application access is remotely we will need to give it permissions enough to be executed

5. We need to tell if we are going to execute SAXS in windows or in linux. Depending of our choose GUI will call to "oca" or to "gallo.bat"

6. We will define as well as paramteres as if we are using Fourier Transform and if we will use a parameter file. In case of use a parameter file we will need to give it the path of the file. Edit option is available too.

Here we can see a screenshot of the execute single experiment window.



Chapter 7

Acknowledgments

At the ESRF, sincere and genuine thanks to Darren Spruce, who was my internship tutor, for his computing experience, his how-to-do and specially for his always positive vision toward this project in particular and life in general. This work would not be possible without Francesco Spinozzi, who was the author of Genfit, a very valuable application for modeling proteins in solution, for his time even on holidays and his lovely skype meetings. To Claudio Ferrero for his complete attention and enthusiasm.

At the UJF I am very grateful to Jean-Marc Thiriet who help us a lot during all our erasmus period and in general to the Universidad politecnica de Valencia and Institut Universitaire de Technologie to make this wonderful experience possible.

My personal acknowledgment to all my close friends in Spain and Grenoble where we are sharing mountains, snow and good moments. Specially to Adri, Jorge and Mario for their motivation in our climbing training. To Tere, Blanca, Karmele, Antonito, Marga, Claude and Angela for their never-ending support at the freres Berthom. To Valentina, Ben, Paz and Fer for their help in our first steps in France and their trusted friendship. To Mum and Dad who are constantly supporting me even in difficult moments. And finally huge thanks to Ana for sharing with me every single moment and making me a rich man.