UNIVERSIDAD AUTÓNOMA DE MADRID

Scipion: a software framework toward integration, reproducibility and validation in 3D Electron Microscopy

by

José Miguel de la Rosa Trevín

A thesis submitted in partial fulfillment for the degree of
Doctor en Ingeniería Informática

in the
Escuela Politécnica Superior

September 2017

Declaration of Authorship

I, José Miguel de la Rosa Trevín, declare that this thesis titled, 'Scipion: a software framework toward integration, reproducibility and validation in 3D Electron Microscopy' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better."

- Edsger W. Dijkstra

UNIVERSIDAD AUTÓNOMA DE MADRID

Abstract

Escuela Politécnica Superior

Doctor en Ingeniería Informática

by José Miguel de la Rosa Trevín

In the past few years, 3D electron microscopy (3DEM) has undergone a revolution in instrumentation and methodology. One of the central players in this wide-reaching change is the continuous development of image processing software. Here we present Scipion, a software framework for integrating several 3DEM software packages through a workflow-based approach. Scipion allows the execution of reusable, standardized, traceable and reproducible image-processing protocols. These protocols incorporate tools from different programs while providing full interoperability among them. Scipion is an open-source project that can be downloaded from `http://scipion.cnb.csic.es`.

Acknowledgements

Aunque el resto de esta tesis está escrito en inglés, en esta sección tengo que usar el español para poder expresar mejor mis agradecimientos a todas aquellas personas involucradas directa o indirectamente.

Tengo que empezar agradeciendo con todo mi corazón a mi familia (la vieja): mis padres, mi hermana, tíos, primos y parientes lejanos. Todos ellos, en mayor o menor medida, han contribuido a que llegara hasta aquí. Me han apoyado siempre, con mucho sacrificio, durante todo el camino y su amor incondicional ha sido una gran fuente de energía para seguir adelante.

Por otro lado está la familia nueva: ese gran pequeño tesoro que es mi hija Amelia. Aunque ahora no pueda leer estas líneas, algún día entenderá cuan importante ha sido su alegría y su cariño. También Airen, mi incondicional compañera en este viaje que es la vida, estando siempre ahí para compartir alegrías y tristezas.

Siguiendo con la familia, quiero agradecer a mi hermano Josué, sin el cual ciertamente no estaría escribiendo estas líneas. Aunque en su día no quiso copiarme Budokan, creo que ya es hora de cerrar ese capítulo, como este doctorado.

Quiero agradecer especialmente a José María, por confiar en mi desde el principio y darme la oportunidad de venir al B13 e involucrarme en el fascinante mundo de la Microscopía Electrónica. También quiero agradecer a Roberto, Carlos y Sjors, que desde los primeros días me brindaron su ayuda incondicional y de los cuales he aprendido muchísimo durantes todos estos años.

La parte científica y profesional también se ha complementado con la parte humana y de colaboración. También mis agradecimientos van para muchos compañeros de lucha: Kino, Analu, Jesús, Adrián, Laura, Javi, Nacho, Vahid, Roberto, Jordi, Pablo y muchos otros. Mi agradeciemiento también para Blanquita, que siempre ha sido un gran apoyo profesional y personal.

Es imposible nombrar a todos en una o dos páginas, pero quiero agradecer de forma general a todas aquellas personas e instituciones que han hecho posible, de una forma o de otra, la realización de este trabajo.

Contents

List of Figures

List of Tables

Abbreviations

3DEM	Three Dimensional Electron Microscopy
API	Application Programming Interface
BCU	BioComputing Unit
CNB	Centro Nacional de Biotecnología
CTF	Contrast Transfer Function
cryo-EM	Cryogenic Electron Microscopy
DDD	Direct Detector Device
EMDB	Electron Microscopy Data Bank
GUI	Graphical User Interface
GPU	Graphical Processing Unit
I2PC	Instruct Image Processing Center
ML	Maximum Likelihood
MPI	Message Passing Interface
NMR	Nuclear Magnetic Resonance
ORM	Object-Relational Mapping
RAM	Random Access Memory
SPA	Single Particle Analysis
XML	eXtensible Markup Language

A mi familia, la vieja y la nueva.

Chapter 1

Introduction

Structural biology aims at determining the three-dimensional structure of proteins to understand their function and inter-relations. In the cell, many processes are carried out by nano-machines, which are macromolecular complexes composed by multiple proteins. Just like daily-life machines, these nano-machines employ deformations and movements of separate parts in their functioning. Due to the inherent flexibility of these complexes, their structural characterization is difficult because the different biophysical techniques tend to average all the conformational states. However, during the last decades, cryo-electron microscopy (cryo-EM), the structural analysis of samples embedded in vitreous ice, has emerged as a promising technique for studying the structure of biomolecules [1].

X-ray crystallography is one of the most used techniques to determine the 3D structure of proteins [2]. If the specimen under study can be crystallized, this technique can achieve atomic resolution. Nuclear magnetic resonance (NMR) may provide unique information about dynamics and interactions, but atomic structure determination is restricted to small complexes. Both techniques typically require large amounts of relatively pure sample. Therefore, structures of large complexes are very difficult to solve by X-ray crystallography or NMR, especially when they are available only in small quantities. cryo-EM provides the potential to visualize such complexes and in some cases, it can be combined with results from these other techniques [3].

For many years, structure determination of biological macromolecules by cryo-EM was limited to relatively large complexes at medium-resolution, but in the last few years we have witnessed a real revolution in the 3DEM field, mainly due to great improvements in equipment, computing power and software tools. The introduction of direct detection devices (DDDs) has made a fundamental difference in image acquisition quality, enhancing the resolution achieved by previous image-recording media, such as photographic films or charge-coupled devices. Moreover, computer power has increased significantly through

the use of multi-core machines, clusters, graphics cards and even the use of cloud computing. All these combined developments have allowed more computationally intensive methods, larger datasets and more challenging biological questions to be posed.

The evolution of cryo-EM has been intrinsically related to the development of scientific software for image processing. The number of programs available for the community have notably increased over the years. Nevertheless, software development for 3DEM has been focused in the scientific side neglecting in many cases good basic software-engineering practices. In this thesis I address essential software problems in the field including: interoperability, traceability, reproducibility, data management, automation and distributed computing. This work describes the design and implementation of a new software framework, Scipion, that models image processing in electron microscopy allowing the integration of several 3DEM software packages under a unified interface for both biologists and developers.

The next section of this chapter summarizes the standard workflow used in single particles analysis. After that, I briefly introduce the problem of software interoperability in cryo-3DEM, followed by the problems of reproducibility and automation. Chapter 2 states the objectives of this work, while Chapter 3 presents a review of the relevant literature, classified in two groups: (1) the evolution of software packages in the 3DEM community and, (2) general workflow systems in bioinformatics and related fields. Chapter 4 goes over the details of the design and implementation of the system. Results and current impact in the field are presented in Chapter 5. Finally, in Chapter 6 future work and general conclusions are discussed.

1.1 Single-Particles Image Processing Workflow

Although there are many possible workflows leading to 3D reconstructions of a biological specimen, in the following I describe a particular one that illustrates common processing steps and shows the processing complexity.

Before the introduction of the DDDs, a project usually started by recording micrographs in an electron microscope. Nowadays, DDDs allow to record in "movie mode", where the electron dose is distributed into several frames of a movie. Therefore, now it is possible to correct the image blurring due to the beam-induced movement. There are many programs that align movie frames to produce an averaged micrograph [4–7]. In this early step, users need to select from many available programs to deal with different parameters and conventions.

After obtaining a set of micrographs, the next step is the estimation of the contrast transfer function (CTF), that can be seen as the Fourier transform of the point spread function of the microscope. The CTF is an oscillatory function in the frequency domain that affects both the amplitudes and phases of the images. Accurate estimation of the CTF parameters is critical to restore the original information in order to obtain a reliable 3D reconstruction. There are several programs available for CTF estimation [8–12]. At this point, users can screen the set of micrographs based on the resulting CTF parameters. Bad micrographs can be discarded and not included in further steps. Micrographs can also be downsampled to improve the signal-to-noise ratio and accelerate subsequent calculations.

After having a good set of micrographs, the project continues with the selection of individual particles, either manually or automatically. This task is very time-consuming and labor-intensive due to the importance of choosing good particles for the subsequent analysis. Including too many poor particles could detriment final results. Manual particle selection is often used when there is not *a priori* information of the protein under study and the distribution of the projection views. At this stage, a human could perform better than the computer, at the risk of having a bias toward views that are more recognizable and omitting minority views. In the semi-automatic approach, there is a two way interaction: the computer detects possible particle candidates and the user adjusts the parameters or correct the initial candidate guesses [13, 14]. Finally, there are algorithms that automatically detect particles from the micrographs [14–19]. Nonetheless, due to the low signal-to-noise ratio, results from automatic picking usually need to be reviewed since many artifacts are still included.

Particle images are extracted from the micrographs at given locations (or coordinates). Some preprocessing may be applied while extracting, such as filters, contrast inversion and others. Particles are usually sorted according to a quality factor to identify possible outliers, like wrongly picked images. At this point, the gallery of particles may be used as input for 2D classification algorithms, so as to detect possible heterogeneity due to sample contamination, different conformations or different specimens, followed by the calculation of a low resolution initial map. Some images may be discarded and not considered in subsequent steps.

There are several approaches to produce an initial low resolution 3D map from 2D class averages [20–26]. Programs used in this step usually produce a collection of possible 3D maps that are visually inspected. After one of the initial volumes is selected, an iterative refinement algorithm will carefully assign projection directions to each of the input images. Some of these refinement algorithms can also be used for dealing with heterogeneity, by comparing several initial 3D references with the gallery of particles.

Identifying and analyzing 3D heterogeneity is, however, still technically challenging. Regarding the classification of images into homogeneous data sets -that is, grouping together images produced by projecting a specific conformation of the specimen under study- many alternatives have been proposed [27]. The most popular ones is based on Maximum Likelihood (ML) or Maximum *a posteriori* (MAP) [28–30]. ML methods aim at finding the 3D references so that the likelihood that a given 3D reference would produce a given experimental data set is maximized. When more than one 3D reference is used, each experimental image has a given probability of being produced by each of the available references in each of the different projection directions.

By far, the most frequently used method to measure reconstruction quality is the Fourier shell correlation (FSC) curve, which provides information on the level of the SNR as a function of the spatial frequency [31], and the resolution of the map. The FSC is obtained by computing correlation coefficients within resolution shells from the Fourier transform of two volumes. One important aspect in the FSC definition is the assumption of the noise independence in the two maps. This condition is difficult to meet in practice and it is often compromised by refining a single dataset while evaluating the FSC with two volumes computed from half-subsets of the dataset. In 3DEM, the "resolution" is a somewhat arbitrarily chosen cut-off level of the SNR or FSC curve [31–34].

One of the main drawbacks of the FSC is that it does not explicitly test for the validity of the reconstructions and that some processing steps improve the nominal value of the resolution without improving the alignment parameters [35]. Another common issue in currently used refinement procedures is their tendency to overfit the data [36], where some features emerge in the map due to the alignment of noise. Current approaches to distinguishing "signal" from "noise" are based on some form of cross-validation, where data not used for the refinement serve to validate the results. Despite the validation of the refinement and its quality are still open problems in the field, there are many approaches that tackle these problems in one way or another: (1) resolution limitation imposed on projection-matching [37]; (2) processing different subsets independently ("gold standard") [38]; (3) tilt-pairs validation [39]; (4) high-resolution noise substitution [40]; (5) check consistency between alignment of images and pure noise [41]; (6) evaluate the alignment consistency between a set of projection images with respect to a given 3D density map [42].

1.2 Software Packages and Interoperability

The evolution of the 3DEM field has been closely related to image processing and software development from the early days of this field. In 1992, seven software packages, that were in use at that time, were reviewed in [43]. A few years later, in 1996, a special issue of the Journal of Structural Biology was dedicated to software tools for molecular microscopy [44]. At that time, the existing 3DEM related software were classified into four groups: general packages, specific packages, applications tools and visualization tools. The same organization is maintained today on the Wikipedia page https://en.wikibooks.org/wiki/Software_Tools_For_Molecular_Microscopy with the added category of "Utilities". At the moment of writing this thesis, the website contained 17 general packages, 24 specific packages, 37 application tools, 19 visualization tools and 6 utilities.

Only considering the packages focused in single-particles analysis, there is a myriad of tools available to the community. They go from command line programs to complete software suites (or packages), where the whole processing workflow can be done. Without creating an exhaustive list, following are some them: Appion [45], Bsoft [46], CTFFIND [8], EMAN2 [47], FREALIGN [27], IMAGIC [48], 2dx [49], RELION [29, 50], SIMPLE [24, 51], SPARX [52], SPIDER [53], and Xmipp [54].

Since the field is expected to continue growing, so will be the list of available tools and packages. In Hegerl [43] it was raised the question as to whether there was a need for seven different packages (at that time) and whether would be more practical to consolidate the existing packages into a modular and adaptable system supported by the entire community. In the review of [55], the authors points the lack of control over someone-else code as one possible disincentive to using code from other labs. Despite the efforts to make standard packages "easy to modify and adapt", the final results are usually black-box libraries that are not easily re-usable.

Each of the current packages has its own strengths and weaknesses and no single package performs the best for all situations. Indeed, in most projects, researchers tend to combine tools from different software packages to achieve a certain processing pipeline. But this is a tedious and error-prone process, considering all the possibilities in file formats, systems of coordinates conventions, different graphical interfaces and all the book-keeping required.

Achieving a smooth interoperability between different EM software packages has been one of the main motivations for developing the Scipion framework. Our goal is to overcome practical problems when combining several tools from different packages in the same processing pipeline, shifting the responsibility from the users to the developers.

This goal has dictated some of the important design decisions during the development of this project. Even if the main focus of Scipion is interoperability, we hope this work will serve to change the current status of software development in the field by working in more collaborative ways and paying more attention to software architecture.

1.3 Provenance: Traceability and Reproducibility

Provenance is a critical concept in scientific workflows, since it allows scientists to understand the origin of their results, to repeat their experiments, and to validate the processes that were used to derive data products.

According to Wikipedia, "*Reproducibility is the ability of an entire experiment or study to be duplicated, either by the same researcher or by someone else working independently*". The scientific method heavily relies on reproducibility, rigour, transparency and independent verification. Of course, a non-reproducible result is not necessarily wrong, neither a reproducible one is right. Nevertheless, conducting transparent and rigorous research studies will move science forward. A general concern has significantly increased among the scientific community due to the difficulty in reproducing published studies. If this problem is not properly addressed, it will strongly damage the public trust and support for scientific research [56].

The importance of replication and reproducibility has recently been exemplified through studies showing that scientific papers commonly leave out experimental details essential for reproduction. For example, psychology researchers showed in [57] that 246 out of 394 contacted authors of papers in American Psychology Association journals did not share their data upon request (62%). Another example was a study published in Nature in 2012 that reviewed a decade of cancer research [58]. The authors found that 47 out of 53 medical research papers were irreproducible. The inappropriate statistical analysis and the unavailability of all the data were two common features among irreproducible studies.

The cryo-EM field also suffer from reproducibility (and validation) problems. Between 2002 and 2005 five independent structures were reported for the same complex, the 1.3 MDa inositol phosphate receptor. Two of the structures were determined in negative stain [59, 60] and three in amorphous ice [61–63]. Although the differences between the maps may be partly explained by differences in biochemical preparation, they are more likely due to errors in the structure determination [64]. A later cryo-EM study at 10Å[65] was in agreement with one of the negative stained maps [60] while being substantially different to the earlier cryo-EM structures. This case shows the lack of validation tools

to prove the correctness (or not) of a given result. A more recent study, which reported the structure of the HIV glycoprotein (HIVGP), has generated a lot of controversy in the 3DEM community when scientists questioned the methodological procedures followed to obtain the structure [66–70]

The work presented in this thesis is strongly influenced by these reproducibility concerns, with the aim to provide a framework for encouraging repetition and validation of previous experiments. Of course, having the software tools alone is not enough to develop a culture of reproducibility. Nonetheless, we hope this framework could notably facilitate the execution and reuse of EM processing workflows in a routine basis as a first step towards reproducibility.

1.4 Automation and Distributed Computing

Computing has changed how science is done nowadays, enabling scientific breakthroughs through new kinds of experiments that would have been impossible only a decade ago. Increasing computer power has made possible to address more complicated problems through the application of more sophisticated algorithms and the manipulation of large amounts of data.

In molecular biology, the development of high-throughput technologies have allowed new discoveries through the analysis of a huge volume of experimental data. Increasing improvement of those technologies are providing new insights about the genome, transcriptome and proteome, among others. Several disciplines have become very data-intensive (apart from computing-intensive) and have posed the computational challenge of managing, processing and analyzing all this vast sea of information.

Computing in cryo-EM is not the exception, existing programs in the field need to perform intensive computations. Moreover, data storage and management is becoming a serious problem that needs attention. The pipeline in cryo-EM involves several steps that are very heterogeneous in terms of the required computing resources. For example, some tasks are interactive, which are better suited to be run locally. Others require hundreds of processors to finish in a reasonable time, while others require more RAM memory. This scenario makes very difficult to complete the whole project in the same computer, which also introduces the problems of data transfer between different locations and the tracking of all the steps performed.

With the advent of the DDDs and other instrumentation improvements for 3DEM, data collection is moving toward a high-throughput technology. This increase in data acquisition speed and quality is revolutionizing the field, but, at the same time, it is

creating a bottleneck in the pipeline for storing and processing the image data. Just a few years ago, before the wide use of DDDs, a typical session recording images of 4k×4k pixels, 12 bit unsigned integers in MRC format would take around 144 Gb of disk space for 24 h, or 1 Tb for the entire week. Now, with DDDs is possible to record dose-fractioned frames instead of a single averaged image, producing around 2 Tb of data in a single day.

Despite workflow systems have been explored and used extensively in bioinformatics, there is little progress in that direction in the cryo-EM community. Current packages provide tools for data management and HPC setup, but still a more general and flexible solution is required to improve the automation of the processing pipeline. Cloud computing, which has emerged as a potential model to address a broad array of computing needs and requirements, is only been explored in cryo-EM recently [71]. The system described in this thesis has been designed to make a more efficient usage of heterogeneous computing resources in an HPC environment. It handles very carefully all data management and simplifies the settings for the final users.

1.5 Project Context

Scipion development has been done in the context of the Instruct Image Processing Center (I2PC, http://i2pc.cnb.csic.es). INSTRUCT, (http://www.structuralbiology.eu/) is the European initiative for Strategic Scientific Infrastructures (ESFRI) for Structural Biology. As such, its centers are primarily designed to provide support, not to perform research. However, a cutting edge infrastructure must also be connected with cutting edge research. In that way, the Biocomputing Unit (BCU) at the "Centro Nacional de Biotecnología" (CNB) moved some research lines to the context of the I2PC.

The I2PC started with the objective to provide efficient support for research projects in Europe demanding expertise in image processing. In order to support a large collection of projects, the first activities of the I2PC were oriented toward the design and implemention of the necessary software tools. It was recognized the need for a platform that could provide a friendly GUI on top of advanced image processing algorithms and, at the same time, assure standardization, traceability and reproducibility.

For the last twenty years, the BCU has developed the well-known software package Xmipp. Nonetheless, the I2PC commitment was to provide access not only to Xmipp, but to other available software as well. This was the main driving force behind the development of Scipion, which can also be seen as an "extension" of Xmipp, in the sense that it is used in our group for processing and provides access to new developed algorithms.

In that way, it is very likely that the project will be supported and maintained in the long term.

Chapter 2

Objectives

The work of this thesis is focused in the design and implementation of Scipion, a new software framework addressing the problems of integration and interoperability in 3DEM, while providing full tracking of the whole image processing workflow. The project is being developed in the I2PC as part of INSTRUCT. In that context, there is a commitment to provide state of the art algorithms and software to the EM community, including software from other groups.

2.1 Integration and interoperability

One of the main goals of this work is to allow the use of different EM software packages in the same project overcoming different formats and conventions. Researchers should be able to smoothly combine tools from different packages at each step of the processing pipeline. The internal conversions should rely in a well defined standard for system coordinates, geometrical image transformations in 2D and 3D spaces.

2.2 Traceability and Reproducibility

Maintaining a high degree of transparency in scientific reporting is essential not just for gaining trust and credibility within the scientific community but also for facilitating the development of new ideas. In that way, another important feature of the framework should be the ability to track all the steps performed and the exact parameters selected for each run. This will allow scientists to further check in details the methods used in a particular project for a given study, or easily reproduce some of the steps with small changes.

In the 3DEM field (as in many other fields) experiment reproducibility is low. This work should ease this task by developing a simple workflow management system to support reproducible computing. Direct benefits of reproducibility are:

1. Other people who want to do research in the field can really start from the current state of the art, instead of spending months trying to figure out what was exactly done in a certain paper.
2. It highly simplifies the task of comparing a new method to existing methods.

2.3 Distributed computing and High-throughput

Despite computer power has increased over the last decades, it is quite challenging to develop scientific software that exploits all technological advances while improving researchers productivity to obtain their results. The same problem applies to cryo-EM , where existing programs need to perform intensive computations while handling huge amount of data.

This work should also address problems related to distributed computing. The framework should efficiently use the computing resources available for a project. It should ease the execution of jobs in different computing environments (clusters, supercomputers, GPU or cloud). The configuration for distributed execution should be centralized and reused by all integrated software in the framework.

2.4 Ease of Use and Extensibility

The proposed solution should also benefit software developers in the 3DEM community by providing an integrative framework with built-in tools. The system should be easy to extend to allow quickly incorporation of new algorithms emerging in the field. In that sense, the task of adding a new algorithm should be as easy as possible without the need of a deep understanding of the internal implementation. It is desired to provide scientific developers with an API to store/retrieve data without requiring knowledge about database or more complex mechanisms.

Chapter 3

State of the Art

The speed and efficiency with which scientific workflows may be performed have increased with the use of modern hardware and software technologies. One workflow can be executed many times with different programs, versions or parameters, or even modified input data, and scientists can compare results from these executions. However, dealing with large volumes of information produced by many executions under a variety of conditions becomes increasingly difficult. In this context, new tools (workflow engines) have been developed to handle the data generated in each execution, along with the origin of this data and the details of a particular execution. It is important to note that, before choosing which data has to be stored, it is necessary to define how this data needs to be structured (data model) so that it can be later recovered and understood.

In this chapter the evolution of several widely-used software packages in EM is analyzed from a software engineering point of view. After that, it is briefly discussed what has been done so far in our field in terms of data modeling. Finally, some of the most popular workflow engines used in scientific applications are reviewed, specially those that are particularly interesting for this project.

3.1 Software Packages in Electron Microscopy

This section presents an overview of some well-known packages in the EM community. The goal is not to provide a detailed survey of all the packages in the field, but to show different strategies used by EM software to handle the increasing computer requirements. The analysis will highlight their progress in terms of data management, integration and workflow automation.

3.1.1 SPIDER

SPIDER (System for Processing of Image Data from Electron microscopy and Related fields) was presented at the Toronto Conference on Electron Microscopy [72, 73]. It was designed as a modular system for image processing in electron microscopy. Despite being later used for 2D crystals and helical processing, the main focus of the package was single particle averaging, classification, and reconstruction.

The first programs were created on a PDP 11/45 around 1975 and, in 1996, the system contained 115,000 lines of FORTRAN code [53]. From its beginning, SPIDER introduced a full programming environment for EM image processing which incorporated DO loops, IF statements, and a hierarchical calling structure based on a command interpreter [73]. These properties resemble the features of modern scripting languages, which were not available or widely adopted at that time. The SPIDER scripting mechanism allowed the development of applications to deal with a large number of images, which also involved a complex set of mathematical operations. Moreover, users could design the processing path through the use of batch scripts, even without a strong background in scientific programming.

In SPIDER, users can invoke any operation interactively at the command line prompt. Commands are two-letter codes (usually mnemotechnic, e.g., "CP" for copying) that may be modified by an option separated by a blank (e.g., "CP PDB" — import from protein data bank). The interactive mode is the standard way for dealing with a limited number of images while developing a processing scheme. However, a reconstruction often involves manipulation of larger datasets in a complex and partially repetitive sequence of operations. Thus the commands invoking SPIDER operations may be stored in batch control files, which can be executed just like single operations. Special types of batch control files are called procedures. Procedures are text files containing a sequence of commands that allow run-time replacements of specified parameters and file names.

Regarding data formats, SPIDER stores images as a sequence of records, each containing a single image line, enabling random access to any portion. The default representation of a pixel is as a four-byte floating point number, to maintain high precision in the processing of molecular images. Related metadata is stored in SPIDER document files, which are text files containing numerical results. For example, angles and shift values resulting from different refinements of alignment of an image set may be vectorially combined without the need of interpolation. Other operations on document files, such as sorting, are also available.

Originally SPIDER contained all the code for the system, including calls to various graphics devices. With the arrival of graphical user interfaces (GUI) and standardization on the Motif/XWindow system, it became difficult to integrate interactive graphics into a command-oriented program such as SPIDER. This problem was addressed by the development of WEB, an independent "image viewer" [53]. After that, SPIDER was exclusively used for operations which do not require, or benefit from, visual interaction, whereas WEB was used for those operations. Users typically launched SPIDER scripts from the command line while simultaneously analyzed the results in a WEB session. Both components used a common data format, so that files could be interchanged between them. WEB served to visualize various results from EM image processing such as: display of images and slices of volumes, contrast enhancement, histograms, windowing, profiling across an image, masking, computation and display of run-time power spectra of selected areas, and recording of screen contents. Other operations were devoted to graphical volume manipulations such as contouring, surface rendering, annotating images with text, lines, etc. Some of the features implemented in WEB are now available to users (and developers) in open source specialized visualization packages, including: USCF Chimera [74], VMD [75], and PyMOL [76], among others.

To facilitate the execution of batch files using a GUI, in 2006 a new layer was introduced: SPIRE [77]. The main goal was to simplify the execution of the processing pipeline in SPIDER, while still preserving the flexibility of writing custom scripts. This new high-level interface did not required a major redesign of the underlying system. Another goal was to manage and organize the many output files created during a project. Moreover, users who were new to SPIDER could start processing without having to learn in detail the command-line interface. SPIRE was written in Python, and used the Tkinter widget library (www.python.org).

SPIRE implements a GUI for executing batch files, organized around dialogs, which present a list of related batch files. Each batch file is associated with an Execution and an Edit buttons, and a brief descriptive label. SPIRE monitors executing processes and, after finishing, checks if expected outputs where generated and add them to an internal project database. The list of successfully executed batch files is displayed in the Project Viewer, as well as output files generated by each one.

When a new project is started in SPIRE, a dialog allows to enter some information related to the project. The set of batch files used and its directory structure can be defined in some XML (eXtensible Markup Language) configuration files, which also control the arrangement of the GUI widgets. A series of batch files can be saved to a list and executed automatically, one after each other, until all are finished or an error halts the process. SPIRE also allows to edit a batch file in an automatically-generated GUI,

but only if the script contains a header following certain rules. Apart from the benefit of the generated GUI, the header constrain encourage the good practice of separating input/output variables from the processing commands in the batch file.

SPIRE stores the output files into an internal project database, which is not synchronized with the system files on disk. In the project view, only the files in the database are shown, while the SPIRE's file browser allows one to see files on disk. SPIDER procedures usually produce two types of data: document files (text files with columns of values) and binary files (2D images, 3D volumes and Fourier transforms). In the SPIRE's Options section is possible to associate different viewers (external programs) for these type of files. Document files are commonly displayed in a text editor, while image files are sent to an image/volume viewer. It is also possible to generate HTML files from a project, which is specially useful at the end of a project as a review of the processing history. Another interesting feature is the possibility to import data from an existing project database, as well as export when the project is finished.

3.1.2 EMAN / SPARX

EMAN (Electron Micrograph ANalysis) is another of the most used packages in the Cryo-EM community. It was presented in 1999 [78], including tools for the whole single particles pipeline to achieve high resolution results. One of its main goals was to make this process more accessible to inexperienced users, while still providing advanced tools for the experienced ones.

EMAN was provided to the community free of charge with full source code. It was written with portability in mind to compile with relatively little effort on Unix based systems. GUIs were developed using the freely available QT toolkit [79], while Fourier transforms were based on the free FFTW library [80]. The image library included support to read and write a variety of formats, including MRC, IMAGIC, SPIDER, TIFF and GIF, among others. It also supported parallel processing on clusters, SMP supercomputers or sets of individual workstations.

Despite the package had a tiered architecture, after many years of development (original in Objective-C, then C++ and later Python bindings) there was no clear organization to incorporate new features [47]. The need of a complete re-factoring became clear during the collaboration with the PHENIX project [81]. Many of the necessary features to take the next steps could not be incorporated into the original EMAN1 design [47].

The EMAN2 refactoring was focused in providing extensibility, a complete Python interface, introspection capabilities for GUI integration, metadata management and built in

unit-testing. All the command-line programs were ported from C++ to Python, a more flexible language. The GUIs were based on a hybrid approach: used WxPython [82] for image display widgets (for compatibility with PHENIX) while other developments used PyQt4 [83].

The SPARX (Single Particle Analysis for Resolution Extension) project was created to provide a uniform environment for end-users, in which they might develop different processing strategies without dealing with all software suites [52]. The system introduced a graphical programming environment to reduce the required knowledge about underlying algorithms. Issues such as file format conversion and Euler angles conventions were dealt with as automatically and transparently as possible. Another of the goals of SPARX was to create an integrated environment for both X-ray crystallography and cryo-EM single particle analysis.

SPARX was based in the core libraries provided by EMAN2 and, consequently, was designed with a similar architecture. It used C++ for the compute-intensive code, while Python was used for higher level tasks. The link between C++ and Python was generated using the Boost Python Library [84]. This same architectural approach was used in the related PHENIX project for crystallographic structure determination[81]. Users could interact with SPARX in three ways: (i) through a graphical programming interface, which requires no formal programming background, (ii) through use of pre-written scripts from a command shell, (iii) through a text-based customized Python interpreter. In principle, SPARX was designed to be expanded with programs from other software packages, although initially only some SPIDER command were integrated.

3.1.3 XMIPP / RELION

Xmipp (X-Windows-based Microscopy Image Processing Package) is another suite that is mainly focused in single particles analysis. Its development started around 1987 and it was presented in the special issue of JSB on 1996 [44, 85]. At that time, the package was written in ANSIC-C and used the X11 library for graphical outputs. From a technical point of view, the most relevant features of the package were the simplicity and portability of standard C code with an X-Window interface. Different operations were implemented in separated command line programs, that could be combined to perform a specific task. In other cases, where there was required more user interaction, graphical tools were implemented such as the tilt pair particle picking. The package was particularly rich in those methodological areas in which the group was more active, mainly classification and 3D reconstruction tools. A variety of classification methods

were provided, ranging from neural networks [86] to fuzzy multivariate statistical analysis [87].

The image format adopted was the same than in SPIDER, with the strategy of using both packages in a complementary way. There were two exceptions with respect to the compatibility with SPIDER: (a) images could be stored in 8 bits with no header to save space and, (b) original images were preserved during processing and the alignment parameters were stored when possible to avoid interpolation errors.

In 2004, a complete re-factoring of the package was done. It was entirely ported to C++ and provided with a better hierarchical organization of well-documented classes and functions [88]. From the previous version, new methodological developments were incorporated. The Xmipp's core was composed by several libraries: a data structure library, a classification library, and a reconstruction library, among others. The data structure library provided core classes such as vectors, matrices, volumes and Fourier transforms, as well as I/O routines, error handling, time and random number functions, among other utilities. The classification library supported implemented algorithms, including hard and fuzzy clustering, partitional clustering, principal component analysis, and SOMs. The reconstruction library provided methods for obtaining a 3D structure, including functions for angular assignment, CTF estimation and correction, 3D reconstruction, and reconstruction-quality evaluation.

That modular design offered a convenient platform for rapid testing of new algorithms by software developers, although many stand-alone programs offer a broad functionality to the user as well. For experienced users, this diversity of programs provided a high level of flexibility in designing different processing strategies. Furthermore, the modularity of these programs allowed changing from or to alternative packages at almost any point in the data processing workflow.

For inexperienced users, however, the multitude of programs presented a relatively steep learning curve. To overcome this problem, in 2008 (Xmipp version 2.4), an additional layer was built on top of the hierarchical structure of Xmipp, consisting of a collection of standardized protocols for most common operations [89]. These protocols synthesized numerous existing scripts and recipes that flourished among the Xmipp users community, thereby representing years of experience by multiple researchers.

These new protocols were implemented as standalone executable Python scripts, each with a header that defines its corresponding parameters. Users could modify these parameters and execute the script either through a GUI (built on the fly from the protocol parameters) or from the command line (editing the header parameters in a standard text editor). In this way, the user was aided during the processing workflow

since the output of one script could be used as the input to the next. Additionally, the protocols standardized logging and visualization of the results. Although the advantage of these scripts was evident for inexperienced users, expert ones could also benefit from this standardized working environment, facilitating the exchange of intermediate results with alternative packages or other users.

Another huge re-factoring of the package was finished in 2013, which led to a new major release, Xmipp 3.0 [54]. This version introduced notable improvements in terms of project management, core libraries, GUIs, image formats and standardization. In Xmipp 2.4 there was no clear concept of a project. While there was a folder with a set of Python protocols for each type of execution, there was no formal relationship among the scripts. In Xmipp 3.0, the processing pipeline was organized around projects, composed of protocol runs. A single database contained all the information related to a given project, but only the metadata, since image files were not stored in the database. The SQLite library [90] was used for the project database, avoiding the need to setup a dedicated server. Projects could easily be moved among computers by simply copying the project folder.

Previous to Xmipp 3.0, the processing was protocol-oriented: the user launched scripts to perform the required operations. Communication between different protocols was achieved manually, which required that the user know what output files were needed as input for the next protocol. There was also no way to track workflows (that is, which protocols were executed and in which order). In Xmipp 3.0, protocols were also implemented in Python, but they were divided into more atomic steps (e.g., functions and program executions) stored in the database. With this approach, users could monitor the progress of a run, restart it from a specific point and validate that the expected results were obtained. Moreover, result files were standardized even more to facilitate inter-communication between protocols.

In the new version the GUIs were notably enhanced. A new application was developed for the visualization of images in a gallery or table model. Particle picking (also tilt-pairs) was re-implemented from scratch. The project window showed the full processing history, either in a list or a graph view. Parameter forms were generated automatically from the protocol definition and several "wizards" were added, specific GUIs to help selecting some parameters.

At the same time that Xmipp was been ported from 2.4 to 3, one of the developers moved to Cambridge and started a new package, RELION [29]. It reused the same Xmipp core libraries for image and metadata management. The maximum likelihood methods in Xmipp evolved in RELION into a Bayesian framework that implemented a maximum a posteriori approach. In the new package, special effort was done to reduce

the computational costs of this type of algorithms. Moreover, it strongly supported the so-called gold-standard FSC procedure to prevent overfitting.

RELION was implemented in C++ and its code was freely available from the web. The package did not provided any binding to Python and the GUI was developed in C++ as well, based on the FLTK toolkit [91]. Regarding parallelization, RELION has followed the same hybrid approach than Xmipp, combining MPI processes with POSIX threads. At the level of MPI, usually a master node dispatches small jobs to computing nodes on demand and combines all the information at the end. On the other hand, using threads over MPI allows to access the same computer memory, so that the memory is used more efficiently in modern multi-core computing nodes.

In Relion 2.0 [50], the latest major version, the implementation was enhanced to take advantage of GPUs (Graphical Processing Units), addressing the most computationally intensive steps of its workflow. Both image classification and high-resolution refinement were accelerated more than an order-of-magnitude, and template-based particle selection was accelerated well over two orders-of-magnitude on desktop hardware. Moreover, memory requirements on GPUs was reduced to fit widely available hardware while maintaining similar results. All these improvements have allowed high-resolution cryo-EM structure determination in a matter of days on a single desktop computer.

3.1.4 Appion

Till the introduction of Scipion, Appion was the only platform that allows real integration among software packages [45]. Appion is tightly connected with a previous project by the same group, Leginon [92], a system for automated data acquisition that tracks and records all microscope parameters associated with every acquired image into a MySQL database [93]. This provides a straightforward method for comparing and contrasting datasets acquired during different microscope sessions.

Similar to Leginon, Appion was implemented primarily in Python, providing inter-package compatibility through generic "wrappers" for numerous software tools used by the EM community. Functions from SPIDER, EMAN, Frealign, Imagic, FindEM, Xmipp and Matlab were all incorporated into the first Appion prototype. One of the reasons to choose Python, a powerful and easy to learn language, was to quickly incorporate new tools into the pipeline.

One of the cornerstones of the Appion system was the underlying database, which contained all the processing information and provided the links between disparate packages. The creation and maintenance of this database was based on the Sinedon library, which

provided an automatically adaptive database structure, that also supported the Leginon and Project databases. The Appion database was relationally linked to these other databases such that information could be traced back from every point of processing. The Project database stored global data associated with the overall biological project, the Leginon database primarily stored information related to the raw data, and the Appion database stored all the parameters, inputs, and outputs from every processing step.

The Sinedon library was based on the concept of object-relational mapping (ORM), which refers to transferring objects from a programming language to a database system. With this library, new tables could be created on-the-fly from Python scripts, as well as launching different queries. The module SQLDict, allows to create Python dictionary-like classes describing the table structures. New objects could be created and filled with values in Python, and then transparently stored into the database. The corresponding table is created if it does not already exist, and the object data is stored in a new row of the table. An essential aspect of Sinedon is its ability to automatically create relational links between classes.

Through Sinedon, Appion developers are not required to write SQL queries. The Appion database, which consists of nearly 100 tables and which is constantly being modified and extended as new procedures are added to the pipeline, does not require any manual maintenance. The underlying database is essential to streamline the reconstruction process and to integrate disparate software tools, as well as to track and manage the metadata generated in each step.

The Python based wrapper scripts in Appion use the classes defined in *appion/Data.py* to store and retrieve data from the database. The general module *appion/Loop.py* iterates over a set of images in an experiment and pause the processing until a new image is entered in the database. This functionality allows to perform some operations (e.g. Particle Picking or CTF estimation) concurrently with the data acquisition.

While the Appion Python scripts can be executed from a command line, most of the processing steps can be launched from a web-based GUI. The web pages are generated using PHP [94], a server-side scripting language, from which Appion SQL database is accessed for reading and writing. This GUI provide access to the underlying processing pipeline, even if the command line can also be used. The Appion pipeline guides users over a wide variety of processing and analysis software packages. This pipeline makes it easy to ensure that each data set is analyzed in the same manner, and variations can be monitored throughout the process.

3.2 Data Modeling in Electron Microscopy

Within the context of data-centric scientific computation, there is a need for a data model that describes how the data is structured and how it may be used. Traditional methods of handling scientific data such as flat sequential files are generally inefficient in storage, access or ease-of-use and prohibit effective data exchange and interoperability between applications, especially for large complex data sets. Modern, commercial relational data management systems do not offer an effective solution because they are more oriented to business applications. The relational model does not accommodate multidimensional, irregular or hierarchical structures often found in scientific data sets nor the type of access that associated computations require. In addition, relational systems do not provide sufficient performance for the size, complexity and type of access dictated by current and future data sets and their potential usage.

Current research in EM depends on the analysis of huge amounts of data. These are in several formats and use different vocabularies. Furthermore, data are often stored or distributed using formats that leave implicit many important features relating to the structure and semantics of the data. It is really surprinsing that a formal data model has not been developed for EM data processing, independent of the underlying storage format.

For example, in Leginon [92], a database is used to organize images acquired by the electron microscope. Nonetheless, all the Leginon's documentation found emphasizes in the benefits of the relational database and its implementation, but does not points out to a more logical data model. The same happens to Appion [45], that follows a similar architecture to Leginon. In this case, a relational database is implemented for data processing but the data model is not clearly described.

Another example is EMEN2 [95], developed by the same group of EMAN2 [47], that implements an electronic notebook using an object oriented database. This database is built around two concepts: protocols and parameters. Parameters are values that might be recorded during an experiment and they are defined by a description, data type and units. A protocol is composed by many parameters and defines the specific type of record that will be stored in the database. EMEN2 uses hierarchical relationships between parameters and protocols to create some kind of ontology. Nevertheless, from an image processing point of view, EMEN2 provides only some simple visualization and evaluation tasks. Their authors claim that it is possible to incorporate portions of the processing pipeline, but currently tasks such as 3D reconstruction and refinement are treated as external processes. It seems that the ontology developed for EMEN2 is not directly used by EMAN2 for the implementation of image processing programs.

Finally, it should be mentioned the EMX [96] initiative, that implemented an exchange format between EM software packages. EMX was one of the main results obtained from the I2PC Developers Workshop held in Madrid in 2012. In EMX, a reduced data model for EM image processing was described and implemented using XML. The proposed data model described some entities such as *Particles* and *Micrographs*, while others were left out of the initial standard. This work represents a first step in the direction of standardization and data modelling, although this model is only supported by current EM packages as an import/export format, but not as their processing model.

3.3 Scientific Workflows

Scientific workflows have been increasingly used in the last decade for data intensive science [97]. A variety of workflow systems, both open source (e.g. Taverna [98], Galaxy [99], Vistrails [100], Kepler [101], Triana [102], ASKALON [103]) and commercial (e.g. Pipeline Pilot2 [104]) are in use in many scientific disciplines such as genomics, astronomy, cheminformatics, etc.

In this section three of these workflow systems are analyzed: Taverna, Galaxy and VisTrails. These were selected because some of their application domains have points in common with our field. Moreover, some ideas implemented in Scipion have been inspired from features in those systems.

3.3.1 Taverna

The Taverna workflow tool suite (`http://www.taverna.org.uk`) allows the creation and execution of complex scientific pipelines, combining distributed Web Services and/or local programs. These pipelines can be executed in desktop computers or in HPC environments such as clusters, Grid or Cloud. Taverna is widely used in bioinformatics (e.g., proteomics or transcriptomics) and in other areas such as data mining. Taverna provides scientists access to a large number of tools and resources made available by different institutions around the world. The workflows in Taverna are like bioinformatics protocols that can be reused and shared by the community. A repository of public workflows is available at `http://www.myexperiment.org`.

A Taverna workflow is defined by a directed graph whose nodes (called processors) represent software components. A processor receives data from its input port and produces results that go to its output ports. Each edge of the graph connects one output port from a given processor to the input port of another one, and denotes a data dependency

between them. Additionally, control links can be placed between the two processors to denote that the dependent processor can only be executed after the other has produced all its outputs. Conceptually, a workflow is executed by pushing data through the directed data links from one processor to all of its descendants. A processor is ready to start when all of its inputs ports are populated with a data item. Each processor has associated an underlying activity (e.g., a web service or a local script) that can be another workflow. This approach allows to encapsulate several tasks under a sub-workflow that can be used as a normal activity in another context.

Most Taverna workflows are composed from a mixture of distributed Web Services, local scripts and other service types. Nevertheless, in some cases the installation only uses local services, where data and software are placed together and network traffic is notably reduced. One example of such scenario is the usage of Taverna in cloud environments, where the input dataset needs to be uploaded once and the engine and services receive a reference to the data. The main advantage of using distributed services, however, is that computations can be off-loaded to remote service providers and there is no need to install tools or data locally. Consequently, complex analyses can be performed regardless of local infrastructure, using distributed tools and resources. A disadvantage of relying on external Web Services is that workflows can be broken by service unavailability or changes in their interface [105]. Nonetheless, there is a large redundancy of web service functions that allows to identify reliable services and alternatives to make workflows more robust.

The workflow engine in Taverna is implemented in Java as a multi-threaded object model, where processors are represented by objects, and data transfers are realized using local method invocations. Since each processor is a separate thread, it can autonomously start working and transfer output data without the need of centralized enactment system. Although execution is decentralized, a facade pattern is used to provide external components with a single point of contact (e.g, monitor and result presentation in Figure 3.1).

The Taverna Workbench is a graphical application that allows bioinformaticians to develop new workflows and test analysis methods, by either creating the workflows from scratch or by composing existing ones (See Figure 3.2). The Workbench can be freely downloaded from `http://www.taverna.org.uk/` and easily installed in Windows, Linux and Mac OS X. Users can easily identify and combine services by dragging and dropping them onto the workflow design plan.

The Taverna Server offers a different mode for workflow execution. It serves as an environment for serving curated workflows to a larger community of scientists. The Server installation is usually combined with a web interface (the Taverna Player) that

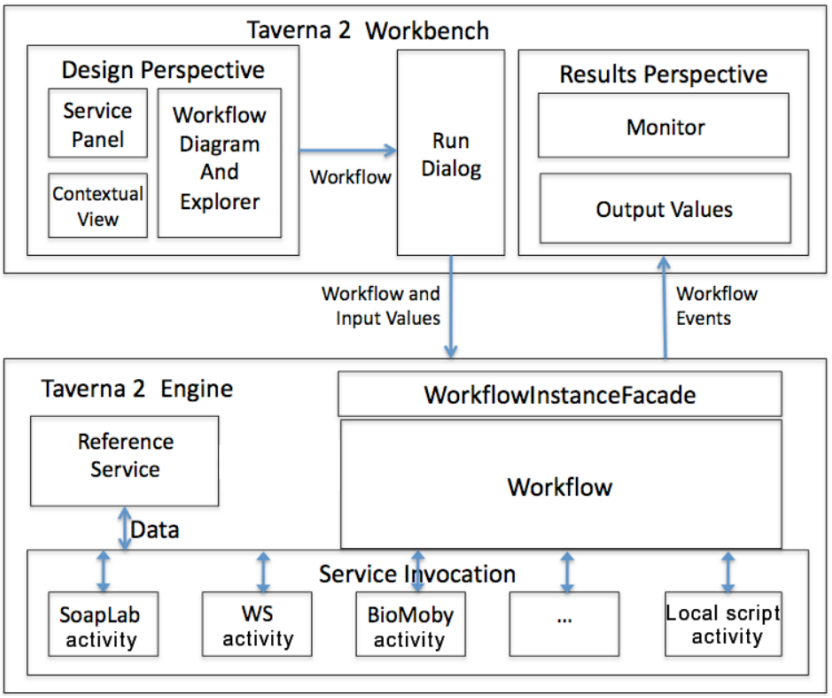

FIGURE 3.1: Overview of Taverna 2 architecture.

provides access to a collection of workflows without the need to download or install any software. One drawback of this mode is that users cannot modify or add new workflows. However, there is an intermediate solution, the Taverna Lite, that also allows users to upload new workflows from different sources.

3.3.2 Galaxy

In 2005, the Galaxy Project started with the main goal of enabling researchers without programming and system administration expertise to perform computational analysis through the Web. It was designed as a system for the integration of genomic sequences, their alignments and functional annotations. Despite being available several genome browsers at that time for data visualization ([106], `http://genome.ucsc.edu`], NCBI MapViewer [107], and Ensembl [108], `http://www.ensembl.org`), more complex analyses still required programming and databases skills. By addressing this problem, Galaxy allowed users to gather and manipulate data from several sources, while storing every user action in the system history, a key element of the framework.

One of the core features of Galaxy is its web-based GUI that provides users access to a large set of computational tools. The main window (shown in Figure 3.3) is divided into three panels: the left one shows available tools grouped into categories that can also be searched; the middle panel is used to fill the parameters of the selected tool or to inspect an output dataset; the right panel contains the full history of actions and

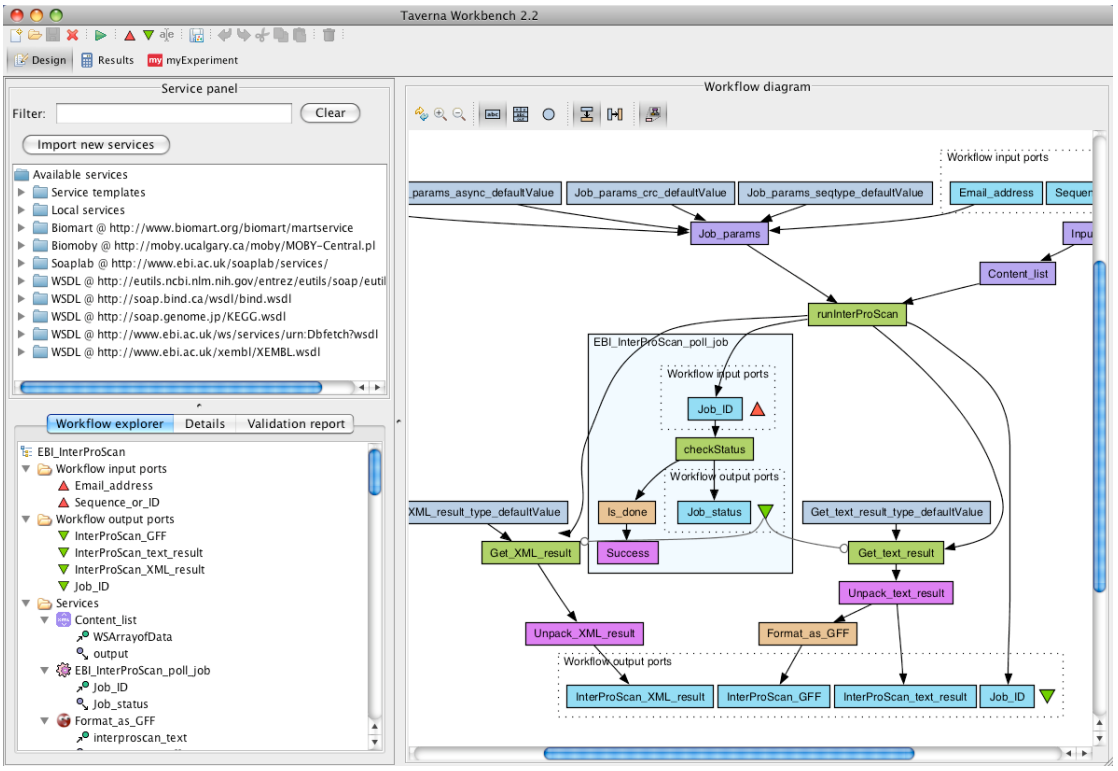

FIGURE 3.2: Graphical interface of Taverna 2 Workbench.

produced outputs. All tools use a single interface to simplify new tools integration and to maximize usability. In the recent years the tools interface was rewritten to improve the user's experience. One addition was the possibility to include citations to facilitate researchers to reference the methods used in their analysis.

Another important component of Galaxy is its graphical workflow editor (see Figure 3.4). It greatly simplifies the task of creating multi-step processes by drag-and-drop different tools and connecting them so that the output of one becomes the input of another. Workflows are critical to automate and repeat the analysis of large datasets. Workflows capabilities have been improved since the first versions of Galaxy. For example, workflows can now be paused and restarted. Moreover, entire workflows can be embedded into another workflow as a single step, what facilitates the creation and management of complex analysis in the Galaxy's workflow editor.

Over the years the number of available tools (more than 650 at the time of the writing) for the Galaxy community has increased notably [99]. Tools vary from simple text manipulation and statistical operations to the analysis of high-throughput genomic datasets, including quality control, genotyping and variant identification, motif finding, RNA-seq, ChIP-seq and metagenomics. The system can handle a large number of data formats (e.g. bam, bcf, bed, bedgraph) and provides convenient conversion and visualization tools. The Galaxy team introduced the Tool Shed [109] as the repository of

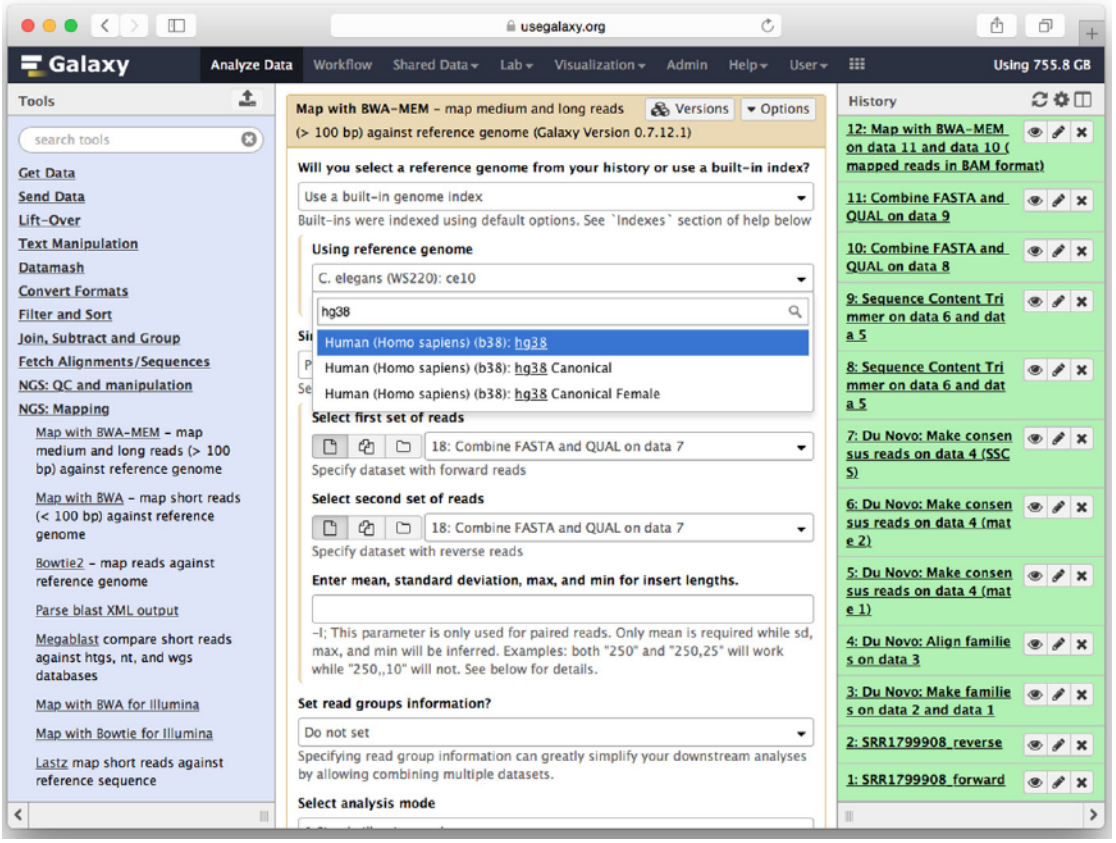

FIGURE 3.3: Galaxy main windows, divided into three panels: (left) available tools grouped into categories; (middle) fill the parameters of the selected tool or to inspect an output dataset; (right) full history of actions and produced outputs.

all new tools and updates, which greatly simplifies the process of tools deployment and discovery. It also improves versioning and reproducibility regarding the software used in a project. The development of new tools is regulated by a commission of volunteers (IUC; https://wiki.galaxyproject.org/IUC), which review and approve them via GitHub (https://github.com/galaxyproject/tools-iuc).

The Public Galaxy Server (https://usegalaxy.org) is an open installation (or instance) that provides access to many tools, visualizations and data sources for the entire community. It has been running since 2007 for everyone to analyze their data free of charge. Most genomic analyses usually need very large datasets, including reference genome sequences, indices and gene annotations. Galaxy recently included a new interface for uploading unlimited number of files concurrently [99]. Data can be uploaded from the local computer or from a remote site (e.g., http or ftp) and the process can always be monitored or even paused and reconfigured. The public server already contains a large number of reference genomes, which are constantly updated. Galaxy also allows researchers to include their own reference data if it is not available from current sources. Data management have moved from a manual process to an automated process using

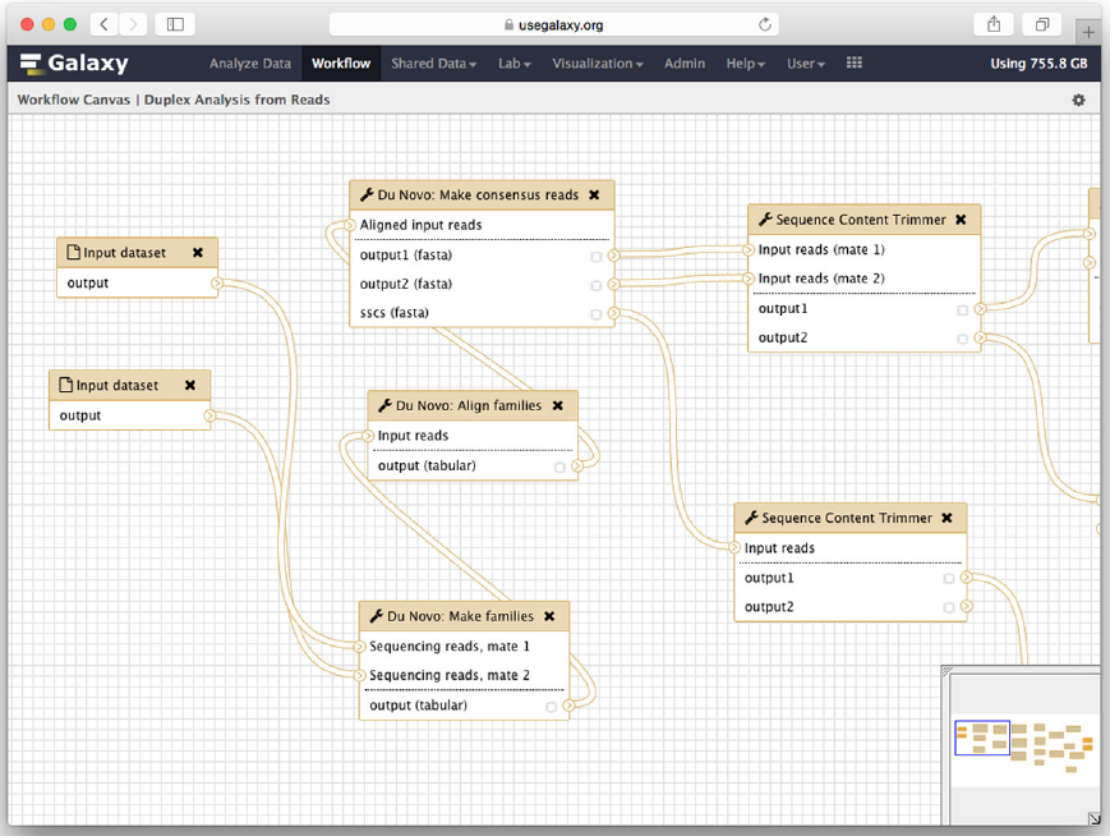

FIGURE 3.4: Galaxy workflow editor.

Data Managers [110]. This approach also makes more easy to fetch and install new data for Galaxy administrators.

Despite the public server is an extremely valuable resource for the community, there are limitations to guarantee fair and safe access. First, there are quotas on the amount of data users can have and the number of concurrent jobs that can be ran. Second, users cannot add their own tools to the server. To overcome these limitations, the Galaxy Team have made available the entire Galaxy platform on the Amazon Cloud [111], that provides computing resources on demand, as a pay-per-use service. Personal Galaxy cloud servers are configured in advance with many tools and reference genome data. Nevertheless, each instance is entirely customizable allowing users to install additional tools and data without any storage quotas.

3.3.3 VisTrails

VisTrails [100, 112] is a workflow system to streamline the pipeline of visualization processes. All the data and metadata related to the visualization products are tracked among all steps. By capturing the provenance of both the visualization processes and data they manipulate, VisTrails enables reproducibility and simplifies the problem of

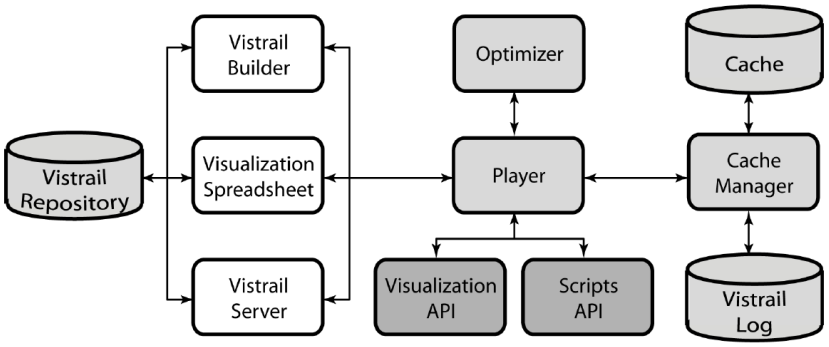

FIGURE 3.5: Overview of Vistrails architecture.

creating and maintaining visualization workflows. Scientists can explore different parameters and strategies by navigating through different dataflow versions and comparatively visualize different results. VisTrails defines the concept of visualization trails (vistrails), which combines the dataflow concept from other workflow systems with its evolving history.

The main components of the VisTrails system are shown in Figure 3.5. The Vistrail Builder allows users to create and edit vistrails (or workflows), which specifications are saved in the Vistrail Repository. Users may also interact with stored vistrails by executing them through the Vistrail Server or by importing them into the Visualization Spreadsheet. Each cell in the spreadsheet contains information related to a dataflow instance, which parameters can be modified by the users. The Vistrail Cache Manager controls the execution of different vistrails by tracking the operations that are invoked and their parameters. Only new combinations of operations and parameters are requested from the Vistrail Player, which executes the operations by invoking the appropriate functions from the Visualization and Script APIs. All operations are logged in the Vistrail Log.

In VisTrails, a dataflow is defined as the sequence of operations used to generate a visualization product while a vistrails consists of several versions of a dataflow. A vistrail serves as a log of all the steps followed to obtain a number of visualizations and can also be used later to automatically reproduce the same (or slightly similar) processing. A vistrail can be represented as a tree where each node corresponds to a dataflow. An edge between a parent and child nodes represents a set of changes applied to the parent node to obtain the dataflow of the child node. The vistrails are stored using XML, which is a very flexible format to accommodate variability in the information being stored. A clear benefit of using an open and self-describing format is the ability to query, share and publish vistrails among scientists.

The execution of a vistrail can take a long time depending on the size of the dataset and the complexity of the operations used. Taking advantage of the high-level specification of a vistrail, the system analyzes and optimizes the execution of dataflows. The Vistrails Cache Manager takes care of the scheduling of modules, it identifies previously computed subnetworks and replace costly operations by constant-time cache lookups [100]. For the modules that needs to be computed, the Vistrail Player receives a dataflow instance (an XML file) and executes it using the underlying Visualization or Script APIs.

VisTrails provides graphical user interfaces for interacting with the system (See Figure 3.6. The Vistrail Builder can be used to create new dataflows and modify existing ones. It writes (and also reads) dataflows in the same XML format as the other components of the system and uses a nodes-and-connections representation similar to other workflow systems. Another important graphical component is the Visualization Spreadsheet, that provides the user a set of visualization windows with a tabular layout. In this way, users can use more screen space and easily compare results from different vistrails executions.

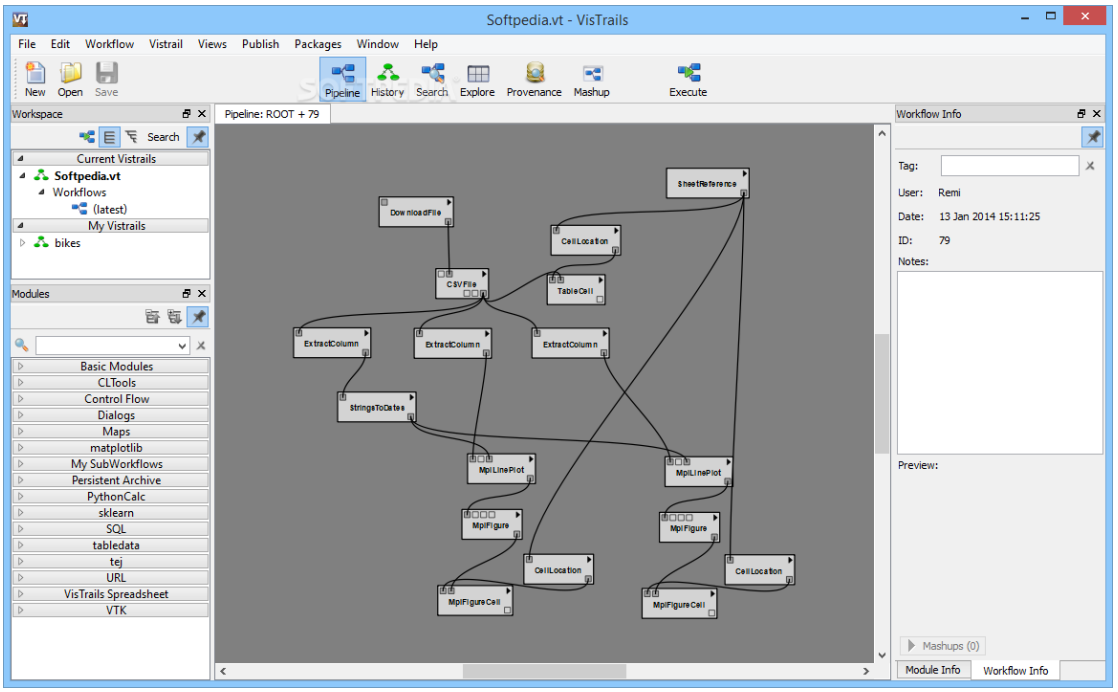

FIGURE 3.6: Graphical interface of Vistrails workflow editor.

The initial motivation for the development of VisTrails came from the Center for Coastal Margin Observation & Prediction (`http://www.stccmop.org`), where scientists deal with a huge amount of observed data from the Columbia River and a wide set of simulations and forecasts. Understanding exactly which data and approaches were used (especially as this data is continuously gathered over time) is extremely important in producing accurate conclusions. VisTrails was used to automate some of the visualization products using workflows.

As the development of VisTrails progressed, the project attracted more users from diverse scientific disciplines, ranging from invasive species modeling and climate data analysis to theoretical physics. Some of the more relevant applications of VisTrails include: the Algorithms and Libraries for Physics Simulations project (http://alps.comp-phys.org), (2) Software for Automated Habitat Modeling (http://www.fort.usgs.gov/products/software/sahm/) and the Ultra-scale Visualization Climate Data Analy- sis Tools (UV-CDAT) project (uvcdat.llnl.gov).

Chapter 4

Design and Implementation

In Chapter 2 the main goals of this work were presented: (1) integration of different EM software packages in the same project, (2) traceability and reproducibility, (3) distributed computing and high throughput, and (4) ease of use and extensibility. This chapter will describe the design and implementation of the whole framework.

To deal with points (1) and (2) it became clear that the first, high priority task, was to model the data (e.g. micrographs) and metadata (e.g. alignment information) involved in a typical EM image processing project. In single particles analysis, at the conceptual level, most packages deals with similar initial data and follow almost identical workflows. The problems arise when these data are stored in different file formats and follow different conventions. The same happens with the processing pipeline, the steps are conceptually similar, but there are some differences between packages in the specific way to execute them. By creating a more conceptual model, the Scipion framework will abstract both users and developers of the underlying data and will allow to work with a single representation. The same benefit applies to the development of new methods, the developer can focus only in a specific part, that is decoupled from other components of the system. In that way, the developer only needs to deal with a defined API, without concerning about which package produced the input data or how it was stored.

The proposed model is divided into two parts: (a) the data model, which represents entities (or objects), their relationships, and encapsulates package-specific formats and conventions and, (b) the process model, which defines the methods or algorithms used at different stages of the project and clearly states their inputs/outputs in terms of data objects. This model is a cornerstone of the entire Scipion framework to integrate of existing EM programs and to adapt to future methodologies and requirements. Moreover, having a good specification of the processed objects allows to go one level up and concatenate more efficiently different steps to form an entire workflow. By providing

the general overview of the project as a workflow, users could track all the operations performed and reproduce them in a more automatic way.

The first section of this chapter summarizes the evolution of this model design and the software technologies used for that. First, we started with a relational database, then we moved to a much powerful ontological database and finally we used Python to bring together the model definition and its usage in the implementation.

In the second section, the model developed for EM is analyzed in more detail. The section first covers the objects hierarchy that constitutes the basic building blocks to form other EM objects, together with their adopted conventions. In the second part, the protocol infrastructure is described, a central piece of the process model. In this section it is also described the hierarchy for the most common protocols for single-particle image processing in EM.

Finally, the implementation details of three important parts of the framework are covered: (a) the persistence mechanism, (b) the graphical interfaces and, (c) the execution engine.

4.1 Design Evolution

The initial Scipion design made use of a relational database (PostgreSQL) to represent the EM data model. Very soon we recognized that it was not a good approach due to the large number of tables required that were very difficult to maintain. Relational databases are very useful when the data model is well understood and it is known how the data will be used. However, for Scipion the developers can never quite know what requirements will be needed to accommodate the different algorithms of each software package (or for different methodologies), so high flexibility was mandatory. In this way, from an initial design in which database tables were strongly related to the different image-processing elements, we changed to a more flexible approach. The new schema reflected image-processing entities in a more abstract way using classes and its properties, that were stored in flat key/value pairs. This new approach was much more flexible to adapt to new data entities and algorithms.

The new design was implemented using the Jena-TDB ontological database (http://incubator.apache.org/jena/documentation/tdb/index.html) for metadata storage. The persistence layer used Empire (an implementation of Java Persistence API), which defines ways for accessing, persisting and managing data between Java objects and a relational databaset. This layer queried the ontological database through SPARQL,

a query language to retrieve and manipulate data stored in RDF (Resource Description Format) format. The Scipion ontology, that was created using Protégé (http://protege.stanford.edu), contained around 300 classes and a similar number of properties.

There were two main issues with this implementation that could seriously attempt against the project goals. First, it was not easy to extend with new protocols since the classes (Java or Python) were automatically generated from the ontology definition. So, adding new protocols or data objects required an entire re-build of the ontology and code generation. This was not an easy procedure to be done each time a new protocol was added. The second big problem was the performance. When a new empty project was created in Scipion, it contained a database with hundreds of thousands rows. Later on, this huge number of rows and the intermediate layers used, reduce the queries performance and detriment the user experience for simple processing tasks.

During this stage of the development of Scipion, our laboratory was also focused in the release of a new major version of Xmipp, our software package for image processing in EM. The main goal of this version was not to add new algorithms but to improve standardization, project management, reproducibility and graphical interfaces. At this point we realized that the two projects were diverging to much in terms of technology, and this implied a great duplication of resources and effort. We made the strategic decision of redirecting both projects into a common path and reusing the best ideas from both sides. In that way, we delegated the project management and graphical interfaces to Scipion, while Xmipp would handle the underlying algorithms and core libraries. This new arrangement allowed both projects to collaborate and benefit from any improvement on the other side.

Another important decision was to move from Java to Python as the language for protocol development. Python has become increasingly popular in the scientific community and it is used in many of the existing software packages in the EM community. It is relatively easy to learn and since it is interpreted shortens the time between implementation and application. As methodologies of EM structure determination are under continuous development, coding in Python makes it possible to rapidly implement, test and integrate new protocols into the Scipion framework. At the same time, it is a rich object oriented language, which provided us with the capacity of modelling different objects and their relationships such as: “A contains B” or “A is subclass of B”. For those operations that are compute-intensive we have complemented Python performance by adding bindings to C++ libraries.

4.2 Modeling the EM Domain

Domain modelling is usually split in two steps: data modelling and business processes. Data modeling involves identifying the different data items we want to store (e.g. movies) and the relationships between them. On the other hand, business processes represent the flow of data through a series of tasks (e.g. CTF estimation) that are designed to create the desired output (e.g. 3D map). Together, the data model and the business processes, conform the so-called business model. In the particular case of this work, our "business" is the whole processing pipeline needed to obtain 3D maps of macromolecular complexes from projection images. In this section, both the data model and the business processes are described.

Before describing the EM data model, it is worth to notice that we often use the same word (e.g. image) with two meanings: (1) the established one in the EM field and, (2) the particular one used by Scipion. When the latter interpretation is used the word will be written in italics and the first characters will be capital (e.g *Image*). The same convention will be followed to name other Scipion classes (e.g., *Integer*, *String*, etc).

4.2.1 Basic Objects Hierarchy

In order to create the EM data model, we have developed a set of basic Python classes that will be used as building blocks to describe complex objects. The root class of this hierarchy is the *Object* class. The *Object* class contains core attributes such as *id, name, label, comment* and *creation date* that allow to annotate objects and track relationships between different objects.

Three main classes are derived from *Object*: *Scalar*, *Set* and *Pointer*. *Scalar* class is further specialized into data types such as *String*, *Float*, *Boolean* and *Integer*. The scalar objects can easily be stored with a key/value approach. On the other hand, objects are non-scalar if they contain references to other objects. The *Pointer* class is a special type of non-scalar. It stores the id number of another object in the system, which is mainly used to refer to input objects from the protocol objects (this will be discussed later in Section 4.2.3). For example, if we have a pointer with value=100 and extended='outputVolume', this means that we can retrieve the property named 'outputVolume' from the object with id 100 in the system. Finally, the *Set* class is intended to store large number of items of the same type and with the same properties. It defines several ways to iterate over the elements or query their properties in an efficient manner (for more details see Section 4.3.1). Since in the EM data processing we usually deal with a large number of images, several classes are based on the *Set* class.

4.2.2 EM Data Model

In this section we describe how Scipion models the most important entities involved in 3DEM image processing (e.g., movies, micrographs, particles, CTFs, projection orientations, etc). These entities have been modelled as complex objects, containing other attributes that are also objects.

Images and Acquistion

Scipion understands images as the combination of binary file/s containing pixel/voxel values plus a separate collection of entries containing parametric and descriptive information (metadata). The *Image* class is the base of all image-derived classes and it is designed to store the metadata information in a flexible manner. This class contains the basic properties of an image in EM and also allows to add extra information. One of the basic properties is the image *location*, that is composed by an *index* and a *filename*, that is a reference to where the image is stored in the filesystem. Representing the location as a tuple allows to reference images that are stored in individual files, or stored together in one or several stack files.

Another crucial attribute of an image is its pixel size (referred as sampling-rate in EM), this property is stored in *Image* objects and it is properly propagated when a new output is obtained by applying some operation. The responsibility for keeping this property updated and consistent is on the developer side and the user should only input it when importing new data into a project. This approach avoids the need to input the pixel size value at different steps (as required by most EM software packages) and reduces the possibility of making mistakes.

All *Images* have also associated an *Acquisition* object, that stores several parameters related to the microscope settings: (1) microscope voltage (usually 100, 200 or 300) in kiloelectronvolt (kV); (2) magnification, (3) spherical aberration coefficient and, (4) amplitude of contrast.

In a typical EM project, we use several hundreds or even thousands of movies/micrographs and tens of hundreds of thousand particles. In consequence, we have created the class *SetOfImages*, which inherits from *Set* and contains *Image* objects as elements. This new class acts as the base for other image sets such as: *SetOfMicrographs*, *SetOfParticles*, *SetOfVolumes* and *SetOfAverages*, among others.

By inheriting from *Set*, the *SetOfImages* provides basic set functions such as adding elements, iterating over the set or accessing an element from a given id. In principle,

a set could contain any type of objects, but we have defined the *ITEM_TYPE* class property to specify the type of elements contained in each set. This property is useful, for example, when we check the type matching for selecting individual objects as input. In this way, if a user must select a *Volume*, the system allows to pick elements inside a *SetOfImages* object. The set subclasses could also define a *REP_TYPE*, that is, the class of a "representative" object of the set. We will discuss more about this property later when describing 2D and 3D classification.

Movies

The *Movie* class is a specialization of *Image* that contains the location of the movies frames stored in the hard disk and can store information related with frames alignment. By storing only the relative shifts between frames to align them, we can later obtain the aligned binary movie on demand. This allow us to avoid several copies of the input movies, which could be of a considerable size (2TB for a one day session). Even if nowadays the disk space is relatively cheap, the raw movies data size is becoming a great concern when several projects are executed simultaneously or different processing strategies are explored. Addressing this concern, the *Movie* class in Scipion allows transparent access to compressed binary files.

Micrographs and CTF

The projection images of biological samples taken in the electron microscope are modulated by a CTF. There are many ways in which the CTF can be formulated. In Scipion we have followed the convention established by the EMX project [96] that requires to store: major and minor defocus plus astigmatism angle. In addition to the minimum set of parameters required by the *CTF* class, for those algorithms that provide a richer CTF description, there is the possibility of adding extra parameters that may be latter used by another algorithm that depend on them.

Another Scipion class based on *Image* is *Micrograph*. *Micrographs* can be imported into a project or can be the result from protocols that average movie frames. In a common SPA workflow, the CTF function is estimated for each micrograph.

After estimating the CTF for a given *SetOfMicrographs*, an object of *SetOfCTFs* is created and related to the *SetOfMicrographs* object. One *SetOfMicrographs* can be related to many *SetOfCTFs* when different estimation algorithms are used. Later in the project, the user can select which of the estimations will be used for further processing.

Coordinates and Particles

After the CTF estimation step, one may discard problematic micrographs and only keep good ones to continue the processing. The next step in the pipeline is the selection of particle images from micrographs, what is usually called particle picking. The user selects coordinates (positions relative to the micrograph image) and a window size. These information will be use later to create a gallery of particles.

The *Coordinate* class contains the coordinate and a reference to the related *Micrograph* object. Optionally, *Coordinate* may store the window size, in pixels.

The *Particle* class is basically an image extracted from a micrograph at a given coordinate. *Particle* inherits from *Image* and stores a *CTF* estimated from their related *Micrograph*, this CTF may be further refined if needed.

Alignment and Classification

In single particle analysis, we want to align and average several particle images to increase the signal from the noise. In 2D there are three alignment parameters per particle: two shifts values (in x and y) and one in-plane rotation. By applying these parameters to an image, we aim to align it to a common motif. This problem is complicated since, in general, we have many motifs and in addition to align we may also need to estimate the best matching motif for a given particle. Most programs solve the problem of alignment and classification in an iterative manner. In 3D, when relating projections with 3D maps, we have to add two extra Euler angles that define the projection direction of a given particle.

The main problem with the Euler angles and shifts parameters for alignment is that there are many ways to define the rotations and displacements. For example: Positive means clockwise or counterclockwise? What are we rotating, the axis or the 3D object? What is applied first, shifts or rotations? [113]

Scipion address this problem by adopting a well-defined transformation matrix that has no ambiguities. Again, we follow the convention proposed by EMX, except for defining the 3D projection direction where we use the inverse matrix. The transformation matrix is defined as:

$$T = \begin{pmatrix} t_{11} & t_{12} & t_{13} & \Delta x \\ t_{21} & t_{22} & t_{23} & \Delta y \\ t_{31} & t_{32} & t_{33} & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The key equation to transform an image (either 2D or 3D) is:

$$f_T(r) = f(T^{-1}r)$$

where $f(r)$ represents the original 2D/3D image and $f_T(r)$ is the result of applying the transformation described in matrix T.

The result of a 2D alignment protocol is stored in Scipion as a *SetOfParticles* that contains 2D alignment information. This means that each *Particle* in the set has its corresponding *Alignment* object with the matrix needed to align the image. Similarly, a 2D classification protocol produces a *SetOfClasses2D*, that also contains particles and their alignment information but grouped into different 2D classes. Each *Class2D* inside a *SetOfClasses2D* is also a valid *SetOfParticles* where the "representative" is the average image of all particles belonging to this class.

In the 3D case, the transformation matrix relates a projection 2D image with the 3D specimen as described in 4.1. In EM, the coordinate system is selected such that the electron beam is parallel to the Z-axis and the origin is at the specimen center. Then, the relation between the collected 2D image $g_T(r)$ and the specimen $f(r)$ is:

$$g_T(r) = \int f(Tr)dz \tag{4.1}$$

Equivalent to 2D, the result of a 3D refinement method is stored in a *SetOfParticles*, but in this case the transformation matrix represents the projection direction. Similarly as in the 2D case, classification in 3D produces a *SetOfClasses3D*. *Class3D* is also a *SetOfParticles* but the set representative is a 3D volume instead of a 2D average image.

4.2.3 Process Model

Another cornerstone of our entire business model is the process model, which is complementary to the previously discussed data model. The central element of the process model in Scipion is the *Protocol* class, that can be defined as a container of algorithms and the engine for its execution. Protocol objects are composed by the same basic Python types discussed in Section 4.2.1. The main challenge when developing the protocols infrastructure was how to deal with a myriad of software packages and programs, with different conventions and file formats. At the same time, we wanted to simplify the task of adding a new protocol to the whole framework for developers and for expert users without a strong computational background.

To reach this goal, we designed the protocol infrastructure following the *Facade* and *Adapter* design patterns [114]. Both patterns address the problem of dealing with an external system that is not under our control that we want to simplify or adapt to fit our desired interface. In our specific case, the *Protocol* class acts as a wrapper over the underlying programs. On one side, each protocol should be consistent with the Scipion interface by using as inputs and output objects defined in our data model. On the other side, the protocols internally "adapt" the Scipion context to each program by converting to the required formats and calling the programs in the correct way for each case. In order to formalize the description of the process model we are going to define some of the more important concepts:

Definition 4.1. A *step* is the minimal amount of work that can be scheduled by the system. Each step could depend on other steps, for example, if a step $s2$ depends on $s1$, then $s2$ can only be executed after $s1$ is finished.

Definition 4.2. A *protocol* is defined by a tuple (I, A, S), where I is a set of input pointers, A is a list of parameters and S is the list of *steps* to be executed by the protocol. Each pointer in I can point to a specific object type. Each parameter in A has a type and the value of the parameter must be an instance of this type.

Definition 4.3. A *run* is an instance of a particular protocol P, containing a set of input objects for the pointers I in P and input values for the parameters A. After executing a run (running all steps defined in S), the output objects O are created.

Definition 4.4. A $workflow$ is a tuple (R, C), where R is a set of runs, along with a set of connections C, linking the runs. A connection $c(o, i)$ links an output object o created by run $r1$ to an input object i of another run $r2$. $r1$ and $r2$ can only be connected through connection $c(o, i)$ if the types of o and i are compatible.

In practice, to extend Scipion with a new protocol, a developer must provide the definition of I and A and implement the list of steps S that will perform the protocol task. The new protocol must inherit from the *Protocol* class and implement the method $\_defineParams$ where I and A should be defined. The steps and their dependencies should be specified in the function $\_insertAllSteps$. Each step is mapped to a protocol function and its input parameters. Usually, the last step is called *createOutputStep* where the protocol results are converted into output objects O.

It might be helpful to use an example to clarify the process of developing a new protocol. Suppose that we want to introduce a new protocol for 2D classification in Scipion that uses the program $classify\_2D$. The first task is to identify the possible inputs (I) and the outputs (O) of the protocol. Additionally, we should define the parameters (A) that controls the protocol behaviour. In this case, our input could be a $SetOfParticles$

object and the protocol will produce a $SetOfClasses2D$ as output. For simplicity, we can assume that $classify\_2D$ program requires the following parameters: (a) the number of classes (b) the particles pixel size (Å/px) and, (c) a particle mask radius (Å) to exclude background from the classification.

```
1: Inputs particles : SetOfParticles
2: Parameters: n (Number of classes), r (Particles mask radius)
3: Outputs classes : SetOfClasses2D

4: function insertAllSteps
5:     Insert step (1): convertInputStep
6:     Insert step (2): classifyStep                ▷ By default depends on (1)
7:     Insert step (3): createOutputStep            ▷ By default depends on (2)
8: end function

9: function convertInputStep(inputId)
10:    Convert particles → (particles.mrc, particles.txt)
11: end function

12: function classifyStep
13:    Read pixel size from particles: px ← particles
14:    Prepare arguments args                       ▷ Taking into account n, r and px
15:    Run program: classify_2D ← args              ▷ With the required environment
16: end function

17: function createOutputStep
18:    Create output: classes ← classes.txt
19: end function
```

Listing 1: Definition of ProtocolClassify2D

The pseudocode in Listing 1 shows the basic skeleton of a protocol class. The first thing to note here is that the code deals with the input $SetOfParticles$ but it is not aware of which protocol produced it. This means that the input object could be obtained from package X, Y or Z, and will even work with future protocols that produce that object and are not known now. Moreover, the framework will ensure that the user could only select objects of a compatible type of $SetOfParticles$ and the protocol does not need to validate it. Another point is that, despite the program $classify\_2D$ requires the pixel size as input, the protocol can get this information from the $SetOfParticles$ and it is not another parameter that the user needs to provide.

As can be seen, the protocol job is split into three steps in this particular case. The first one, *convertInputStep*, will convert the input particles from Scipion into the format (both binary file format and conventions) that is required to run this program. If the program $classify\_2D$ is inside a software suite, it is likely that the conversion routine

can be reused for other programs that also receive particles as input. The $classifyStep$ function will prepare the arguments and call the $classify\_2D$ program. Finally, the function *createOutputStep* will read the information (2D alignment and classification in this example) generated by the program to create the output object for Scipion. Analogous to the input interoperability, now the output can be used by any program that receives *SetOfClasses2D* as input.

With this design, the specificity of a program is isolated by the protocol code. The protocol will hide the details of the program: its syntax, parameters, conventions, file formats and, at the same time, will comply with a standard interface of the Scipion framework to ensure interoperability. For example, if there is a new version of the program $classify\_2D$, probably the code of the protocol needs to be updated to properly handle the changes in the new version. But those changes will not affect other protocols that still can be concatenated with this one. This is an important aspect for keeping the pace with new releases of EM software packages. In this task expert users, even if not purely software developers, can contribute with their expertize and benefit the entire community.

Worflows

Informally, a workflow is an abstract description of the steps required for executing a particular real-world process, and the flow of information between them. During this century, there has been a significant increase on the number of workflow systems focused on supporting the scientific research process in life sciences. When designing and implementing the Scipion framework, one of the tougher decision was if we should use an already established workflow manager or implement our own home-made solution. In programming, it is always a great dilemma when to write your own code (maybe at the risk of re-inventing the wheel) or using an external library or framework (also with its advantages and disadvantages). In this case, after analyzing several workflows engines, we decided to go for a simpler home-made solution since existing workflows engines introduced a high overhead and we already had basic workflow engine developed for Xmipp 3 that could be tailored to our necessities.

In Scipion, we can think of two levels of workflow management: (a) the steps inside a protocol can be considered an internal, low-level, mini-workflow engine, and (b) the concatenation of runs can be seen as a high-level, simple workflow system. Developers are more likely to interact with the former, while users will do with the latter.

On one side, developers can dynamically define the list of steps that will be executed, i.e, the step list could be different depending on the user input or the properties of the input

objects. As mentioned before, the steps are defined in the function *_insertAllSteps* and it is valid to use conditionals or loops for specifying the steps. For example, if in our previous protocol case, the program $classify\_2D$ requires that the input particles should be phase-flipped, we could add an extra step depending on that condition, as shown in pseudocode 2.

```
1: function insertAllSteps
2:     Insert step (1): convertInputStep
3:     if not particles.phaseFlipped then
4:         Insert step (1b): phaseFlipStep                ▷ Depends on (1)
5:     end if
6:     Insert step (2): classifyStep                      ▷ Depends on (1) or (1b)
7:     Insert step (3): createOutputStep                  ▷ By default depends on (2)
8: end function
```

Listing 2: Defining conditional steps

Until now, in all the examples, steps have been added without any explicit dependencies. In this case, they will automatically depend on the previous inserted step. In some cases, it is quite common to have tasks that do not depend on each other and can be executed in parallel. The dependencies between steps is an easy way for developers to define concurrent steps (more details about the implementation are given in the Section 4.3.3). Protocols that work on set of movies or micrographs can be used to illustrate this case since most of the operations are performed on each image independently of the others. For example, when extracting particles, some filters are applied to the micrographs before extracting the particles boxes. The pseudocode in Listing 3 shows how to define the steps for filtering each micrograph and extracting their particles, and that can be done in parallel for each of them. Later these steps can be executed concurrently by using either POSIX threads or MPI processes, depending on the computing environment.

```
1: function insertAllSteps
2:     Insert step (1): convertInputStep
3:     for all mic in inputMicrographs do
4:         Insert step: filterStep(mic, prerequisites = [1])   ▷ Depends on step 1
5:         Insert step: extractStep(mic)                       ▷ Depends on previous step
6:         Store last step id in allExtractIds
7:     end for
8:     Insert step (3): createOutputStep(prerequisites = allExtractIds)
9: end function
```

Listing 3: Defining steps dependencies

On the other side, users may interact with the workflow infrastructure at a higher level, without the need to care about the low-level implementation details. Users can launch a run, after selecting the input objects and parameters, and analyze its results when it is finished. By connecting the output object o of a run $r1$ with the input object i of another one, the user builds the connections C of the project pipeline or workflow. For consistency in tracking and reproducibility, after an output object produced by $r1$ is used as input of another run, $r1$ can no longer be modified, i.e., its parameters cannot be changed and the run can not be re-executed. This restriction keeps dependents runs of $r1$ consistent, since their input objects are never modified after they were created.

In order to repeat a run with slightly different parameters, we could make a "copy" of it, modify some parameters, and then execute the copied run. The same copy process can be applied to several runs at once in a given branch. This action will create a copy of all the runs selected and will re-create the connections properly. For example, if we have $r2$ that depends on $r1$, and both of them are copied, we obtain two copied runs, $r2'$ that will depends on $r1'$. The connection between $r2$ and $r1$ will also be replicated.

The same copy philosophy can be applied to export/import workflow templates. In a given project, one could select all or many runs to export them as a "workflow template", that will be stored in simple text file. The workflow template will contain the necessary information to replicate selected runs, with their parameters and the interconnections between them. If we want to apply the same processing strategy to a new dataset, we can simply export the workflow template from the previous project and import it into the new one. Once imported, the new runs will appear as new copies with the proper parameters and connections. We can execute them as usual or modify some steps of the pipeline. Since they are stored in text files, workflow templates can be easily shared with collaborators or students in the lab.

Streaming Processing

The first Scipion release was designed so that if a run $r2$ depends on an object created by a run $r1$, then $r2$ could not be launched until $r1$ finished. Soon we discovered that, under certain circumstances, this approach introduced a performance limitation. For example, it prevented to execute pre-processing steps (e.g. movie alignment or CTF estimation) while the data were been acquired.

To address this new requirement we made some changes in our data and process model. First, we introduced the property $STREAM\_MODE$ to the Set class discussed in Section 4.2.1. This property could have the value $OPEN$, when still new elements are being added to the set, or $CLOSED$, when no more elements will be added. Furthermore,

streaming protocols create the output set as soon as the first element is processed and update it with new elements later. For example, in streaming mode, the import protocol keeps monitoring the data folder to add more elements as they are acquired by the microscope.

To support the data stream processing, some protocols were modified to adopt a more *proactive* behaviour to discover new input data, process it and update their output accordingly. As illustrated in Examples 2 and 3, the developer of a protocol could add different steps depending on the parameters or the input data, but with the assumption that input objects were complete. In the case of Example 3, there are two steps added per input micrograph, but it will not process more micrographs if they are added after the *_insertAllSteps* function is executed. We added a new function, *_stepsCheck*, that will be called automatically by the underlying execution engine. This function will allow the developer of a protocol to handle incoming data items and schedule the required processing steps. Moreover, it should also detect when new items have been processed to update the output set. With this approach, protocols could be adapted to receive and pass the data stream.

The pre-processing steps that were adapted for streaming include: global and local movie alignment, dose compensation and micrographs CTF estimation. With the current developed classes, adding a new movie protocol just requires to implement a few functions and not the entire streaming mechanism. In this approach we also followed our philosophy of reducing the learning curve to add new methods into the framework. In the future, we plan to extend the streaming capability to particle picking protocols, or even go further to a preliminary 2D classification and initial model building.

To complement the streaming processing, we also developed a set of monitors, which are protocols that are checking the progress of other protocols. The monitors are designed to produce analysis plots, generate reports or raise alerts when some problems are detected. A monitor example is the CTF monitor, that checks the computed defocus values for each micrograph and can raise an alert if the values are above or below certain thresholds. Monitors were also designed in a modular way to be modified or extended. This feature was very important since we knew that different facilities will have different needs and requirements to be fulfilled.

4.2.4 Extensibility: Packages, Protocols and Viewers

Packages

As mentioned in previous chapters, Scipion was conceived with extensibility in mind. For EM we developed a basic data and process model to represent the processing pipeline, but Scipion design allows the extension of this model.

In order to extend the current EM model, it is required to provide new *packages* (referring to EM packages such as Xmipp, Relion, EMAN2, SPIDER, etc), that contains mainly protocols and viewers. The flexibility of Python facilitated the task to design packages as a plug-and-play architecture, where all the information of a package is contained in a single folder.

When a new package folder (Python module) is dropped into Scipion, the framework will automatically discover the protocols and viewers defined in that package. When there are many protocols implemented within a given package, it is useful to define some utility functions at the package level, such as converting particles from Scipion to the package format and vice-versa.

Protocols

Together with the EM data model discussed previously, we also developed several EM protocol classes that serve as the base for further extension. The classes hierarchy diagram shown in Figure 4.1 represents the main categories of protocols for EM. Protocols derived from $ProtImport$ serve to incorporate external data into a Scipion project. The subclasses of $ProtMicrographs$ represents movies and micrographs operations, while the subclasses of $ProtParticles$ contains protocols related to particles. Protocols that perform 2D analysis (such as alignment and classification) should inherit from $Prot2D$, while in 3D (inital volume, 3D refinement and classification) should do it from $Prot3D$. These classes form the base from which any other EM protocol will be derived. In some cases, the base classes contains some code to perform common tasks, facilitating the development of new protocols with similar functionality.

Viewers

Another type of objects that could be defined in the framework is the $Viewer$, which handle the visualization of other objects such as protocols or data. Viewers are also important objects since visualization and data analysis are essentials for EM processing.

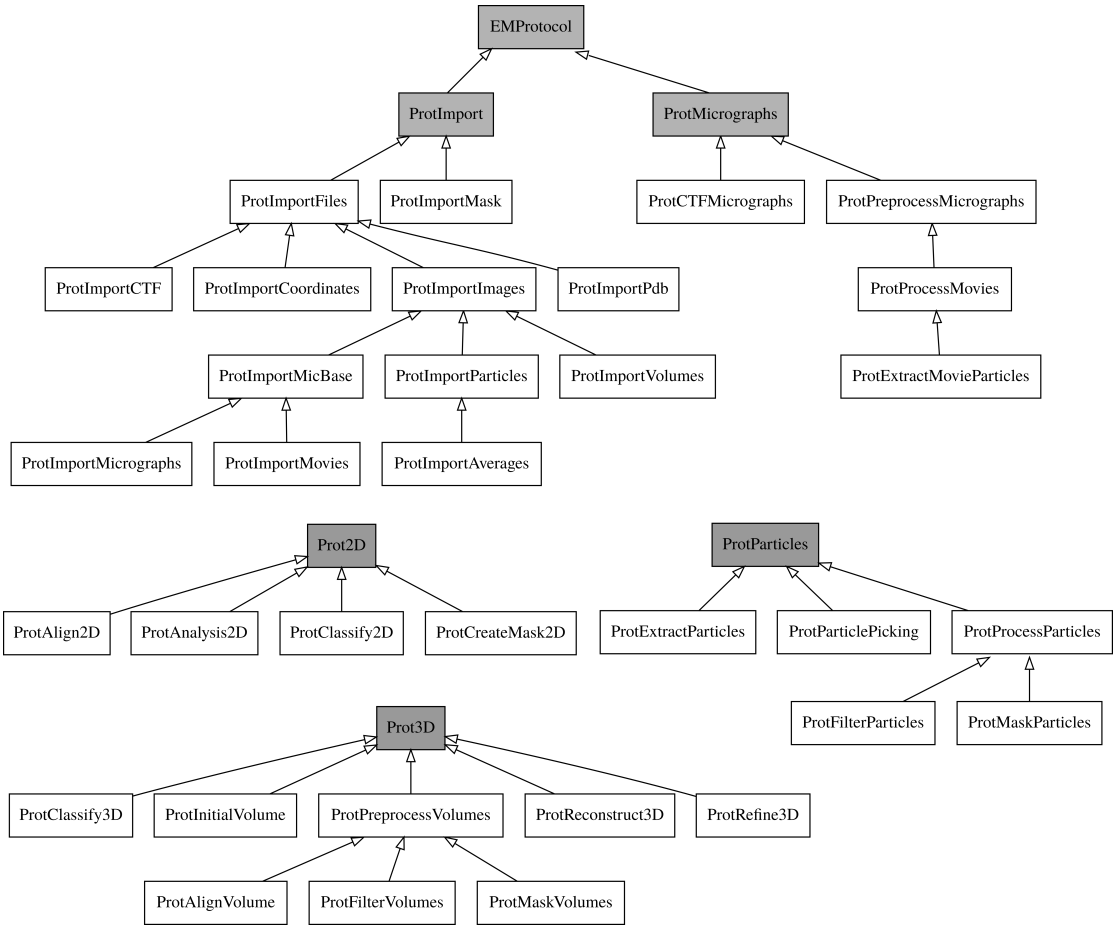

FIGURE 4.1: Protocol classes hierarchy diagram. Arrows represent "inherit from" relations. The dark boxes are the main base classes that inherit from *EMProtocol* in which protocols are grouped (some arrows are not shown for clarity).

In Scipion, a viewer class can be linked to one or several objects types via its *_targets* property. Moreover, for a given class C, more than one viewer can be defined. Then, in a given visualization context, the system can discover all viewers that are available for a given object and the user can choose among them. The viewers mechanism also allows us to quickly incorporate existing visualization tools from EM packages or from other scientific visualization applications.

Furthermore, viewer actions are encapsulated as $View$ objects to separate the logic of the data representation from the environment where it is going to be displayed. Examples of views include 2D plots, image and metadata visualization, among others (see Figure 4.2). Following this approach we are able to visualize a view object in different contexts, for example, in the desktop GUI or in a web application. Again, the developer of a protocol or a viewer, can focus on the best way to present the results and the framework will take care about delivering it in the proper context.

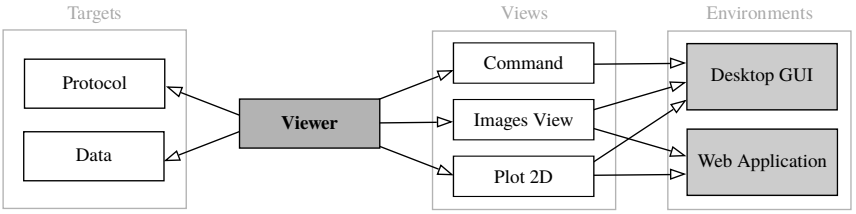

FIGURE 4.2: Diagram representing the interaction of *Viewers* with other components. *Viewers* "know" how to visualize their target classes and will produce a list of *Views*. Each *View* can then be displayed in one or more environment, such as the desktop or web application.

4.3 Implementation

This section describes the implementation of three important parts of the Scipion framework: (a) the persistence mechanism, (b) the graphical interfaces and, (c) the execution engine. Most Scipion code is written in Python since it is a very versatile scripting language. Python has been widely adopted for scientific applications and is also used by several EM packages in the field. The persistence layer uses Sqlite as the underlying storage and implements a simple ORM to work with Python objects that are mapped to the database. The execution mechanism is implemented purely in Python and it is designed to be used in high-performance environments such as multi-core machines or clusters. Graphical interfaces have been implemented to maximize modularity and usability so they can be reused in different parts of the framework. Most GUIs were implemented using the Tkinter library, which is one of the standard widget libraries for Python. Other visualization tools, base on previous Xmipp code, were developed in Java and use the ImageJ library [115] which is widely used for image processing in biological applications.

4.3.1 Automatic Data Persistence

In 3DEM, most software suites use text files for metadata storage. A less popular alternative is databases. Leginon and Appion use the Sinedon library to map Python classes onto a relational database. The underlying database was implemented with MySQL (`https://www.mysql.com/`), which requires the setup of a dedicated database server. The Sinedon library was developed using the SQLDict Python library that operates on dictionary-like objects. In EMAN2, the file-oriented Berkley Database (BDB) was used for some time, but it was removed from version 2.1 because users needed to care too much about the BDB files structure. At present, EMAN2 stores information using JSON and HDF5 files. The SPIRE layer on top of SPIDER introduced a database to stored result files, with the drawback that it is not fully synchronized with all files in

the project folder. In Xmipp 3, the SQLite library was used internally to operate on image metadata.

In Scipion, one of the design principles was to separate the conceptual model from the physical model. For achieving that goal, we looked into the object-relational mapping (ORM) concept, which allows data transfer between objects in a programming language to/from an underlying storage. In Python there are many libraries that provide ORM functionality, including SQLAlchemy (`http://www.sqlalchemy.org/`, Django's ORM (`https://www.djangoproject.com/`, Peewee (`http://docs.peewee-orm.com/`), PonyORM (`http://ponyorm.com/`) and SQLObject (`http://sqlobject.org/`), among others. Some of these libraries implement the *Active Record* pattern [116], which maps a given object into a row of the corresponding table. This pattern is very intuitive to link between the database structure and the programming objects, but have the downside that objects are too tied to the database structure. Another pattern is the *Data Mapper* [116], in which there is a clear separation between the domain and the database and the objects do not have persistence methods. Both patterns have their pros and cons, as well as using ORM tools. One of the concerns that is associated with the use of ORMs is the potential for reduced performance, together with the danger of shifting the complexity from the database to the application code.

After testing some of the above mentioned libraries we decided to implement our own mapper because performance was critical for us when dealing with large EM datasets. Furthermore, most of these ORM libraries require to use a particular data model philosophy that constrained too much the Scipion model design. We choose Sqlite, a relational database engine, with built-in functionality for querying, updating, sorting and searching. Despite we have used Sqlite as the engine to store and retrieve objects, our implementation allows to replace this mapper with other approaches, either based on another database engine or even using text files (e.g, xml or json).

In Scipion, we found two different contexts with different requirements for the ORM layer. In one hand, we wanted to store different kind of objects and their relations in a simple manner for the entire project. On the other hand, we wanted a very efficient mapper to handle huge datasets, that are common in EM processing. We have developed two different strategies that fit in each context, both of them are based on the basic objects hierarchy described in Section 4.2.1 and use SQLite as the database backend. In the following sections we describe both approaches.

SqliteMapper

SqliteMapper was the first implemented mapper for storing/retrieving objects derived from *Object* (the root class in our Python basic hierarchy). One of the main requirements for this mapper is flexibility, because *SqliteMapper* needs to deal with objects that have a variable number of attributes. For example, we have a *Particle* class that could be mapped into an equivalent table, where the basic properties of the particle are stored. If this approach is followed, it would be hard to allow different algorithms to add information to a particle (such as a quality score) specific to the method. Furthermore, we wanted to avoid a complex database with a large number of tables, which is hard to maintain and adapt to changes in the data model.

Our approach for this mapper was to create a database with a single table that could allocate all objects and their properties, which is known as the *Single Table Aggregation* pattern [117]. To implement this pattern, each object is decomposed in several rows storing key-value pairs for each of the attributes of the object. Our implementation also has some elements of the *Data Mapper* pattern, in which there is an entity (the *Mapper* in our case) that handles persistence of objects.

The algorithm for storing an object iterates over its attributes checking if the attribute is a *Scalar* or not. If it is a *Scalar*, a new row is created with its value and other properties. If not, it recursively iterates over the attribute's attributes and so on. The pseudo-code in Listing 4 summarizes the implementation of the *store* function.

```
1: function STORE(object)
2:     Insert new row for object
3:     if object is Scalar then
4:         Store object.value in row
5:     else
6:         for all attributes in object do
7:             Call STORE(attribute)
8:         end for
9:     end if
10: end function
```

LISTING 4: Pseudocode of SqliteMapper.store function

The code shown in Listing 5 illustrates how to create a *Particle* object, set some of its properties (including other objects such as *CTFModel* or *Coordinate*), and store the particle using a mapper instance. The database table generated by this code is shown in Figure 4.3. Each row contains an *id* and a *parent_id*, pointing to the parent object. Moreover, each row has a *name* property containing all ancestors *ids* and the attribute

id	parent_id	name	classname	value	label	comment	creation
1			Particle				2016-08-09 07:50:15
2	1	1._index	Integer	0			2016-08-09 07:50:15
3	1	1._filename	String	image.spi			2016-08-09 07:50:15
4	1	1._samplingRate	Float	1.5			2016-08-09 07:50:15
5	1	1._micId	Integer				2016-08-09 07:50:15
6	1	1._classId	Integer				2016-08-09 07:50:15
7	1	1._ctfModel	CTFModel				2016-08-09 07:50:15
8	7	1.7._defocusU	Float	1510.0			2016-08-09 07:50:15
9	7	1.7._defocusV	Float	1500.0			2016-08-09 07:50:15
10	7	1.7._defocusAngle	Float	120.0			2016-08-09 07:50:15
11	7	1.7._defocusRatio	Float	1.00666666666667			2016-08-09 07:50:15
12	7	1.7._psdFile	String				2016-08-09 07:50:15
13	1	1._coordinate	Coordinate				2016-08-09 07:50:15
14	13	1.13._x	Integer	1000			2016-08-09 07:50:15
15	13	1.13._y	Integer	2000			2016-08-09 07:50:15
16	13	1.13._micId	Integer				2016-08-09 07:50:15
17	13	1.13._micName	String				2016-08-09 07:50:15

FIGURE 4.3: Sqlite table generated after storing a Particle with CTFModel and Coordinate attributes.

name. The *name* property allows to retrieve in a single SQL query all the rows belonging to a root object.

```
import pyworkflow.mapper as pwmap
import pyworkflow.utils as pwutils
from pyworkflow.em.data import *

dbFn = "test.sqlite"
pwutils.cleanPath(dbFn)
mapper = pwmap.SqliteMapper(dbFn)

img = Particle()
img.setFileName("image.spi")
img.setSamplingRate(1.5)

ctf = CTFModel(defocusU=1500, defocusV=1510, defocusAngle=30)
ctf.standardize()
img.setCTF(ctf)

coord = Coordinate(x=1000, y=2000)
img.setCoordinate(coord)

mapper.store(img)
mapper.commit()
```

LISTING 5: Example of storing a Particle object using SqliteMapper

Apart from the value, the attribute *classname* is also stored in its corresponding row. By having the *classname*, the mapper can later reconstruct an object and all its properties. When loading objects, the mapper needs a dictionary to map the *classname* (which is

a string value) to the corresponding Python class. This dictionary is built by Scipion once at startup, containing all basic object classes along with data, protocol and viewer classes discovered dynamically by the system.

An additional table *Relations* is created to reflect relations between different objects. One of the most used relations is $RELATION\_SOURCE$, that relates input and output objects of a given protocol. Even if this information can be inferred from the runs and their dependencies, storing the relations makes more explicit the data provenance for further analysis.

A special type of $RELATION\_SOURCE$ is the $RELATION\_TRANSFORM$, where the developer is specifying that an input object was "transformed" into a given output. This relation is particularly useful for tracking a sequence of images derivations, since some properties could be applied to ancestors in this data-transformation tree. For example, we could have a $SetOfParticles$ S_1, then apply a filter operation to obtain a transformed set S_2. The set S_2 can be aligned and classified in 2D, from which one could create a subset (that also preserves the $RELATION\_TRANSFORM$) S_3. Since we have the transformation path $S_1 \rightarrow S_2 \rightarrow S_3$, we could extract some properties from S_3 and apply them to S_1.

Another relation used in the context of Cryo-EM is $RELATION\_CTF$. Since the CTF can be calculated with several programs for a given $SetOfMicraphs$, this one-to-many relation is tracked with $RELATION\_CTF$. When a protocol for CTF estimation finishes, a $RELATION\_CTF$ row is added that contains the $SetOfMicrographs$ and the computed $SetOfCTFs$ *id*s. This relation, in combination with $RELATION\_TRANSFORM$, allow to use the estimated CTF parameters in other related $SetOfMicrographs$ (both ancestors or descendant).

The resulting *SqliteMapper* class fulfilled our requirement of flexibility, since it allows to store and retrieve any kind of objects in a relatively simple way. This mapper is used for the two main databases of a project: *project.sqlite* and *settings.sqlite*. The first one stores information related with the different runs in the project while the second maintains tracks of the project GUI status.

SqliteFlatMapper

Since we work with many sets in a given EM project (micrographs, CTFs, particles, etc), and they could contains large number of items, we store each set separately of the project database. In this way we could load items on demand and reduce the risk of performance issues if the database grows too much. First we tried to used the same

SqliteMapper to store set elements, but soon we realized that it was not particularly well suited for this task. The main problem with *SqliteMapper* was the persistence strategy, that was focused in flexibility rather than performance. For large sets, storing several rows per object was not affordable and it also reduced the efficiency of iterating over the set. These limitations led us to implement a different mapper, *SqliteFlatMapper*, to be used for datasets management.

SqliteFlatMapper follows the *Active Record* pattern idea, where each object is mapped to a single row. Nonetheless, the implementation has some differences from the original pattern. First, the objects to be stored do not have persistence methods, this responsibility is centralized in the *Set* object, that acts as a persistence manager to hide the interaction with the underlying mapper.

This mapper shares with the *SqliteMapper* the same basic object hierarchy, but the strategy to store each object is different. In this approach, all the attributes of an object are "flattened" into a single row of the table. As mentioned before, this type of mapper is designed to store elements of a given set. The first time an object is inserted into the set, the mapper checks which columns should be created to store the object attributes. At this time, the mapper also computes the $INSERT$ and $UPDATE$ SQL statements that will be used for inserting and updating items. For each new object, its attributes are retrieved and stored into the database. This approach requires that all items stored in the set should be of the same class and contain the same attributes. Since this is normally the case for EM sets, this requirement does not represent a problem.

Usually, each *Set* uses a separate Sqlite database file to store the data. For example, a $SetOfParticles$ is commonly persisted in a *particles.sqlite* file. When storing a set, two tables are created: *Classes* and *Objects*. The *Classes* table contains the list of columns (numbered c_1, c_2, ... c_n) that will be present in the *Objects* table and their mapping to the corresponding object attributes. In this table the attribute name and class are also stored to allow re-constructing the object from their recorded values in a given row. The *Objects* table has one row per item in the set, containing the values for the object attributes.

The Figure 4.4a shows the *Classes* table of a $SetOfParticles$ database where the *label_property* column have the nested names of all attributes of a *Particle* object. It can be seen the corresponding column in the *column_name* column and the Python type in *class_name*. With this information, *Particle* objects can be retrieved from the values of each row in table *Objects* (shown in Fig. 4.4a).

In some cases, more than one set can be stored into the same physical database to avoid the need of many files. This is the case when storing $SetOfClasses2D$, where each class

id	label_property	column_name	class_name
1	self	c00	Particle
2	_index	c01	Integer
3	_filename	c02	String
4	_samplingRate	c03	Float
5	_micId	c04	Integer
6	_classId	c05	Integer
7	_ctfModel	c06	CTFModel
8	_ctfModel._defocusU	c07	Float
9	_ctfModel._defocusV	c08	Float
10	_ctfModel._defocusAngle	c09	Float
11	_ctfModel._defocusRatio	c10	Float
12	_ctfModel._psdFile	c11	String
13	_ctfModel._xmipp_ctfQ0	c12	Float
14	_ctfModel._xmipp_ctfSphericalAberration	c13	Float
15	_ctfModel._xmipp_ctfVoltage	c14	Float
16	_acquisition	c15	Acquisition
17	_acquisition._magnification	c16	Float
18	_acquisition._voltage	c17	Float
19	_acquisition._sphericalAberration	c18	Float
20	_acquisition._amplitudeContrast	c19	Float
21	_coordinate	c20	Coordinate
22	_coordinate._x	c21	Integer
23	_coordinate._y	c22	Integer
24	_coordinate._micId	c23	Integer

(A) Table *Classes* of a *SetOfParticles* database. It contains the name and class of *Particle* attributes and the corresponding colunm in the *Objects* table.

id	enabled	creation	c01	c02	c03	c04	c07	c08	c09	c10
1	1	2016-08-15...	1	Falcon_2012_06_12-14_33_35_0_movie_aligned.stk	3.54	1	34364.7	34146.6	122.565	1.00638...
2	1	2016-08-15...	2	Falcon_2012_06_12-14_33_35_0_movie_aligned.stk	3.54	1	34364.7	34146.6	122.565	1.00638...
3	1	2016-08-15...	3	Falcon_2012_06_12-14_33_35_0_movie_aligned.stk	3.54	1	34364.7	34146.6	122.565	1.00638...
4	1	2016-08-15...	4	Falcon_2012_06_12-14_33_35_0_movie_aligned.stk	3.54	1	34364.7	34146.6	122.565	1.00638...
5	1	2016-08-15...	5	Falcon_2012_06_12-14_33_35_0_movie_aligned.stk	3.54	1	34364.7	34146.6	122.565	1.00638...
6	1	2016-08-15...	6	Falcon_2012_06_12-14_33_35_0_movie_aligned.stk	3.54	1	34364.7	34146.6	122.565	1.00638...
7	1	2016-08-15...	7	Falcon_2012_06_12-14_33_35_0_movie_aligned.stk	3.54	1	34364.7	34146.6	122.565	1.00638...
8	1	2016-08-15...	8	Falcon_2012_06_12-14_33_35_0_movie_aligned.stk	3.54	1	34364.7	34146.6	122.565	1.00638...
9	1	2016-08-15...	9	Falcon_2012_06_12-14_33_35_0_movie_aligned.stk	3.54	1	34364.7	34146.6	122.565	1.00638...
10	1	2016-08-15...	10	Falcon_2012_06_12-14_33_35_0_movie_aligned.stk	3.54	1	34364.7	34146.6	122.565	1.00638...
600	1	2016-08-15...	1	Falcon_2012_06_12-16_26_22_0_movie_aligned.stk	3.54	7	28407.2	28330.8	136.447	1.00269...
601	1	2016-08-15...	2	Falcon_2012_06_12-16_26_22_0_movie_aligned.stk	3.54	7	28407.2	28330.8	136.447	1.00269...
602	1	2016-08-15...	3	Falcon_2012_06_12-16_26_22_0_movie_aligned.stk	3.54	7	28407.2	28330.8	136.447	1.00269...
603	1	2016-08-15...	4	Falcon_2012_06_12-16_26_22_0_movie_aligned.stk	3.54	7	28407.2	28330.8	136.447	1.00269...
604	1	2016-08-15...	5	Falcon_2012_06_12-16_26_22_0_movie_aligned.stk	3.54	7	28407.2	28330.8	136.447	1.00269...
605	1	2016-08-15...	6	Falcon_2012_06_12-16_26_22_0_movie_aligned.stk	3.54	7	28407.2	28330.8	136.447	1.00269...
606	1	2016-08-15...	7	Falcon_2012_06_12-16_26_22_0_movie_aligned.stk	3.54	7	28407.2	28330.8	136.447	1.00269...
607	1	2016-08-15...	8	Falcon_2012_06_12-16_26_22_0_movie_aligned.stk	3.54	7	28407.2	28330.8	136.447	1.00269...
608	1	2016-08-15...	9	Falcon_2012_06_12-16_26_22_0_movie_aligned.stk	3.54	7	28407.2	28330.8	136.447	1.00269...
609	1	2016-08-15...	10	Falcon_2012_06_12-16_26_22_0_movie_aligned.stk	3.54	7	28407.2	28330.8	136.447	1.00269...
1222	1	2016-08-15...	1	Falcon_2012_06_12-17_02_43_0_movie_aligned.stk	3.54	11	19956.9	19896.8	41.5873	1.00302...
1223	1	2016-08-15...	2	Falcon_2012_06_12-17_02_43_0_movie_aligned.stk	3.54	11	19956.9	19896.8	41.5873	1.00302...
1224	1	2016-08-15...	3	Falcon_2012_06_12-17_02_43_0_movie_aligned.stk	3.54	11	19956.9	19896.8	41.5873	1.00302...
1225	1	2016-08-15...	4	Falcon_2012_06_12-17_02_43_0_movie_aligned.stk	3.54	11	19956.9	19896.8	41.5873	1.00302...
1226	1	2016-08-15...	5	Falcon_2012_06_12-17_02_43_0_movie_aligned.stk	3.54	11	19956.9	19896.8	41.5873	1.00302...
1227	1	2016-08-15...	6	Falcon_2012_06_12-17_02_43_0_movie_aligned.stk	3.54	11	19956.9	19896.8	41.5873	1.00302...
1228	1	2016-08-15...	7	Falcon_2012_06_12-17_02_43_0_movie_aligned.stk	3.54	11	19956.9	19896.8	41.5873	1.00302...
1229	1	2016-08-15...	8	Falcon_2012_06_12-17_02_43_0_movie_aligned.stk	3.54	11	19956.9	19896.8	41.5873	1.00302...
1230	1	2016-08-15...	9	Falcon_2012_06_12-17_02_43_0_movie_aligned.stk	3.54	11	19956.9	19896.8	41.5873	1.00302...
1231	1	2016-08-15...	10	Falcon_2012_06_12-17_02_43_0_movie_aligned.stk	3.54	11	19956.9	19896.8	41.5873	1.00302...

(B) Table *Objects* of a *SetOfParticles* database, where each row contains the values for a *Particle* attributes (empty columns *c*05 and *c*06 were omitted for simplicity).

FIGURE 4.4: Example of the *Classes* and *Objects* tables of a given *SetOfParticles* database file.

has its own set of particles, but all are stored in the same *classes2d.sqlite*. For these cases, the mapper could receive an extra parameter $prefix$ for each *Set*, and will create the pair of tables $prefix\_Classes$ and $prefix\_Objects$. With this simple approach, we preserve the storing logic and we group related sets. This mechanism also allows to reuse the database connection among different mapper objects.

An important property of our design is that a change in the mapper will not affect the code of developed EM protocols. They interact with the *Set* API which is agnostic of the specific mapper used. Hiding the details of the mapper implementation allows the developer to focus in the protocol logic rather than in persistence details. For example, the code in Listing 6 shows how to load a $SetOfParticles$ from a given project. In this case, we iterate over the particles of the input set, and then we select only ten particles from each micrograph to modify the particle filename. This code creates a new set with the selected particles.

```
from pyworkflow.manager import Manager
import pyworkflow.utils as pwutils
from pyworkflow.em.data import *

manager = Manager()
project = manager.loadProject('Betagal_Tutorial')
inputSet = project.getObject(494)
dbFn = "particles.sqlite"
pwutils.cleanPath(dbFn)
partSet = SetOfParticles(filename=dbFn)
partSet.copyInfo(inputSet)
lastMic = None

for particle in inputSet.iterItems(orderBy='_micId'):
        micId = particle.getMicId()
        if micId != lastMic:
                c = 0
                lastMic = micId
        if c < 10:
                index, fn = particle.getLocation()
                newFn = pwutils.basename(fn)
                particle.setLocation(index, newFn)
                partSet.append(particle)
        c += 1
partSet.write()
partSet.close()
```

LISTING 6: Example of code that iterates over a $SetOfParticles$, modify some particle properties and creates a new subset.

4.3.2 Graphical interfaces

Project Window and Protocol Form

Users interact with Scipion through a collection of GUIs that provides a uniform interface for a plethora of heterogeneous EM programs. One of the main GUI is the project window, which is similar to a workflow editor of other systems (See Figure 4.5). The left panel contains a protocol menu that can be customized. The top-right panel shows the project workflow, and the bottom-right panel offers information about the selected run such as inputs, outputs, summary and program log. By default, flowcharts are used to represent workflows. In this representation, protocols are shown as boxes connected by lines when the output of one protocol is the input of another.

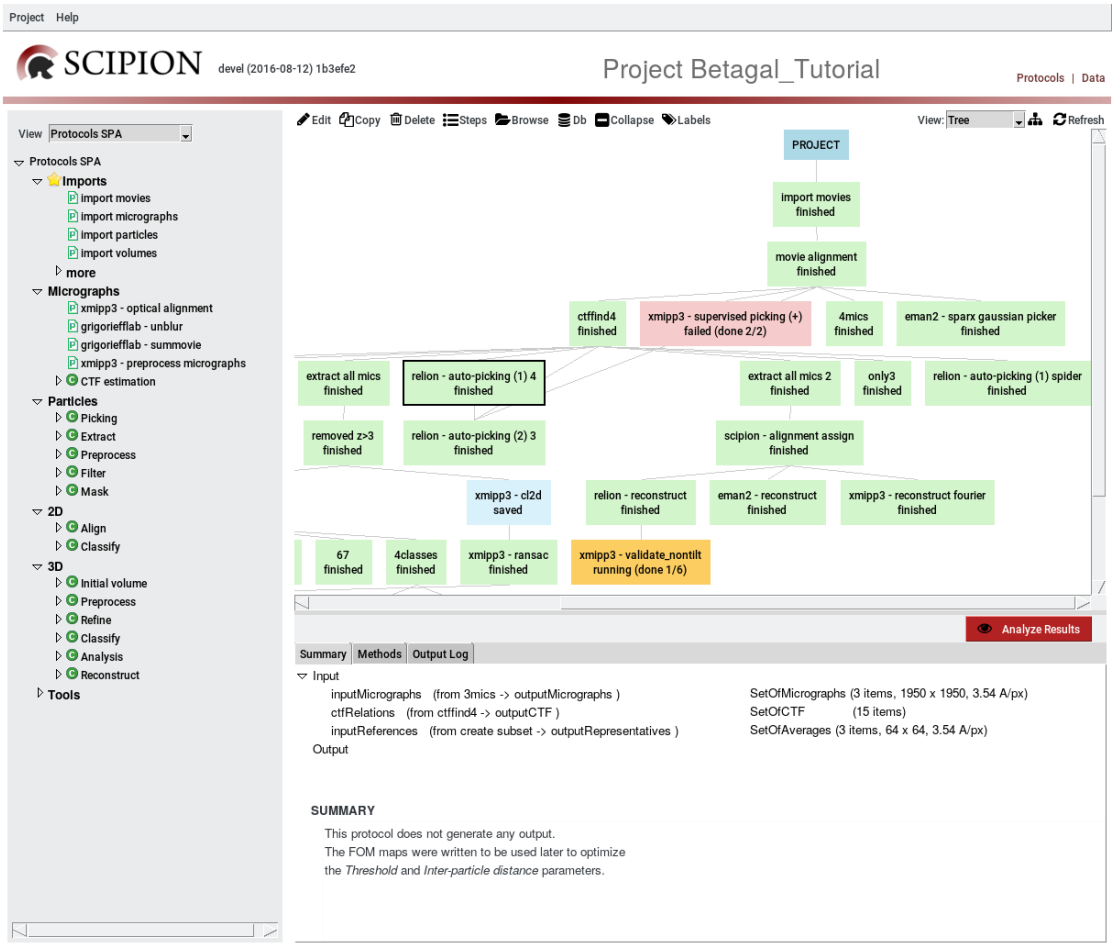

FIGURE 4.5: Scipion project window divided in three main panels: (left) protocols menu; (top-right) project workflow display as a flowchart; (bottom-right) summary of inputs and outputs, together with logs.

When clicking on a protocol box, a form is displayed (Figure 4.6A) that allows users to provide parameters to the underlying programs. Users can easily consult the references

to the methods used in the protocol as shown in Figure 4.6B or read each parameter documentation (not shown).

All the protocol forms are generated automatically from the protocol description. This means that a developer of a new protocol does not need to care about programming a new GUI. Since the type of the input objects is known from the protocol definition, Scipion can provide a selection dialog to choose only among objects in the project that have this type (Figure 4.6C). This approach avoids direct manipulation and selection of files and reduces the possibility of choosing incorrect input. Developers can easily define more conditions to be met by the input objects (e.g, particles are phase-flipped or contain CTF information). Again, none of this utilities requires an extra effort for the developer such as writing complicated SQL queries.

Moreover, the basic input types (e.g, *Integer*, *String*, *Float*) are automatically validated to avoid syntactically incorrect values. Additionally, the developer can specify other validation rules for some parameters that could prevent common mistakes. All validations are checked before launching the protocol job and, in the presence of errors, the user is notified and the execution is aborted. This approach minimize the risk of launching long jobs with incorrect parameters and wasting computing resources.

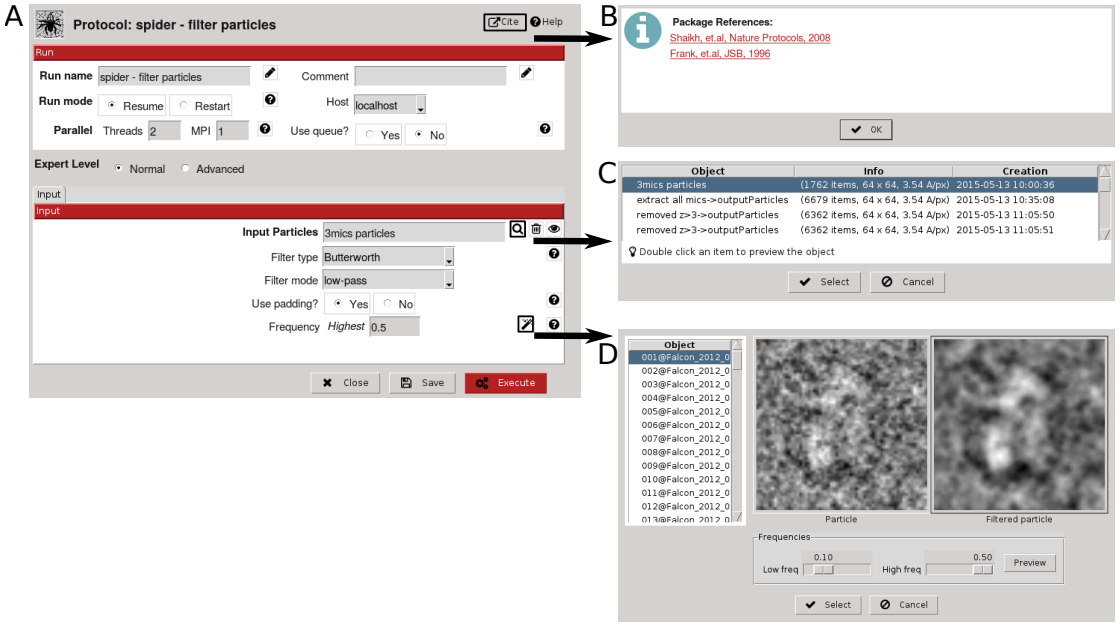

FIGURE 4.6: (A) Generated protocol form for Spider filter-particles protocol. (B) Dialog showing the citations of this protocol. (C) Dialog to browse input objects from the project database. (D) A wizard displaying the effects of the filter operation before launching the job.

Associated with the action of choosing some of the key parameters for a specific method, Scipion has "wizards", special interfaces that allow the selection of parameter values while showing their effects in real time. Fig. 4.6D shows a wizard for a SPIDER

protocol, in which the user can observe the filtered image obtained with the parameters selected before applying it to the whole set of images. Despite wizards are not generated automatically, there are several common ones that can be re-used to implement new ones without writing the code from scratch.

Visualizing Images and Metadata

One of the core graphical tools in Scipion is the application for displaying images and metadata, known as *showj*. It is a Java-based application to visualize different types of metadata and most of the EM image formats. The *showj* application was originally developed for Xmipp 3 and later extended with more Scipion specific functionality. This application also interfaces with ImageJ, a widely used image processing library in biological sciences.

The primary function of *showj* is to visualize images, in single files, stacks or referenced from metadata files (such as STAR files or Scipion Sqlite files). By default, images are displayed in a gallery view as shown in Figure 4.7. User can adjust the zoom or other display parameters to better analyze the data. The metadata (or tabular) view can be used to display images together with their associated metadata (see Figure 4.8). In this view the user can show/hide columns, change their order or sort the data by a given column.

In Scipion, the *showj* viewer is associated by default to several data types, such as $SetOfParticles$, $SetOfVolumes$, $SetOfMicrographs$ and $SetOfCTFs$, among others. Consequently, any protocol that produces this kind of output objects has a default viewer. Furthermore, the developer can customize how the output will be visualized. For example, for a particular protocol we may set the default view of a $SetOfParticles$ in metadata view, sorted by a scoring value (ascending or descending) and showing only the most relevant columns. This specification is considered a $View$, mentioned earlier in Section 4.2.3, that can also be reused in a web environment (see Section 4.3.2).

Inspecting the resulting sets to discard some elements and create a subset is a common operation in EM data processing. Currently, this operation is commonly done by manipulating text files, which can lead to mistakes. *showj* also helps with this task. The user can easily discard elements and create a new set, which will be stored as a new box in the project runs flowchart. Subsets can also be created by grouping elements after the classification protocols. Figure 4.9 shows another *showj* view of a $SetOfClasses2D$, in gallery mode and displaying the class id and the number of elements. From this view, the user can open images assigned to a given class or group several classes to create a new set.

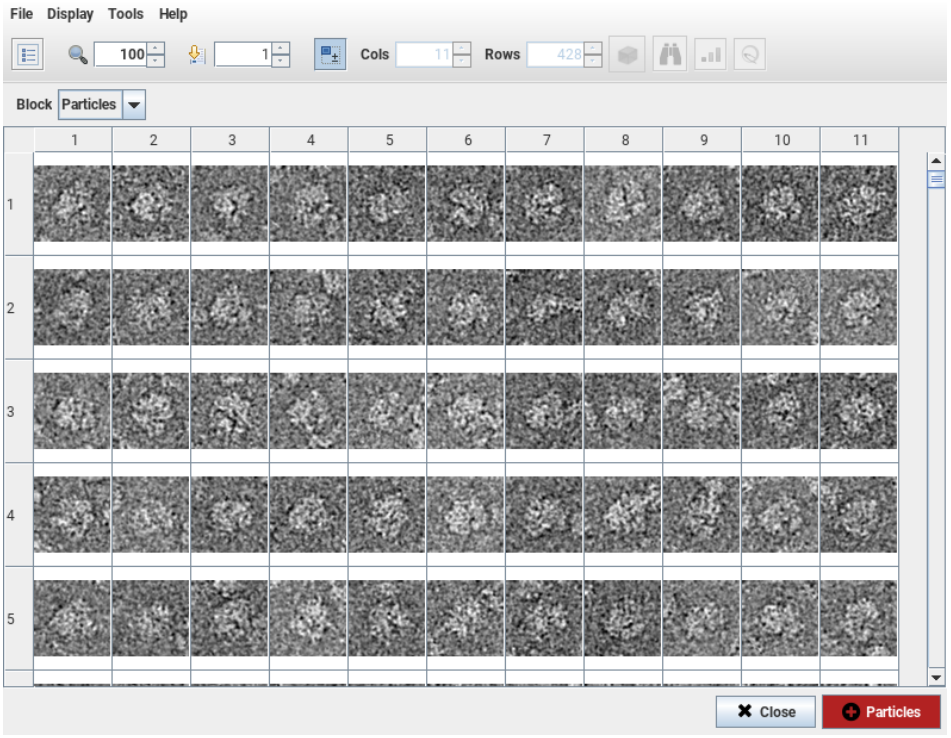

FIGURE 4.7: Screenshot of the *showj* application displaying images in the gallery mode.

Web Interface and Online Tools

Online web applications and tools are common in Bioinformatics, providing easy access to algorithms without the need of local software installation. This is not the case for EM, mainly because most of the operations involve manipulation of huge datasets that are not easy to transfer. Nevertheless, a few examples already exist for specific tasks of the processing pipeline. The European Bioinformatics Institute hosts the tilt pair validation server (`http://www.ebi.ac.uk/pdbe/emdb/validation/tiltpair/`) and the Fourier shell correlation server (`http://www.ebi.ac.uk/pdbe/emdb/validation/fsc/`). Maskiton [118] is a web interface to facilitate the creation of 2D masks and classification of aligned datasets. Another example is found for Electron Tomography, where a web application was build on top of the existing PyTom package [119, 120].

Realizing the potential of web applications, we decided to develop a web interface on top of the existing Scipion infrastructure. We basically implemented a web version of the project window in Figure 4.5 and also a web alternative to the versatile *showj* application. With these two components, we provide access to almost all processing infrastructure. All web implementation is based on the Django web framework for Python, to access the data model in the same programming language.

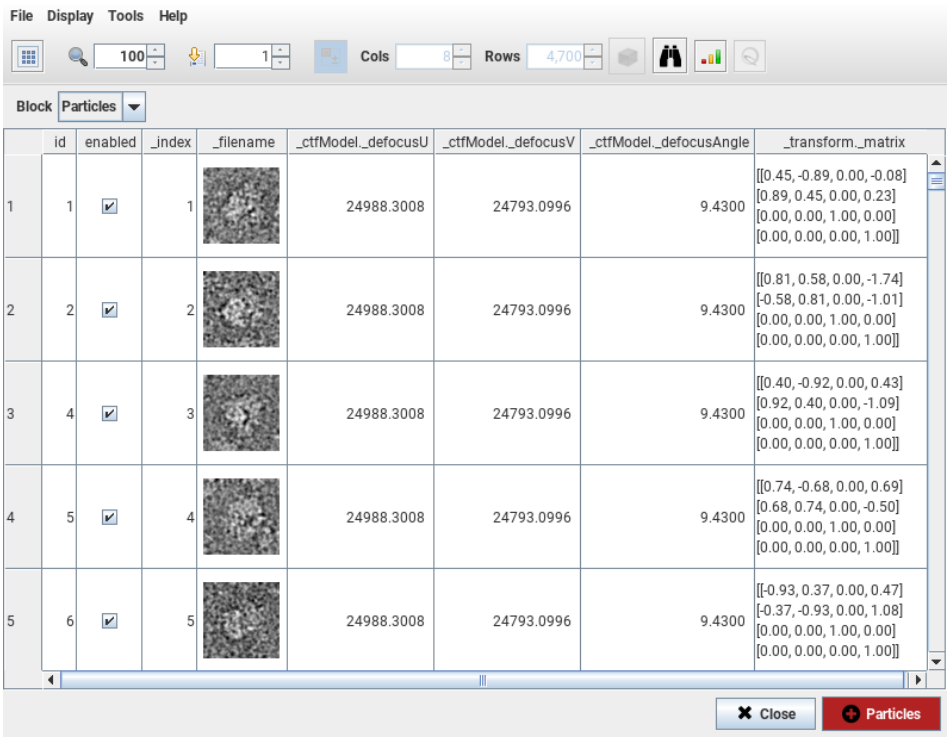

FIGURE 4.8: Visualization of images in table mode. In this mode user can select which columns are visible, which one are rendered and also sort by any column.

The modular design of *Viewers* and *Views* also allowed us to share visualization logic between the desktop and the web application. For example, if the visualization of a given protocol generates a *View* that is a Matplotlib plot, it can be rendered in both desktop and web. More challenging is the use of 3D visualization applications (such as UCSF Chimera) in web environments. The advances in web technologies like WebGL seems to be promising for such tasks, but still needs to be evaluated for the specific needs of EM data visualization.

Despite the desktop GUI being the one used for our daily work and with higher development priority, the web interface also proved its usefulness. It allowed us to develop the Scipion Web Tools (SWT), a subset of simplified EM workflows to be used online at `http://scipion.cnb.csic.es/m/services/`. The Figure 4.10 shows the entry webpage that gives access to several tools.

The list of currently implemented web tools shown in Figure 4.10 are divided in two groups: SPA tools and Reliability tools. The first group contains protocols for movie alignment, initial volume determination and resolution analysis.

In SWT, movie alignment can be performed using global and local methods [4, 6], which deliver the averaged micrographs set. For estimating the initial map, users can use e2initialmodel [47] from EMAN2, RANSAC [25] and Significant [26] from Xmipp3. Finally, local resolution can be estimated using Resmap [121]. In close collaboration with

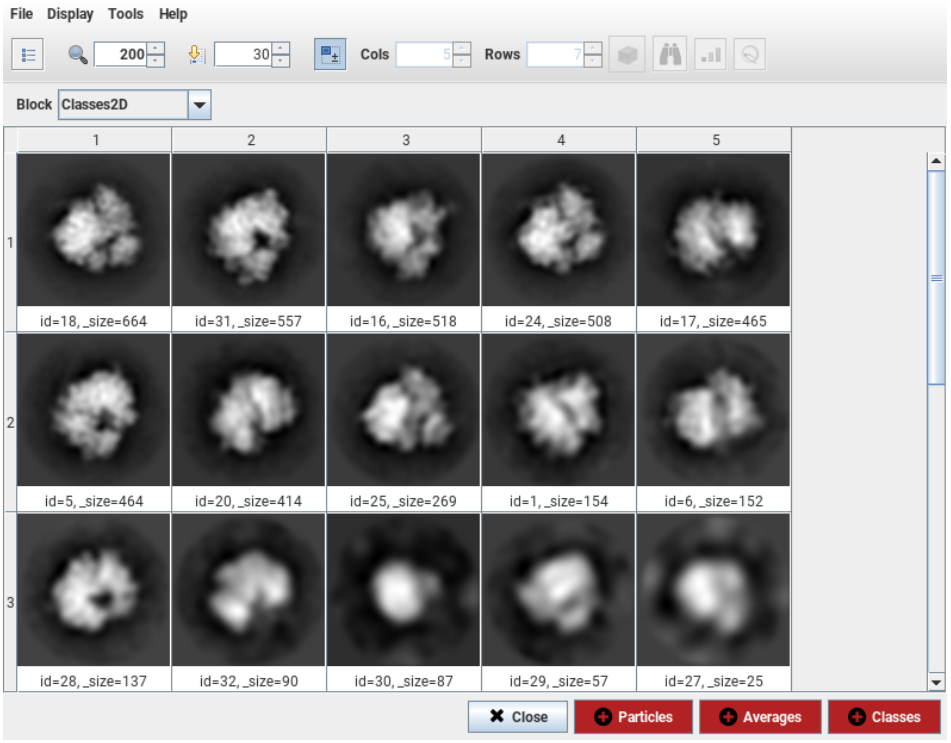

FIGURE 4.9: A *showj* visualization of a *SetOfClasses2D*. The class averages are displayed and images associated with each class can be opened. This view allows to create subset of particles by joining several class elements.

Resmap authors, the original implementation was refactored to separate the algorithm itself from the output visualization.

4.3.3 Execution engine

The whole Scipion framework heavily relies on the underlying execution engine, i.e., how jobs are setup and launched, or how protocol steps are executed. This section describes the mechanisms used by Scipion to facilitate jobs management for users, as well as for developer during protocols development.

To understand the implementation of the execution engine, it may be helpful to analyze a standard use-case. When a user opens a project, related information is loaded from the project database, together with other graphical settings. After that, the user can select any existing protocol (or create new ones), fill its input values and then execute it (Figure 4.11 1). The *Project* process will then load (or create) a protocol instance to generate the parameters form and to validate the input values.

As mentioned in previous sections, if the input validation fails for a protocol, it is not launched and the errors are shown to the user (Figure 4.11 2). If the validation is passed, *Scipion* prepares the conditions for the protocol execution. First, the project database

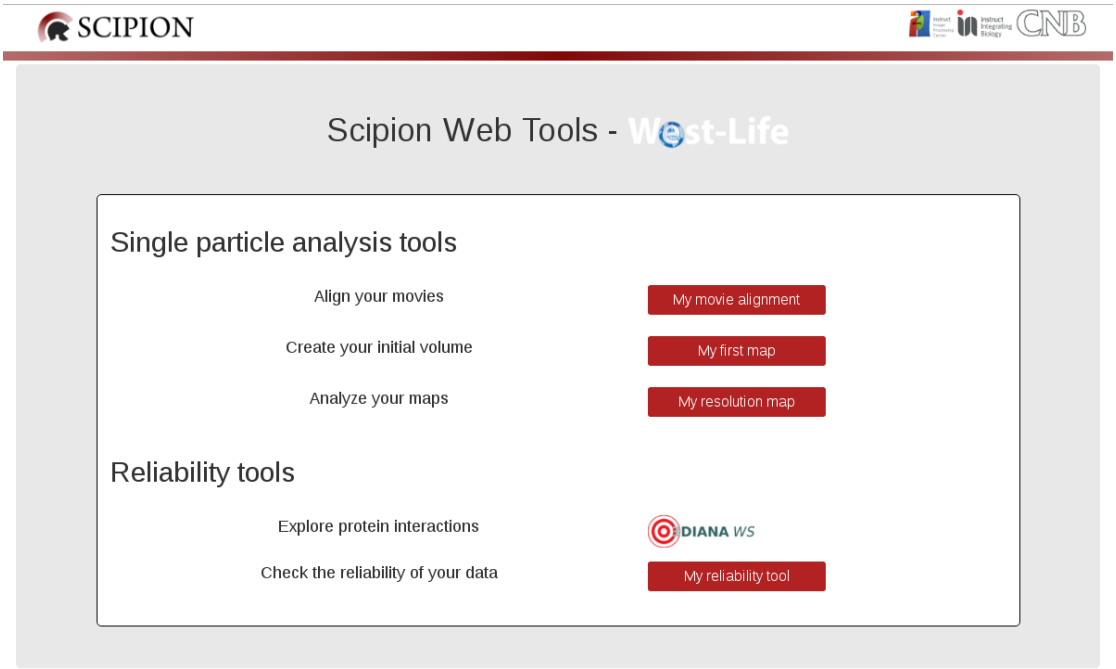

FIGURE 4.10: Webpage to access the Scipion web-tools.

is updated with the new state of this protocol (LAUNCHED). Secondly, the necessary folders are created (or cleaned up) to contain the resulting files of the protocol execution. A separate database is created for this run exclusively, which means that it will not need to access the main project database for recording its progress and results. Finally, the protocol is executed, either directly by spawning a new process or by submitting it into a job management system.

In the case of a job management system, Scipion will generate the required script to submit the job to the system. For this task, Scipion allows to configure a template script specific for each system. In this template script, some tokens will be replaced with the values provided by the user. The commands used by the system (e.g, launch or stop a job) should also be configured, for example, *sbatch*, *scancel* and *squeue* are used for the SLURM (`http://slurm.schedmd.com/`) workload manager. This approach allows users to launch jobs directly from the GUI when setting the protocol inputs, and save their time in editing manually the submission script. Another benefit is its flexibility to adapt to different parallel environments. Currently, we have installed Scipion in some of the most widely-used job management systems, such as SLURM, TORQUE (`http://www.adaptivecomputing.com/products/open-source/torque/`) or OGE (Oracle Grid Engine, previously Sun Grid Engine, `http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html`).

Each protocol is executed in an independent process, spawned either by the jobs management system or directly by the *Project* process (Figure 4.11 3). The new process

FIGURE 4.11: Activity diagram showing how protocols are executed. (A) First, the user opens a project and select a given protocol, fills its parameters and execute it. (B) Inputs selected by the user are validated and the *Project* updates the protocol information and prepares the files required for execution. The protocol process is created by the *Project* or by a jobs management system. (C) The protocol process loads its own database and starts executing the list of steps. The protocol progress is constantly monitored by the *Project*.

will load input values and the project environment from its own database. Depending on the inputs, the protocol will create the list of steps that needs to be executed. The class *StepsExecutor* will handle the execution of the steps, ensuring the proper synchronization and their dependencies.

In some cases, it is desired not to recompute all completed steps, for example, when continuing a job after an unexpected shutdown or when changing parameters that only affect some steps. Protocols have a *Resume* mode, in which Scipion only execute steps whose parameters have changed or that have not been executed previously. For each run, Scipion uses the *steps.sqlite* database to record the steps, their parameters and their status. When a protocol runs in *Resume* mode, the current list of steps is compared with the previous list (if exists) stored in *steps.sqlite* to determine which steps need to be executed. Scipion provides this re-starting mechanism to developers, who only need to care about defining relevant parameters for each *Step* function, but not about the underlying implementation.

Another built-in feature for developers is the parallel execution of steps. As discussed before, the developer can easily specify dependencies among the steps in the *_insertAllSteps* function. Step dependencies make sense when some tasks can be done simultaneously and multiple cores are available for computing. The developer can set the protocol execution mode to $STEPS\_PARALLEL$ for enabling the parallel execution. In that case, Scipion can use either POSIX threads or MPI processes to execute steps concurrently. When using threads, the $ThreadStepExecutor$ class is instantiated and it will spawn N threads (N is the number of parallel processes selected by the user) to work on the steps until the whole protocol is finished. There is a master thread that manages the work of the other threads and tracks the run progress into its database. The behaviour is similar when using MPI, but in this case N processes are created within the MPI environment. Only one of the processes will execute the protocol steps as usual, but the others will be in a "slave" mode, waiting for commands from the master process. The $MPIStepsExecutor$ class is based on $ThreadStepExecutor$, but each thread will send the commands to its associated MPI slave instead of executing it itself.

Even if a protocol execution mode is not $STEPS\_PARALLEL$, it still can use parallel computing. In EM is common that CPU-intense operations are implemented with MPI support. Then, one of the protocol steps could be a system call to spawn a MPI job. The $runJob$ function handles the creation of external processes and, by default, uses the $numberOfMpi$ value defined in the protocol for allocating the proper number of working processes. The $runJob$ function also takes into account the specification of the MPI flavor for the current environment in the Scipion configuration. Moreover, before creating an EM process from a given package, Scipion will setup the required

environment for its execution, that is independent of the user environment. This means that users do not need to modify their global environment for running EM programs of different packages. Modifying the global environment paths have the potential risk of creating conflicts between system libraries and those provided by EM packages.

Chapter 5

Results and Discussion

Even before the first Scipion manuscript was published on July of 2016, Scipion had been deployed in many production environments. Therefore, there is lot of feedback on its usage from both users and developers. In this chapter, I evaluate the impact of Scipion as an integrative framework for 3DEM image processing.

The chapter has been divided into three sections. The first two sections describe how users and developers are working with Scipion, while in the last one I provide my personal view on the present and future of the framework.

5.1 Scipion Usage: User Perspective

5.1.1 Downloads

A first measure of Scipion success is the number of downloads and installations. Since its release, Scipion has been downloaded more than 700 times. Unfortunately, it is difficult to know if it has been used for daily work or only occasionally. In order to address this lack of information, in the next release (version 1.1) Scipion will collect information on the protocols usage, if activated by the user. This information will be sent to an online service that maintains a table with the number of times that each protocol has been executed. The results of an experimental version of this service (where internal users at the CNB are filtered out) are available at http://calm-shelf-73264.herokuapp.com/.]

5.1.2 Internal Use at the CNB

Scipion has been developed at the BioComputing Unit (BCU) of the CNB. Our research group belongs to the Structural Biology Department where six groups use Electron Microscopy for their research. Therefore, the natural first targeted users were our close research fellows. Even before the first stable release, we made many internal demonstrations, and provided support. Our efforts have been rewarded with the adoption of Scipion by all the groups in our department as their primary working tool. After this, approximately 40 users with different level of expertise (PhD students, post-docs and senior researchers) have been exposed to Scipion as an image processing framework. Their testing and contributions have been crucial for the improvement and addition of many features. In the following, we describe some of the tasks carried out using Scipion in the CNB. The list is not an exhaustive collection of the works done with Scipion but rather a selection of a few projects that may provide an idea of the power and flexibility of Scipion.

Optimization of CNB Microscope Acquisition Parameters

After the Talos-Artica microscope was installed in the CNB, Dr. Daniel Luque used Scipion to run several experiments to calibrate some of the acquisition parameters and to find optimal parameters for the first steps of the workflow. As a test case, he used a Circovirus dataset acquired in the Talos microscope. Following are some of his findings:

1. **Comparison between 73k and 92k magnification for image acquisition:** The comparison was performed between a dataset of 58k particles extracted from the 73k images (calibrated pixel size of 1.37 Å/px) and another dataset of 89k particles extracted from the 92k images (nominal pixel size of 1.14 Å/px). In both cases, 31 frames were averaged using first correlation global alignment plus the optical flow local alignment. The obtained resolutions were similar at all frequencies, except for higher ones, where the 73k dataset seems to have a better resolution. Moreover, the pixel size was calibrated for the 92K dataset. It was found that from the nominal pixel size of 1.14 A/px, the calibrated one is 1.09 A/px.

2. **Comparison of the selected frame range for movie alignment:** Using the 73k images, the same set of 58k particles was re-extracted and refined using a difference range of frames. For all the tests, the alignment was done using correlation and optical flow. It was found that the maximum resolution was obtained for a subset of frames between 13 and 16.

3. **Comparison of different combination of programs for movie alignment:** The same dataset of 58k particles (with 73K magnification) was used to compare the following combinations for movie alignment: (a) correlation plus optical flow, (b) motioncor2 and, (c) motioncor2 with dose compensation. For this comparison, all frames from 1 to 31 were used for the alignment. After the experiments, it was found that both cases (a) and (b) yield similar results, while (c) was slightly better, in terms of the resolution.

The possibility to easily repeat branches from the processing pipeline greatly simplified the execution of all these experiments to calibrate the microscope and to find optimal parameters. Moreover, Scipion full track of the parameters and logs allowed a more complete analysis of the results.

Data Processing for 3DEM Map Challenge

Another important showcase of the Scipion value was its utilization for the 3DEM Map Challenge which is a community-wide challenge being sponsored by EMDataBank, started in 2015 to critically evaluate 3DEM methods that are coming into use, with the ultimate goal of developing validation criteria associated with every 3DEM map and map-derived models (`http://challenges.emdatabank.org/?q=2015_map_challenge`). Seven structures (GroEl in silico, T20S Proteasome, Apo-Ferritin, TRPV1 Channel, 80S Ribosome, Brome Mosaic Virus and β-Galactosidase) were defined as the challenge targets. These targets were based on recently described 3DEM single particle structure determinations with data collected as movies, using the latest generation of detectors.

This challenge provided us with an extraordinary scenario for testing Scipion capabilities to carry out several EM processing experiments while keeping track of the different workflows. Additionally, published structures for each dataset were available, so we could compare the results obtained with Scipion with an official reference. In our case, we were particularly interested in measuring the improvements of the final reconstruction when applying our local movie alignment method based on optical flow [6].

We submitted 12 maps, coming from six structures (all targets except Apo-Ferritin). For each structure, we obtained two maps resulting from two movie alignment strategies. In one case, we used only the motioncor (MC) program for global alignment. In the other case, after MC we also applied optical flow for local correction (MC+OF). All obtained maps were compared with its corresponding map calculated from the PDB, obtaining a FSC curve. Table 5.1 shows the results for each processed structure, all reported resolutions corresponds to FSC=0.143.

TABLE 5.1: Resolution obtained for each dataset using MC and MC+OF.

Dataset	**DDD**	**Particles**	MC/PDB	MC+OF/PDB
T20S Proteasome (10025)	K2	49954	3.06 / 2.99	3.02 / 2.97
TRPV1 Channel (10005)	K2	35645	3.35 / 3.98	3.24 / 3.86
80S Ribosome (10028)	Falcon2	105247	3.26 / 3.40	3.22 / 3.27
Brome Mosaic Virus (10010)	DE12	30000	4.20 / 4.00	4.32 / 4.15
β-Galactosidase (10013)	K2	11726	3.31 / 3.37	3.31 / 3.37

In all cases, except β-Galactosidase , maps obtained by particles coming from MC+OF alignment have a higher resolution than maps coming from MC alignment. The enhancement of the resolution when OF is applied, indicates, for most of the structures, that there are some beam-induced motion that global alignments cannot correct. This enhancement of the resolution does not depend on the DDD used to collect the data, although this enhancement is smaller for K2 cameras.

5.1.3 External Use and Collaborations

In addition to the efforts for spreading Scipion within the CNB, we have been also trying to engage external users as well. Some of these users have been introduced to Scipion during training workshops, while others by scientific collaborations with our group. The manuscript where Scipion was officially presented to the community was published in the Journal of Structural Biology on July, 2016 [122]. Until now, there are 17 citations of Scipion, although some works that used it have been published before that date. In the following, some examples of these works are briefly described.

Scipion was used in a work where the structure of a mammalian 48S initiation complex was reported at 5.8 Å resolution [123]. This work showed the relocation of subunits eIF3i and eIF3g to the 40S intersubunit face on the GTPase binding site, at a late stage in initiation. On the basis of a previous study, the authors demonstrated the relocation of eIF3b to the 40S intersubunit face, binding below the eIF2-Met-tRNAiMet ternary complex upon mRNA attachment. The analysis revealed the deep rearrangement of eIF3 and unraveled the molecular mechanism underlying eIF3 function in mRNA scanning and timing of ribosomal subunit joining.

In another work from the same group, it was presented the first full 70S ribosome structure from *Staphylococcus aureus*, a Gram-positive pathogenic bacterium, solved by cryo-electron microscopy using Scipion. This work provides the structural basis for the many studies aiming at understanding translation regulation in *Staphylococcus aureus* and for designing drugs against this often multi-resistant pathogen.

In collaboration with our group, researchers from Netherlands used the single particles workflow in Scipion to determine the asymmetric virion structure of bacteriophage MS2 [124]. It was found that, *in situ*, the viral RNA genome can form a branched network of stem-loops that are mostly allocated near the capsid inner surface, while predominantly binding to coat protein dimers that are located in one-half of the capsid.

As a last example, Scipion was also used in a work studying the structural details about the assembly of Supercomplexes of plant photosystem I with cytochrome b6f, light-harvesting complex II and NAD(P)H dehydrogenase complex [125]. The authors established, by single particle analysis, the binding position of Cytb6f at the antenna side of PSI.

5.1.4 Training and User Support

Training

A non-negligible amount of time has been spent on the dissemination of Scipion among the community through a collection of courses and workshops.

On 2013, we had the first workshop in which Scipion was used to teach image processing concepts. This workshop, held in Munich, was primarily focused in SPIDER protocols for 2D alignment and classification. Scipion clearly showcased the benefits of using a GUI such as the wizards to preview operations in an intuitive manner as opposed to writing/editing scripts.

Later, in the summer of 2014, I visited the group of Joachim Frank at Columbia University in New York. Even if Scipion was not stable at that time, we were able to install it there and use it for some projects illustrating the potential of the combination of several EM packages. During this visit, Scipion was presented in the "Fifth Annual Minisymposium on Computational Methods for Three-dimensional Microscopy Reconstruction" organized by Joachim Frank and Gabor Herman.

During 2015, several one-day practical sessions were organized in different institutions/countries such as: the Laboratory of Molecular Biology (Cambridge, UK), the Institut Génétique Biologie Moléculaire Cellulaire (Strasbourg, France), the VIB institute (Ghent, Belgium) and the National Center of Biotechnology-CSIC (Madrid, Spain). In the same year, two longer Instruct workshops also used Scipion: the "Summer School - A Practical Course in Three-Dimensional Electron Microscopy" (Brno, Czech Republic) and "IP2C hands on course on image processing applied to the structural characterization of biological macromolecules" (Madrid, Spain).

In 2016, we continued using Scipion in several events. For example, in March a one-day practical workshop was organized at the "International Symposium on Grids and Clouds" (Taipei, Taiwan). Due the success of this practical workshop, the cryo-EM session with Scipion was repeated for the same event on 2017. Another workshop was given on April of 2016 during the Instruct practical course "Advanced methods for the integration of diverse structural data with NMR data – 2nd Edition" (Utrecht, Netherlands).

Thermo Fischer (formerly FEI) started an initiative to organize an in-depth cryo-EM school with an estimated duration of 6 months, in collaboration with an academic team at NeCEN/Leiden University and our group at the CNB-CSIC. The ultimate goal was that participants become full independent practitioner in the single particle analysis (SPA) cryo-EM workflow, from sample preparation and data collection to reconstruction and molecular modelling. At the beginning 2017 the first pilot school was started, where Scipion was used as the tool for the image processing training.

User Support

As part of the I2PC role inside Instruct, we have provided image processing assistance in several projects using Scipion. A total of 28 projects were completed through the I2PC Instruct Platform, some of which have initiated long-term collaborations and publications. Moreover, there are currently 32 projects in progress. From these 32 projects, 6 are managed through Instruct, while the other 27 are related to other European infrastructure projects (iNext `http://www.inext-eu.org/` and Corbel `http://www.corbel-project.eu`). In the following, a few examples of these projects are highlighted.

In 2015, we received the visit of Ilaria Peschiera to study structural properties of the mAb-antigen and Fab-antigen by using electron microscopy. The main goal was to obtain the 3D structures of fHbp in complex with two mAbs and two Fabs. This information was used to improve the understanding of the molecular details of vaccine-induced protection against *Neisseria meningitidis*.

In another project, in collaboration with Pascal Albanese and Cristina Pagliano, the target was the Photosystem II (PSII), involved in the water splitting reaction (powered by sunlight) of the photosyntetic process. Although the X-ray crystal structure is available for PSII cores from cyanobacteria at 1.9 Å[126], for the PSII core from higher plants only an intermediate resolution (8 Å) structure has been obtained so far by electron crystallography [127, 128]. In this case, a dataset of cryo-EM images was previously acquired. The objective of the collaboration was to further refine the 3D reconstruction

of this supercomplex toward high resolution by processing a new dataset in the Scipion framework.

Moreover, we have been providing support for Scipion installations worldwide. The following list shows some of the more relevant institutions in which we have installed Scipion:

- CASPER supercomputer of the Department of Control and Computer Engineering at Politecnico di Torino (http://hpc.polito.it/)
- HPC facility of the National Center for Protein Sciences Shanghai (http://www.sibcb-ncpss.org/index.action)
- HPC in Netherlands for science and industry (SurfSARA, https://www.surf.nl/en/about-surf/subsidiaries/surfsara), that supports the Netherlands Centre for Electron Nanoscopy (NeCEN, http://www.necen.nl/)
- Diamond synchroton (http://www.diamond.ac.uk/Home.html)
- HPC at the CIC bioGUNE (http://www.cicbiogune.es/)
- HPC at Center for Cellular Imaging and NanoAnalytics, Biozentrum, University of Basel https://c-cina.unibas.ch/
- Molecular Biology Consortium at Research Triangle Park (http://www.rtp.org/), a collaborative environment supported by Duke University, UNC Chapel Hill and the National Institute of Environmental Health Science.

5.2 Scipion Usage as a Development Framework

Despite that the primary focus of Scipion is to assist users with data processing, its design was also conceived with EM developers in mind. For that reason, one of the first principles was to allow integration of new methods and tools with relatively small effort. Moreover, during the Scipion development, we have made our best to simplify developer tasks and let them concentrate in writing new algorithms. In that way, we actively collaborate with other software developers to extend Scipion functionality and take their feedback into account. In this section we describe some of the most important cases of such collaborations.

5.2.1 Hybrid Electron Microscopy and Normal Modes Analysis

In collaboration with Slavica Jonic (IMPMC, Sorbonne Universités - CNRS), we developed a method to study large-scale conformational changes by combining EM single-particle analysis and normal mode analysis (NMA). It was referred to as HEMNMA [129], which stands for hybrid electron microscopy normal mode analysis. NMA of a reference structure (atomic-resolution structure or EM volume) was used to predict possible motions that are then confronted with EM images within an automatic iterative elastic 3D-to-2D alignment procedure to identify actual motions in the imaged samples.

HEMNMA was shown to be a good approach to analyze multiple conformations of a macromolecular complex but it could not be widely used in the EM field due to a lack of an integral interface. In particular, its use required switching among different software sources as well as selecting modes for image analysis, which was difficult without a graphical interface. Thus, in a further work we developed an integral graphical interface for HEMMA [130] to simplify its use. This GUI was implemented in Xmipp 3.1 and only a small part of it relied on MATLAB. Such integration provides the user with an easy way to perform the analysis of macromolecular dynamics and forms a direct connection to the single-particle reconstruction process. In a visit to Paris in November 2014, this GUI was ported to Scipion. All graphical tools were implemented inside the framework using only open-source software, removing the dependency to MATLAB and the requirement to buy its license.

5.2.2 SPIDER Multivariate Data Analysis

One of the 2D classification workflows in SPIDER is Multivariate Data Analysis (MDA, [131]). In this approach, images are treated as points in a high-dimensional space. The basic idea behind MDA is to reduce the dimensionality of this space by representing images as a composition of a few main "principal" components, which show the most significant variations of the data. The whole workflow in SPIDER involve four main steps: (1) low-pass filtration of input images, (2) alignment in two dimensions, (3) dimensionality reduction and (4) classification.

In the traditional SPIDER processing pipeline all four steps are done using scripts, and the parameters are selected mainly by a trial-and-error approach. Moreover, editing SPIDER scripts will require users to remember cryptic commands and their parameters, which can lead to errors. Together with Dr. Tanvir Shaikh, a SPIDER expert and one of its maintainers, we integrated these procedures into Scipion.

In Scipion, users select the parameters using the GUI and the required SPIDER scripts are generated and executed under-the-hood. Furthermore, some wizards were added to preview, in real-time, operations before launching the job for the whole dataset. Data analysis was enhanced by the using of integrated visualization tools in Scipion for visualizing the results from these protocols. Another advantage in Scipion is the possibility to import all the steps of the MDA pipeline from a workflow template, which will guide the user during the processing.

5.2.3 ResMap refactoring

Resmap estimates local resolution for three-dimensional electron Cryo-EM density maps by using local sinusoidal features [121]. The algorithm has no free parameters and is applicable to other imaging modalities, including tomography. In close collaboration with Resmap authors (Dr. Alp Kucukelbir and Dr. Hemant Tagare), during a visit to US in 2014, we made a re-factoring of the code to separate the computing part from the visualization. This change allowed us to incorporate it in Scipion with the philosophy of other protocols, where the computing is done in the protocol steps while the visualization is shown in the "Analyze Results". Moreover, this modularization was convenient to also build a web interface over the existing Resmap program. This web interface is being develop as part of other "Web Tools" that provide access to some small workflows in Scipion. By having web access, users can try the algorithm without any local installation, increasing the accessibility of the method.

5.2.4 More protocols with external collaborators

During a visit to Strasbourg in 2015, we integrated gEMpicker [19], one of the first particle picker algorithms implemented in both CPU and GPU. After that visit, we started a fruitful collaboration with Dr. Grigory Sharov, from the Dr. Patrick Schultz laboratory in IGBMC. Later, the first version of the gEMpicker protocol in Scipion was improved by adding new options and enhancing parameter descriptions. Furthermore, Dr. Sharov also updated the protocol to work with a newer version of the program. As part of this collaboration, he also visited our group in the Spring of 2016, supported by the Instruct Exchange program. During his time in Madrid, we developed wizards for gEMpicker and GAutomatch picking programs. We also worked in some protocols for the MDA workflow in the IMAGIC package. The following is a summary of the contributions made by Dr. Sharov:

- **gEMpicker**: improved parameters and help information, and updated to work with newer version of the program.

- **GCtf**: integrated this algorithm for estimating the CTF in GPU (developed by Dr. Kai Zhang at the LMB in Cambridge).
- **GAutomatch**: integrated this particle picking algorithm that is implemented for GPU (also developed by Dr. Zhang).
- **MDA SPIDER**: several bug-fixes and improvement in the SPIDER MDA protocols.
- **MDA IMAGIC**: integrated some protocols for the IMAGIC MDA workflow.
- **Motioncor2**: integrated a new version of this program for frames alignment from movie images.
- **Magdistortion**: integrated programs for estimation and correction of magnification distortions in electron micrographs using images of polycrystalline samples such as gold shadowed diffraction gratings.

During the summer of 2016 we received two more visits with the aim of collaborating in new Scipion protocols.

First, Dr. Juha Huiskonen and his student Serban Ilca (from the Oxford Particle Imaging Centre, Oxford, UK) came to integrate their recently published method "Localized Reconstruction" [132], which is a general method for the localized three-dimensional reconstruction of substructures bound to a larger particle. After determining the particle orientations via conventional methods, local areas corresponding to the subunits ('subparticles') can be extracted and treated as single particles. Several utility scripts were implemented in Python that used Bsoft and Relion commands. The integration of Localized Reconstruction methodology into Scipion made the processing workflow clearer and easy to follow. Furthermore, the underlying library was modified to use the built-in functions provided by Scipion.

The second visitor was Dr. Hans Elmlund, from the ARC Centre of Excellence for Advanced Molecular Imaging , at the Department of Biochemistry and Molecular Biology of Monash University (Sidney, Australia). He is one of the main developers of SIMPLE, another innovative software package for 3DEM. SIMPLE was initially focused in *ab initio* 3D reconstruction of low-symmetry single-particles, but its functionality has been extended to cover most of the pipeline for single-particles analysis. The package contains modular command line programs, but lacks GUIs tools or project management. During the this visit, most of the programs available in SIMPLE were integrated into Scipion. As result of this visit, a Cryo-EM workshop was held in Monash University during February 2017 in some sessions were devoted to Scipion.

5.2.5 Integration with ISPyB at Diamond Synchroton

In partnership with the University of Oxford, the Electron Bio-Imaging Centre (eBIC) was established at Diamond to allow scientists to combine their techniques with many of the other cutting-edge approaches that Diamond offers. eBIC provides scientists with state-of-the-art experimental equipment and expertise in the field of Cryo-EM, for both single particle analysis and cryo-tomography.

The Diamond software team is in the process of implementing automated processing pipelines for both single particle and tomography applications. In the future this will enable users to get real-time feedback similar to that offered by the MX beamlines. Ultimately, all the important results and metadata from all user experiments will be accessible via the ISPyB database [133]. Dr. Juha Huiskonen, as a senior EM scientist from the Oxford University, helped the software team to setup some automated pipelines for EM processing.

After Dr. Huiskonen's visit to Madrid, that was his first contact with Scipion, a promising collaboration was established to evaluate Scipion's potential to be used at Diamond. A visit was scheduled to Diamond in order to install Scipion in different computing environments, with the additional challenge of the strict security policies of the synchroton. Another goal of the visit was to interconnect Scipion with their booking system, ISPyB, which is going to be extended to accommodate information about EM experiments.

Diamond –and therefore ISPyB- follows a strict policy about the use of graphical interfaces in the computing nodes: no graphic libraries are present in these nodes. This setup caused that Scipion jobs failed because the import of graphical libraries could not be satisfied. During a visit to Diamond in September 2016, in collaboration with their software team, Scipion code was refactored to avoid these issues. We centralized the code where graphic libraries are imported and made protocols execution independent of that. Communication between ISPyB and Scipion required the creation of a new protocol with three main functions:

- Access Scipion image processing protocols and gather information about them.
- Create images and metadata in the formats required by ISPyB.
- Launch a new ISPyB process that gathers the created data and metadata to update ISPyB database and GUI.

Since both Scipion and ISPyB are implemented in Python, our first approach was to access the ISPyB API directly from Scipion without launching any extra process. Unfortunately, we found an incompatibility between the Python used by ISPyB and the one

used by Scipion. ISPyB Python requires a MySql module to be able to communicate with the underlying database and makes use of the Anaconda Python distribution. We sorted out this problem by executing a monitor protocol in Scipion, that produces the necessary data and launches an external process (with ISPyB Python) to populate the underlying database.

At present Scipion is able to execute in streaming mode a small collection of protocols related with the first steps of image processing. By streaming, we mean that the data can be processed as soon as it has been acquired. In Diamond they were very interested in this feature and we planned to extend Scipion streaming capabilities to other steps in the pipeline such as particle picking or 2D classification.

Apart from Diamond, the streaming capabilities in Scipion have attracted the interest of many EM facilities to do on-the-fly processing. At the end of July of 2017, a meeting was organized in Madrid with participants from facilites such as: NeCen, CEITEC, Diamond, ESRF, SciLifeLab and from Denmark.

5.2.6 Infrastructure Projects based on Scipion

As part of the MoBrain ("A Competence Center to Serve Translational Research from Molecule to Brain" https://wiki.egi.eu/wiki/CC-MoBrain) project, our group has been involved in the Task 2: Cryo-EM in the Cloud, bringing clouds to the data. The objective of this task is to facilitate the use of Cloud Computing resources for the Cryo-EM community. To achieve this goal, we prepared and tested a Scipion-Cloud image to be used in the EGI Federated Cloud (https://www.egi.eu/federation/egi-federated-cloud/. Furthermore, this image was deployed on the SurfSARA site and used for several workshops.

In the context of the WestLife project "Bring the world of complex data analysis in Structural Biology to a simple Web browser-based Virtual Research Environment" (west-life.eu), we are integrating Scipion's Web Tools on a Virtual Research Environment. The ultimate goal of the project is to develop a web portal that provides an entry point to all services and tools.

Additionally, our group created and tested a Scipion-Cloud image (AMI) in the Amazon Cloud (AWS EC2) through funding of an Instruct Research and Development pilot project. This AMI could be used by EM users to easily instantiate their own Scipion Virtual Machine on the AWS cloud.

5.2.7 Scipion outside the Electron Microscopy Field

Scipion's execution framework can be seen as a general tool, not necessarily attached to Electron Microscopy. In particular, it is useful for any domain that can be decomposed into tasks in such a way that the output of a task is the input of the next one. This is, for example, the case of Pharmacokinetics (PK) and Pharmacodynamics (PD) modeling. PKPD is the part of pharmacology that tries to explain with mathematical models how a drug is absorbed, distributed, metabolized, and excreted. These models explain how the drug concentration evolves at the patient plasma, and at the different tissues. Then, this concentration can be translated into a drug effect, whose intensity depends on the drug concentration and/or its time evolution.

The applicability of Scipion to PKPD modeling has already been initiated by one of the members of the CNB group and it has resulted in a fork of the project which has already been used successfully in the data modeling of several pharmaceutical companies. The main web page of this project can be seen at https://github.com/cossorzano/scipion/wiki. The fork has retained the open-source nature of the project and can be freely downloaded.

5.3 Present and Future of Scipion

As can be seen, the work presented in this thesis has resulted in a useful image processing framework for both users and developers in the 3DEM community. Despite its success, there are many aspects that need to be improved. The main goal of this section is to analyze the current state of Scipion and its projection into the future. The analysis is divided in two parts: users' and developers' perspective. In both cases, I will discuss negative points that have prevented a wider adoption of the framework, as well as positive aspects that have popularized Scipion.

From the users' point of view, one of the first barriers that one may find is the natural resistance to changes: if someone is comfortable/experienced with a software package, it should have a clear reward to consider trying a new environment. We believe that Scipion offers many benefits for the users and we will continue to spread its use by providing regularly courses and workshops as well as creating a reliable product.

Other factor that can slow down Scipion spread is the current dominance of Relion in the community. One of the main advantages of Scipion is the ability to combine algorithms implemented by different packages. Therefore, if there is a single package that dominates the market this advantage is less important. Nevertheless, the field is

evolving very quickly and past experience proves that package dominance is ephemeral. Furthermore, Scipion reproducibility and traceability are two important assets that has no rival in any available 3DEM image processing package.

Another point that can be seen as negative is the fact that, after a new version of a program (or a new algorithm) is released, it may take some time until it is incorporated into Scipion. In a future, if more developers are engaged in the project, this delay can be minimized, or even removed if we work together in the release timings. Furthermore, Scipion is evolving towards a plug-in system where the release of the workflow and bookkeeping engine will be decoupled from the release of new wrappers for the different programs.

Some users also consider Scipion's management of files as an impediment. In Scipion philosophy, users should not deal with files and should only care about the processing input/outputs. This approach sometimes generates a feeling of "loss of control" about the data files. Furthermore, Scipion stores metadata in databases rather than text files. These database metadata files cannot be edited directly without some knowledge of SQL language. Unfortunately, there is no way around here. Two are the main reasons. First, many processes, sometimes related with different packages, may need to access simultaneously to the same data. Therefore, a locking systems that regulate concurrent access to data and metadata is needed. Second, metadata design should be able to provide extra parameters for some packages that are not required by other packages. Furthermore, metadata should be backwards compatible even if certain modifications needed for new software are incorporated. Although these goals may be satisfied using text files it is much easier to comply with them using a database.

Another aspect that can be improved considerably is the documentation. Scipion gives the possibility to easily associate tips with the parameters and a protocol description, but it is up to the developers to provide useful descriptions. Documentation requires a lot of time and dedication, but it is a well invested effort in the long term.

On the positive side, Scipion constitutes a unified environment that give users access to many heterogeneous tools and programs. This allows users to familiarize with a single interface and facilitates the discovery process of new algorithms and their combination. Scipion's GUI highly organizes the processing pipeline into different views, making easier to track the entire workflow, together with the inputs and outputs of each step. Scipion also take cares of project's files management and present users a higher level of abstraction.

After import, users no longer need to deal with files, but with the output from previous operations that are compatible with the input of a given one. On top of that, Scipion provides all the necessary infrastructure to ensure reproducibility.

All used parameters in a given workflow can be later inspected and reused in other projects. Transfer an entire project is as simple as copy a data folder, which may become very handy for exchanging projects if Scipion widely propagates.

On the developers side, there are also negative and positive aspects related to Scipion. For example, it can be argued that Scipion could reduce the visibility of large and established software packages, what makes their developers less willing to contribute to the framework. They can also be afraid of loosing control of their code if it is incorporated into Scipion. Moreover, for a new developer starting with Scipion, there is a learning curve that should be taken into account. Despite we have made our best to simplify the task of adding new functionality, it still requires to learn and understand the basic Scipion concepts and some of the underlying data model.

Developers of individual, less-known programs may greatly benefit from their integration into Scipion. They can rely on the built-in tools for images manipulations, plus the automatic GUI generation. This means that a developer can write a Scipion protocol (a Python wrapper around one or many programs) and it will be discovered by the framework and a GUI will be generated. It is relatively easy to extend the framework with new protocols, viewers and data objects. This feature is specially important for attracting users that have EM experience and some programming skills but not a strong computational background.

It is difficult to predict how EM software will evolve in the future. My view is that software developers will continue to add algorithms to the different EM packages, but that the burden of many operations will be shifted from packages to frameworks. Bookkeeping will require special attention to provide real tracking and reproducibility. Workflows will have a key role when exploring processing alternatives. Most algorithms will need to use distributed computing through clusters or the Cloud. Scipion is our first step in implementing an integrative framework to address important problems in the field simply and effectively.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In the following, the main objectives of this thesis are listed together with the corresponding contributions.

1. **Integration and interoperability of several EM software packages.**
    - A novel data model for the EM domain has been designed and implemented.
    - On top of this model, a large number of EM protocols have been developed, that allows to complete different SPA pipelines.
    - The data model and the underlying storage have been decoupled. A Mapper layer, that take cares of storing and retrieving any object transparently, has been implemented.
2. **Provide full traceability and support for reproducibility**.
    - Processing tasks have been modeled as protocols, with a well-defined inputs and outputs.
    - Workflows have been defined as the concatenation of several protocols that can be stored and retrieved.
    - Workflow templates can be exported from a given processing, or imported into a new project.
3. **Distributed computing and High-throughput**
    - Parallel execution of independent tasks can be automatically handled by the framework.

- Protocol encapsulation makes MPI and queue configuration independent from the underlying EM programs.
- On demand computing is possible thanks to the implementation of several Scipion cloud images.
- Streaming processing has been implemented to overlap with data acquisition and to allow a more efficient use of computing resources.

4. **Ease of Use and Extensibility**

   - A unified interface has been built that allows access to a variety of EM packages.
   - An API was developed to interact with the data and protocol models, among other components.
   - The framework has been structured to automatically discover new functionality (data types, protocols, viewers, etc).

5. **Other Achievements**

   - Scipion can be used as a general-purpose scientific processing framework (e.g., PKPD extension)

6.2 Future Work

Even if the Scipion framework has already been used for EM data processing in real projects, there are many aspects that could be further improved. For example, workflow management features could be enhanced to make the process even more friendly and intuitive. Moreover, its currently functionality focused in EM single particles analysis could be extended to others EM techniques, or even to other scientific areas. Another important aspect that needs more research and development is distributed execution, mainly due the heterogeneous nature of computing requirements and their increasing demands. In this section we will discuss some ideas of future lines of work that can be followed to extend current Scipion functionality.

Workflows

Scipion operates as a simple workflow management system, because we gave more priority to practical aspects for EM data processing than general advanced workflow features. Despite the current implemented tools provide users with many advantages over existing EM packages, there is still a lot of room for improvement in this area. Some of the possible improvements are outlined below.

- *Jobs scheduling.* A single job in Cryo-EM could take hours or days, but this is changing with recent advances in computing power (such as GPU computing) and faster algorithms. For this reason, now users may need to schedule jobs to run one after each other, expecting the computing to be done overnight. In Scipion, it is required to run manually a job after the previous one is completed. To improve usability, a very handy feature would be the possibility to schedule one or many jobs that will be executed after the required input is ready. This feature will facilitates the repetition of processing branches within a project and also improve the automation of the streaming processing.

- *Workflow Visualization.* As long as the processing pipeline becomes more complex, it is harder to draw the associated graph. This is still an open problem in computing, since there are many parameters that can be optimized for the "best" graph representation, but it is very dependent on users perception and the application domain. In Scipion we have implemented a graph drawing algorithm that places nodes in different hierarchy levels. On top of that, we implemented a collapse/expand mechanism to reduce the space of the graph for big projects, but it is still not enough. One possible addition could be the grouping/ungrouping approach to reduce the number of boxes shown. It could be also useful to implemented a zooming mechanism to easily located and move to different parts of the graph, again important mainly for big projects with a complex pipeline. Before implemented our own graph visualization widget (using Python Tkinter library), we evaluated several graph libraries, but none of them fulfilled all of our requirements. Nonetheless, for the future, we should keep an eye in new graph libraries for Python in which we could rely instead of our own implementation.

- *Workflows Repository.* One useful feature of Scipion is its ability to export/import workflow templates. In this way, users can easily reuse previous processing pipeline or share with other researchers. In order to encourage this, we could develop a workflow repository, which will basically contains a list of workflow contributed by the community. Ideally, the repository could allow to annotate each workflow to help future searches. Moreover, the import utility could be extended to communicate with this repository, either to publish new workflows or to import from it.

- *EMDB Deposition.* In the same line of action, we have started to discuss with EMDB the possibility to submit the Scipion workflow template together with other information of the deposition. In the case that a deposited 3D structure also contains the related workflow, other users could visualize it online or download it to be used for their own research. Apart from that, we could incorporate a tool to

generate part of the information required for submitting a new EMDB entry. Since Scipion have a full data provenance, we could include information from different steps of the processing.

Distributed Execution

In terms of distributed execution, Scipion has been designed to work in very heterogeneous environments. It can be used from a personal computer to a cluster of several nodes, which could be even instantiated in the Cloud. Nevertheless, there are many things that can be done in order to effectively use such diverse computing resources for EM data processing.

- *Distribute Jobs across several Nodes.* Currently, Scipion can be configured to run in a normal workstation or to submit jobs to different queue management system. Even if the projects can be moved from one machine to another, in a given moment, only a single computing "host" can be used. One idea that we plan to develop in the future is the capability to define multiple computing hosts associated to the same project. Scipion will take care about the input files that are required for a given job, transfer to the computing host and then transfer back the result files. We need to find the balance between automatically detecting input/output files and reducing the burden to protocol developers.
- *Wake-up Nodes on-demand.* In the same distributed context, we can think in a more efficient use of computing resources by allocating nodes on-demand. This idea could be specially useful if the Cloud is used as a computing infrastructure. Right now, we have deployed ready-to-use Cloud images for Amazon and EGI, but we could move one step forward by developing a layer that can instantiate Cloud machines on the fly. This means that, when a new job request some resources, this new layer could directly turn on the required machines and launch the job there. In the same way, when the job finishes, we could copy the results back and shut-down those machines.

Extending Scipion to other EM techniques

For the first version of Scipion, we have focused in the methodology of single particles analysis in 3DEM, in which our group has more experience and the most used one, if we use as metric the number of maps deposited in EMDB. Since there is some common functionality among different EM techniques, a natural expansion of Scipion would

be to add protocols for these other techniques, such as: Electron Tomography (including Sub-Tomogram Averaging), Helical Processing or the later steps of atomic model building/fitting.

- *Electron Tomography and Sub-tomogram Averaging.* This technique allows to reconstruct 3D volumes from 2D projection images of a tilted specimen. Similarly to single particles analysis, there are many copies of the same structure within a tomogram, that need to be extracted, aligned and averaged. In order to extend Scipion to provide support for this technique, we should start by creating a data model for the inputs and outputs used by different algorithms. After that, we should create the base hierarchy of protocols, that will be implemented by each of the operations in this particular pipeline.

- *Helical Processing.* The determination of helical assemblies has become a useful technique, in which the last advancements of single particle analysis has had an important impact. Nevertheless, the helical processing workflow is a bit different, where one of the most time-consuming steps is the determination of the initial symmetry parameters. For this case, we should collaborate with experienced researchers in this specific technique to include protocols and visualization tools for facilitating the processing pipeline. We should evaluate the SPRING [134] package, which already implement a workflow approach for the helical processing.

- *Model Building.* The new developments in single particles analysis have allowed to solve the structures of macromolecules at near-atomic resolution. The next logical step, given the obtained resolution, is to build an atomic model. At resolutions of 4.5Å the C-α backbone of the protein can be built based on the map alone, and at resolutions better than 4.0Å some amino-acid side chains can be traced. In Scipion, the integrated protocols goes until reconstruction and refinement of the EM model, but we have not include yet any atomic modeling program, such as COOT [135] or REFMAC [136]. The model building workflow comprises the combination of automatic and interactive tasks, that could benefit from the provenance mechanisms provided by Scipion. To use a 3D EM map in some programs, some format modifications are required, which can be encapsulated in Scipion as a proper protocol.

Architecture

From an architectural point of view, there are always points that requires further improvement, specially to make the framework more robust and better prepared for future

challenges. We have implemented Scipion as modular as we could, to allow an incremental development and enhancement process. There are two main points that need to be addressed in order to increase the current impact.

- *Image and Metadata Library.* Despite the fact that Scipion is conceptually independent of the underlying EM packages, at present, it uses Xmipp libraries for image and metadata manipulations. Since Xmipp is a complex package with many requirements, this dependende makes Scipion installation more demanding. We plan to make Scipion totally independent of Xmipp by developing a standalone IO library that will be used both by Xmipp and by Scipion.

- *Domain Discovery.* Scipion can be extended by adding new protocols under the existing EM packages (scipion/pyworkflow/em/packages/) or adding a new package. This EM packages works as plugins, the developer only needs to put a Python module there (a folder with some specific files) and the new data objects, protocols or viewers defined will be automatic discovered by the framework. We could take this idea one level up and define the entire domain (EM in this case) also as a plug-and-play component. Even multiple domains could be combined if it makes sense. This feature will allow to also reuse the basic Scipion framework in other scientific areas. To achieve this goal, some re-factoring is required to make more flexible the root directory where all entities are discovered.

Bibliography

[1] W. Kuhlbrandt. Biochemistry. The resolution revolution. *Science*, 343(6178): 1443–1444, 2014.

[2] E. H. Egelman. The current revolution in cryo-em. *Biophysical Journal*, 110(5): 1008 – 1012, 2016.

[3] J. Milne, M. Borgnia, A. Bartesaghi, E. Tran, L. Earl, D. Schauder, J. Lengyel, J. Pierson, A. Patwardhan, and S. Subramaniam. Cryo-electron microscopy – a primer for the non-microscopist. *FEBS Journal*, 280(1):28–45, 2013.

[4] X. Li, P. Mooney, S. Zheng, C.R. Booth, M. B. Braunfeld, S. Gubbens, D. A. Agard, and Y. Cheng. Electron counting and beam-induced motion correction enable near-atomic-resolution single-particle cryo-EM. *Nat. Methods*, 10(6): 584–590, 2013.

[5] S. H. W. Scheres. Beam-induced motion correction for sub-megadalton cryo-em particles. *eLife*, 3:e03665, 2014.

[6] V. Abrishami, J. Vargas, X. Li, Y. Cheng, R. Marabini, C.O.S Sorzano, and J.M. Carazo. Alignment of direct detection device micrographs using a robust optical flow approach. *J. Struc. Biol.*, 189(3):163 – 176, 2015.

[7] J. L. Rubinstein and M. A. Brubaker. Alignment of cryo-em movies of individual particles by optimization of image translations. *J. Struc. Biol.*, 192(2):188 – 195, 2015.

[8] J. A. Mindell and N. Grigorieff. Accurate determination of local defocus and specimen tilt in electron microscopy. *J. Struc. Biol.*, 142(3):334–347, 2003.

[9] C. O. Sorzano, S. Jonic, R. Núñez Ramírez, N. Boisset, and J. M. Carazo. Fast, robust, and accurate determination of transmission electron microscopy contrast transfer function. *J. Struc. Biol.*, 160(2):249–262, 2007.

[10] P. A. Penczek, J. Fang, X. Li, Y. Cheng, J. Loerke, and C. M.T. Spahn. CTER—rapid estimation of CTF parameters with error assessment. *Ultramicroscopy*, 140:9 – 19, 2014.

[11] A. Rohou and N. Grigorieff. CTFFIND4: Fast and accurate defocus estimation from electron micrographs. *J. Struc. Biol.*, 192(2):216–221, 2015.

[12] K. Zhang. Gctf: Real-time CTF determination and correction. *J. Struc. Biol.*, 193(1):1–12, 2016.

[13] D. Woolford, G. Ericksson, R. Rothnagel, D. Muller, M. J. Landsberg, R. S. Pantelic, A. McDowall, B. Pailthorpe, P. R. Young, B. Hankamer, and J. Banks. SwarmPS: Rapid, semi-automated single particle selection software. *J. Struc. Biol.*, 157(1):174 – 188, 2007.

[14] V. Abrishami, A. Zaldívar-Peraza, J. M. de la Rosa-Trevín, J. Vargas, J. Otón, R. Marabini, Y. Shkolnisky, J. M. Carazo, and C. O. S. Sorzano. A pattern matching approach to the automatic selection of particles from low-contrast electron micrographs. *Bioinformatics*, 29(19):2460–2468, 2013.

[15] K. R. Lata, P. Penczek, and J. Frank. Automatic particle picking from electron micrographs. *Ultramicroscopy*, 58(3):381 – 391, 1995.

[16] H.Chi Wong, J. Chen, F. Mouche, I. Rouiller, and M. Bern. Model-based particle picking for cryo-electron microscopy. *J. Struc. Biol.*, 145(1–2):157 – 167, 2004.

[17] A. M. Roseman. FindEM a fast, efficient program for automatic selection of particles from electron micrographs. *J. Struc. Biol.*, 145(1–2):91 – 99, 2004.

[18] N. R. Voss, C. K. Yoshioka, M. Radermacher, C. S. Potter, and B. Carragher. DoG Picker and TiltPicker: software tools to facilitate particle selection in single particle electron microscopy. *J. Struc. Biol.*, 166(2):205–213, 2009.

[19] T. V. Hoang, X. Cavin, P. Schultz, and D. W. Ritchie. gEMpicker: a highly parallel GPU-accelerated particle picking tool for cryo-electron microscopy. *BMC Structural Biology*, 13(1):1–10, 2013.

[20] P. A. Penczek, J. Zhu, and J. Frank. A common-lines based method for determining orientations for n <3 particle projections simultaneously. *Ultramicroscopy*, 63(3–4):205 – 218, 1996.

[21] T. Ogura and C. Sato. A fully automatic 3D reconstruction method using simulated annealing enables accurate posteriori angular assignment of protein projections. *J. Struc. Biol.*, 156(3):371 – 386, 2006.

[22] A. Singer, R. R. Coifman, F. J. Sigworth, D. W. Chester, and Y. Shkolnisky. Detecting consistent common lines in cryo-EM by voting. *J. Struc. Biol.*, 169(3): 312 – 322, 2010.

[23] R. R. Coifman, Y. Shkolnisky, F. J. Sigworth, and Amit Singer. Reference free structure determination through eigenvectors of center of mass operators. *Applied and Computational Harmonic Analysis*, 28(3):296 – 312, 2010.

[24] H. Elmlund, D. Elmlund, and S. Bengio. PRIME: Probabilistic initial 3d model generation for single-particle cryo-electron microscopy. *Structure*, 21(8):1299 – 1306, 2013.

[25] J. Vargas, A. L. Álvarez Cabrera, R. Marabini, J. M. Carazo, and C. O. S. Sorzano. Efficient initial volume determination from electron microscopy images of single particles. *Bioinformatics*, 30(20):2891–2898, 2014.

[26] COS Sorzano, J Vargas, JM de la Rosa-Trevin, J Oton, AL Alvarez-Cabrera, V Abrishami, E Sesmero, R Marabini, and JM Carazo. A statistical approach to the initial volume problem in single particle analysis by electron microscopy. *J. Struc. Biol.*, 189(3):213–219, 2015.

[27] N. Grigorieff. FREALIGN: High-resolution refinement of single particle structures. *J. Struc. Biol.*, 157(1):117 – 125, 2007.

[28] S. H. W. Scheres, H. Gao, M. Valle, G. T. Herman, P. Eggermont, J. Frank, and J. M. Carazo. Disentangling conformational states of macromolecules in 3D-EM through likelihood optimization. *Nat Meth*, 4(1):27–29, 2007.

[29] S. H. W. Scheres. RELION: Implementation of a bayesian approach to cryo-EM structure determination. *J. Struc. Biol.*, 180(3):519 – 530, 2012.

[30] D. Lyumkis, A. F. Brilot, D. L. Theobald, and N. Grigorieff. Likelihood-based classification of cryo-em images using FREALIGN. *J. Struc. Biol.*, 183(3):377 – 388, 2013.

[31] P. A. Penczek. Chapter Three - Resolution Measures in Molecular Electron Microscopy. In G. J. Jensen, editor, *Cryo-EM, Part B: 3-D Reconstruction*, volume 482 of *Methods in Enzymology*, pages 73 – 100. Academic Press, 2010.

[32] M. Unser, C.O.S. Sorzano, P. Thévenaz, S. Jonić, C. El-Bez, S. De Carlo, J.F. Conway, and B.L. Trus. Spectral signal-to-noise ratio and resolution assessment of 3d reconstructions. *J. Struc. Biol.*, 149(3):243 – 255, 2005.

[33] M. van Heel and M. Schatz. Fourier shell correlation threshold criteria. *Journal of Structural Biology*, 151(3):250 – 262, 2005.

[34] H. Y. Liao and J. Frank. Definition and estimation of resolution in single-particle reconstructions. *Structure*, 18(7):768 – 775, 2010.

[35] Y. Cheng, N. Grigorieff, P. A. Penczek, and T. Walz. A primer to single-particle cryo-electron microscopy. *Cell*, 161(3):438 – 449, 2015.

[36] A. Stewart and N. Grigorieff. Noise bias in the refinement of structures derived from single particles. *Ultramicroscopy*, 102(1):67 – 84, 2004.

[37] T.S. Baker and R.H. Cheng. A model-based approach for determining orientations of biological macromolecules imaged by cryoelectron microscopy. *J. Struc. Biol.*, 116(1):120 – 130, 1996.

[38] S. H. W. Sjors and S. Chen. Prevention of overfitting in cryo-em structure determination. *Nat Meth*, 9(9):853–854, 2012.

[39] R. Henderson, S. Chen, J. Z. Chen, N. Grigorieff, L. A. Passmore, L. Ciccarelli, J. L. Rubinstein, R. A. Crowther, P. L. Stewart, and P. B. Rosenthal. Tilt-pair analysis of images from a range of different specimens in single-particle electron cryomicroscopy. *J. Struc. Biol.*, 413(5):1028 – 1046, 2011.

[40] S. Chen, G. McMullan, A. R. Faruqi, G. N. Murshudov, J. M. Short, S. H. W. Scheres, and R. Henderson. High-resolution noise substitution to measure overfitting and validate resolution in 3d structure determination by single particle electron cryomicroscopy. *Ultramicroscopy*, 135:24 – 35, 2013.

[41] J. B. Heymann. Validation of 3d em reconstructions: The phantom in the noise. *AIMS Biophysics*, 2(1):21–35, 2015.

[42] J. Vargas, J. Otón, R. Marabini, J. M. Carazo, and C. O. S. Sorzano. Particle alignment reliability in single particle electron cryomicroscopy: a general approach. *Scientific Reports*, 6, 2016.

[43] R. Hegerl. A brief survey of software packages for image-processing in biological electron microscopy. *Ultramicroscopy*, (46):417 – 423, 1992.

[44] B. Carragher and P.R. Smith. Advances in computational image processing for microscopy. *J. Struc. Biol.*, 116(1):2 – 8, 1996.

[45] G. C. Lander, S. M. Stagg, N. R. Voss, A. Cheng, D. Fellmann, J. Pulokas, C. Yoshioka, C. Irving, A. Mulder, P. W. Lau, D. Lyumkis, C. S. Potter, and B. Carragher. Appion: an integrated, database-driven pipeline to facilitate EM image processing. *J. Struc. Biol.*, 166(1):95–102, 2009.

[46] J. B. Heymann and D. M. Belnap. Bsoft: image processing and molecular modeling for electron microscopy. *J. Struc. Biol.*, 157(1):3–18, 2007.

[47] G. Tang, L. Peng, P. R. Baldwin, D. S. Mann, W. Jiang, I. Rees, and S. J. Ludtke. EMAN2: an extensible image processing suite for electron microscopy. *J. Struc. Biol.*, 157(1):38–46, 2007.

[48] M. van Heel, G. Harauz, E. V. Orlova, R. Schmidt, and M. Schatz. A new generation of the IMAGIC image processing system. *J. Struct. Biol.*, 116(1): 17–24, 1996.

[49] B. Gipson, X. Zeng, Z. Y. Zhang, and H. Stahlberg. 2DX–user-friendly image processing for 2D crystals. *J. Struc. Biol.*, 157(1):64–72, 2007.

[50] D. Kimanius, B. Forsberg, S. H. W. Scheres, and E. Lindahl. Accelerated cryo-em structure determination with parallelisation using gpus in RELION-2. *eLife*, 5:e18722, 2016.

[51] D. Elmlund and H. Elmlund. SIMPLE: Software for ab initio reconstruction of heterogeneous single-particles. *J. Struc. Biol.*, 180(3):420 – 427, 2012.

[52] M. Hohn, G. Tang, G. Goodyear, P. R. Baldwin, Z. Huang, P. A. Penczek, C. Yang, R. M. Glaeser, P. D. Adams, and S. J. Ludtke. SPARX, a new environment for cryoem image processing. *J. Struc. Biol.*, 157(1):47–55, 2007.

[53] J. Frank, M. Radermacher, P. Penczek, J. Zhu, Y. Li, M. Ladjadj, and A. Leith. SPIDER and WEB: processing and visualization of images in 3d electron microscopy and related fields. *J. Struc. Biol.*, 116(1):190–9, 1996.

[54] J.M. de la Rosa-Trevín, J. Otón, R. Marabini, A. Zaldívar, J. Vargas, J.M. Carazo, and C.O.S. Sorzano. Xmipp 3.0: An improved software suite for image processing in electron microscopy. *J. Struc. Biol.*, 184(2):321 – 328, 2013.

[55] R. Smith and B. Carragher. Software tools for molecular microscopy. *J. Struc. Biol.*, 163(3):224–228, 2008.

[56] Enhancing reproducibility. *Nat Meth*, 10(5):367–367, 2013.

[57] J. M. Wicherts, D. Borsboom, J. Kats, and D. Molenaar. The poor availability of psychological research data for reanalysis. *American Psychologist*, 61(7):726–728, 2006.

[58] C. G. Begley and L. M. Ellis. Drug development: Raise standards for preclinical cancer research. *Nature*, 483(7391):531–533, 2012.

[59] P. C. A. da Fonseca, S. A. Morris, E. P. Nerou, C. W. Taylor, and E. P. Morris. Domain organization of the type 1 inositol 1,4,5-trisphosphate receptor as revealed by single-particle analysis. *Proceedings of the National Academy of Sciences*, 100(7):3936–3941, 2003.

[60] K. Hamada, A. Terauchi, and K. Mikoshiba. Three-dimensional rearrangements within inositol 1,4,5-trisphosphate receptor by calcium. *Journal of Biological Chemistry*, 278(52):52881–52889, 2003.

[61] Q. Jiang, E. C. Thrower, D. W. Chester, B. E. Ehrlich, and F. J. Sigworth. Three-dimensional structure of the type 1 inositol 1,4,5-trisphosphate receptor at 24 å resolution. *The EMBO Journal*, 21(14):3575–3581, 2002.

[62] C. Sato, K. Hamada, T. Ogura, A. Miyazawa, K. Iwasaki, Y. Hiroaki, K. Tani, A. Terauchi, Y. Fujiyoshi, and K. Mikoshiba. Inositol 1,4,5-trisphosphate receptor contains multiple cavities and l-shaped ligand-binding domains. *Journal of Molecular Biology*, 336(1):155 – 164, 2004.

[63] I. I. Serysheva, S. L. Hamilton, W. Chiu, and S. J. Ludtke. Structure of ca2+ release channel at 14 Å resolution. *Journal of Molecular Biology*, 345(3):427 – 431, 2005.

[64] R. Henderson, A. Sali, M. L. Baker, B. Carragher, B. Devkota, K. H. Downing, E. H. Egelman, Z. Feng, J. Frank, N. Grigorieff, W. Jiang, S. J. Ludtke, O. Medalia, P. A. Penczek, P. B. Rosenthal, M. G. Rossmann, M. F. Schmid, G. F. Schröder, A. C. Steven, D. L. Stokes, J. D. Westbrook, W. Wriggers, H. Yang, J. Young, H. M. Berman, W. Chiu, G. J. Kleywegt, and C. L. Lawson. Outcome of the First Electron Microscopy Validation Task Force Meeting. *Structure*, 20(2):205 – 214, 2012.

[65] S. J. Ludtke, T. P. Tran, Q. T. Ngo, V. Yu. M. Bell, W. Chiu, and I. I. Serysheva. Flexible Architecture of IP3R1 by Cryo-EM. *Structure*, 19(8):1192 – 1199, 2011.

[66] Y. Mao, L. Wang, C. Gu, A. Herschhorn, A. Désormeaux, A. Finzi, S. Xiang, and J. G. Sodroski. Molecular architecture of the uncleaved hiv-1 envelope glycoprotein trimer. *Proceedings of the National Academy of Sciences*, 110(30): 12438–12443, 2013.

[67] R. Henderson. Avoiding the pitfalls of single particle cryo-electron microscopy: Einstein from noise. *Proceedings of the National Academy of Sciences*, 110(45): 18037–18041, 2013.

[68] M. van Heel. Finding trimeric hiv-1 envelope glycoproteins in random noise. *Proceedings of the National Academy of Sciences*, 110(45):E4175–E4177, 2013.

[69] S. Subramaniam. Structure of trimeric hiv-1 envelope glycoproteins. *Proceedings of the National Academy of Sciences*, 110(45):E4172–E4174, 2013.

[70] Y. Mao, L. R. Castillo-Menendez, and J. G. Sodroski. Reply to subramaniam, van heel, and henderson: Validity of the cryo-electron microscopy structures of the hiv-1 envelope glycoprotein complex. *Proceedings of the National Academy of Sciences*, 110(45):E4178–E4182, 2013.

[71] M. A. Cianfrocco and A. E. Leschziner. Low cost, high performance processing of single particle cryo-electron microscopy data in the cloud. *Elife*, 4, 2015.

[72] J. Frank, Goldfarb W., D. Eisenberg, and T. S. Baker. Reconstruction of glutamine synthesis using computer averaging. *Ultramicroscopy*, 3:283–290, 1978.

[73] J. Frank, B. Shimkin, and H. Dowse. SPIDER a modular software system for electron image processing. *Ultramicroscopy*, 6:343–358, 1981.

[74] E.F. Pettersen, T.D. Goddard, C.C. Huang, G.S. Couch, D.M. Greenblatt, E.C. Meng, and T.E. Ferrin. Ucsf chimera—a visualization system for exploratory research and analysis. *Journal of Computational Chemistry*, 25(13):1605–1612, 2004.

[75] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[76] PyMOL. URL `https://www.pymol.org/`.

[77] W. T. Baxter, A. Leith, and J. Frank. SPIRE: the SPIDER reconstruction engine. *J. Struc. Biol.*, 157(1):56–63, 2007.

[78] S. Ludtke, P. Baldwin, and W. Chiu. EMAN: Semiautomated software for high-resolution single-particle reconstructions. *J. Struc. Bio.*, 122:82–97, 1999.

[79] QT. URL `https://www.qt.io/`.

[80] M. Frigo and S. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[81] P. D. Adams, R. W. Grosse-Kunstleve, L. W. Hung, T. R. Ioerger, A. J. McCoy, N. W. Moriarty, R. J. Read, J. C. Sacchettini, N. K. Sauter, and T. C. Terwilliger. Phenix: building new software for automated crystallographic structure determination. *Acta Crystallographica Section D-biological Crystallography*, 58:1948–1954, 2002.

[82] wxPython. URL `www.wxpython.org`.

[83] PyQt. URL https://www.riverbankcomputing.com/software/pyqt/intro.

[84] Boost.Python. URL http://www.boost.org/doc/libs/1_63_0/libs/python/doc/html/article.html.

[85] R. Marabini, I. M. Masegosa, S. Marco M. C. San Martín, J. J. Fernández, C. Vaquerizo L. G. de la Fraga, and J. M. Carazo. Xmipp: An image processing package for electron microscopy. *J. Struc. Biol.*, 116:237–240, 1996.

[86] R. Marabini and J. M. Carazo. Pattern recognition and classification of images of biological macromolecules using artificial neural networks. *Biophys. J.*, 66: 1804–1814, 1994.

[87] J. M. Carazo, F. F. Rivera, E. L. Zapata, M. Radermacher, and J. Frank. Fuzzy sets-based classification of electron microscopy images of biological macromolecules with an application to ribosomal images. *J. Microsc.*, 157: 187–203, 1990.

[88] C. O. S. Sorzano, R. Marabini, , J. Velázquez-Muriel, J. R. Bilbao-Castro, S. H. W. Scheres, and et.al. XMIPP: A new generation of an open-source image processing package for electron microscopy. *J. Struc. Biol.*, 148:194–204, 2004.

[89] S. H. W. Scheres, Rafael R. Núñez Ramírez, C. O. S. Sorzano, J. M. Carazo, and R. Marabini. Image processing for electron microscopy single-particle analysis using XMIPP. *Nat Protoc*, 3(6):977–990, 2008.

[90] Sqlite. URL https://www.sqlite.org/.

[91] FLTK Toolkit. URL http://www.fltk.org.

[92] C. Suloway, J. Pulokas, D. Fellmann, A. Cheng, F. Guerra, J. Quispe, S. Stagg, C. S. Potter, and B. Carragher. Automated molecular microscopy: the new Leginon system. *J. Struc. Biol.*, 151(1):41–60, 2005.

[93] MySQL. URL https://www.mysql.com/.

[94] PHP. URL http://www.php.net/.

[95] I. Rees, E. Langley, W. Chiu, and S. J. Ludtke. EMEN2: An object oriented database and electronic lab notebook. *Microsc Microanal*, 19(1):1–10, 2013.

[96] R. Marabini, S. J. Ludtke, S. C. Murray, W. Chiu, J. M. de la Rosa-Trevín, A. Patwardhan, J. B. Heymann, and J. M. Carazo. The Electron Microscopy eXchange (EMX) initiative. *J. Struc. Biol.*, 194(2):156–163, 2016.

[97] O. Corcho, K. Belhajjame, C. Goble, P. Alper, Y. Gil, and D. Garijo. Common Motifs in Scientific Workflows: An Empirical Analysis. In *Proceedings of the 2012 IEEE 8th International Conference on E-Science (e-Science)*, E-SCIENCE '12, pages 1–8, Washington, DC, USA, 2012. IEEE Computer Society.

[98] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Res*, 41(Web Server issue):W557–W561, 2013.

[99] E. Afgan, D. Baker, M. van den Beek, D. Blankenberg, D. Bouvier, M. Čech, J. Chilton, D. Clements, N. Coraor, C. Eberhard, B. Grüning, A. Guerler, J. Hillman-Jackson, G. Von Kuster, E. Rasche, N. Soranzo, N. Turaga, J. Taylor, A. Nekrutenko, and J. Goecks. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic Acids Research*, 44 (W1):W3, 2016.

[100] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: Visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 745–747, New York, NY, USA, 2006. ACM.

[101] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, August 2006.

[102] I. Taylor, M. Shields, I. Wang, and A. Harrison. *The Triana Workflow Environment: Architecture and Applications*. Springer, New York, Secaucus, NJ, USA, 2007.

[103] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H. Truong, A. Villazon, and M. Wieczorek. *ASKALON: A Development and Grid Computing Environment for Scientific Workflows*, pages 450–471. Springer London, London, 2007.

[104] Pipeline pilot. URL http://accelrys.com/products/pipeline-pilot/.

[105] K. Belhajjame, M. Roos, E. Garcia-Cuesta, G. Klyne, J. Zhao, D. De Roure, C. Goble, J. M. Gomez-Perez, K. Hettne, and A. Garrido. Why workflows break — understanding and combating decay in taverna workflows. In *Proceedings of*

the 2012 IEEE 8th International Conference on E-Science (e-Science), E-SCIENCE '12, pages 1–9, Washington, DC, USA, 2012. IEEE Computer Society.

[106] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and D. Haussler. The Human Genome Browser at UCSC. *Genome Res*, 12(6):996–1006, 2002.

[107] D. L. Wheeler, T. Barrett, D. A. Benson, S. H. Bryant, K. Canese, D. M. Church, M. DiCuccio, R. Edgar, S. Federhen, W. Helmberg, D. L. Kenton, O. Khovayko, D. J. Lipman, T. L. Madden, D. R. Maglott, J. Ostell, J. U. Pontius, K. D. Pruitt, G. D. Schuler, L. M. Schriml, E. Sequeira, S. T. Sherry, K. Sirotkin, G. Starchenko, T. O. Suzek, R. Tatusov, T. A. Tatusova, L. Wagner, and E. Yaschenko. Database resources of the national center for biotechnology information. *Nucleic Acids Res*, 33(Database Issue):D39–D45, 2005.

[108] E. Birney, T. D. Andrews, P. Bevan, M. Caccamo, Y. Chen, L. Clarke, G. Coates, J. Cuff, V. Curwen, T. Cutts, T. Down, E. Eyras, X. M. Fernandez-Suarez, P. Gane, B. Gibbins, J. Gilbert, M. Hammond, H. Hotz, V. Iyer, K. Jekosch, A. Kahari, A. Kasprzyk, D. Keefe, S. Keenan, H. Lehvaslaiho, G. McVicker, C. Melsopp, P. Meidl, E. Mongin, R. Pettett, S. Potter, G. Proctor, M. Rae, S. Searle, G. Slater, D. Smedley, J. Smith, W. Spooner, A. Stabenau, J. Stalker, R. Storey, A. Ureta-Vidal, K. C. Woodwark, G. Cameron, R. Durbin, A. Cox, T. Hubbard, and M. Clamp. An overview of ensembl. *Genome Res*, 14(5):925–928, 2004.

[109] D. Blankenberg, G. Von Kuster, E. Bouvier, D. Baker, E. Afgan, N. Stoler, J. Taylor, and A. Nekrutenko. Dissemination of scientific software with Galaxy ToolShed. *Genome Biol*, 15(2):403–403, 2014.

[110] D. Blankenberg, J. E. Johnson, J. Taylor, and A. Nekrutenko. Wrangling galaxy's reference data. *Bioinformatics*, 30(13):1917, 2014.

[111] E. Afgan, D. Baker, N. Coraor, H. Goto, I. M. Paul, K. D. Makova, A. Nekrutenko, and J. Taylor. Harnessing cloud-computing for biomedical research with galaxy cloud. *Nature biotechnology*, 29(11):972–974, 11 2011.

[112] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: enabling interactive multiple-view visualizations. In *VIS 05. IEEE Visualization, 2005.*, pages 135–142, 2005.

[113] C. O. S. Sorzano, R. Marabini, J. Vargas, J. Otón, J. Cuenca-Alba, A. Quintana, J. M. de la Rosa-Trevín, and J. M. Carazo. *Interchanging Geometry Conventions*

in 3DEM: Mathematical Context for the Development of Standards, pages 7–42. Springer New York, New York, NY, 2014.

[114] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN 0-201-63361-2.

[115] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri. NIH Image to ImageJ: 25 years of image analysis. *Nat Meth*, 9(7):671–675, 2012.

[116] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. ISBN 0321127420.

[117] E. Yourdon, K. Whitehead, J. Thomann, K. Oppel, and P. Nevermann. *Mainstream Objects, An Analysis and Design Approach for Business*. Prentice Hall, 1995.

[118] C. Yoshioka, D. Lyumkis, B. Carragher, and C. S. Potter. Maskiton: Interactive, web-based classification of single-particle electron microscopy images. *J. Struc. Biol.*, 182(2):155 – 163, 2013.

[119] T. Hrabe, Y. Chen, S. Pfeffer, L. K. Cuellar, A. Mangold, and F. Förster. PyTom: A python-based toolbox for localization of macromolecules in cryo-electron tomograms and subtomogram analysis. *J. Struc. Biol.*, 178(2):177 – 188, 2012.

[120] T. Hrabe. Localize.pytom: a modern webserver for cryo-electron tomography. *Nucleic Acids Research*, 43(W1):W231, 2015.

[121] A. Kucukelbir, F. J. Sigworth, and H. D. Tagare. Quantifying the local resolution of cryo-em density maps. *Nat Methods*, 11:63–65, 2014.

[122] J.M. de la Rosa-Trevín, A. Quintana, L. del Cano, A. Zaldívar, I. Foche, J. Gutiérrez, J. Gómez-Blanco, J. Burguet-Castell, J. Cuenca-Alba, V. Abrishami, J. Vargas, J. Otón, G. Sharov, J.L. Vilas, J. Navas, P. Conesa, M. Kazemi, R. Marabini, C.O.S. Sorzano, and J.M. Carazo. Scipion: A software framework toward integration, reproducibility and validation in 3d electron microscopy. *J. Struc. Biol.*, 195(1):93 – 99, 2016.

[123] A. Simonetti, J. Brito Querido, A. G. Myasnikov, E. Mancera-Martinez, A. Renaud, L. Kuhn, and Y. Hashem. eif3 peripheral subunits rearrangement after mrna binding and start-codon recognition. *Molecular Cell*, 63(2):206–217, 2016.

[124] R. I. Koning, J. Gomez-Blanco, I. Akopjana, J. Vargas, A. Kazaks, K. Tars, J. M. Carazo, and A. J. Koster. Asymmetric cryo-em reconstruction of phage ms2 reveals genome structure in situ. *Nature Communications*, 7:12524, 2016.

[125] K.N.S. Yadav, D. A. Semchonok, L. Nosek, R. Kouřil, G. Fucile, E. J. Boekema, and L. A. Eichacker. Supercomplexes of plant photosystem i with cytochrome b6f, light-harvesting complex II and NDH. *Biochimica et Biophysica Acta (BBA) - Bioenergetics*, 1858(1):12 – 20, 2017.

[126] Y. Umena, K. Kawakami, J. Shen, and N. Kamiya. Crystal structure of oxygen-evolving photosystem ii at a resolution of 1.9[thinsp]a. *Nature*, 473(7345): 55–60, 2011.

[127] K. Rhee, E. P. Morris, J. Barber, and W. Kuhlbrandt. Three-dimensional structure of the plant photosystem ii reaction centre at 8 Å resolution. *Nature*, 396(6708):283–286, 1998.

[128] B. Hankamer, E. Morris, J. Nield, C. Gerle, and J. Barber. Three-dimensional structure of the photosystem ii core dimer of higher plants determined by electron microscopy. *J. Struc. Biol.*, 135(3):262 – 269, 2001.

[129] Q. Jin, C. O. S. Sorzano, J. M. de la Rosa-Trevín, J. R. Bilbao-Castro, R. Núñez-Ramírez, O. Llorca, F. Tama, and S. Jonic. Iterative Elastic 3D-to-2D Alignment Method Using Normal Modes for Studying Structural Dynamics of Large Macromolecular Complexes. *Structure*, 22(3):496–506, 2 2014.

[130] C. O. S. Sorzano, J. M. de la Rosa-Trevín, F. Tama, and S. Jonić. Hybrid electron microscopy normal mode analysis graphical interface and protocol. *J. Struc. Biol.*, 188(2):134 – 141, 2014.

[131] T. R. Shaikh, H. Gao, W. T. Baxter, F. J. Asturias, N. Boisset, A. Leith, and J. Frank. SPIDER image processing for single-particle reconstruction of biological macromolecules from electron micrographs. *Nat. Protocols*, 3(12): 1941–1974, 2008.

[132] S. L. Ilca, A. Kotecha, X. Sun, M. M. Poranen, D. I. Stuart, and J. T. Huiskonen. Localized reconstruction of subunits from electron cryomicroscopy images of macromolecular complexes. *Nature Communications*, 6:8843, 2015.

[133] S. Delageniere, P. Brenchereau, L. Launer, A.W. Ashton, R. Leal, S. Veyrier, J. Gabadinho, E.J. Gordon, S.D. Jones, K.E. Levik, S.M. McSweeney, S. Monaco, M. Nanao, D. Spruce, O. Svensson, M.A. Walsh, and G.A. Leonard. ISPyB: an information management system for synchrotron macromolecular crystallography. *Bioinformatics*, 27(22):3186–3192, 2011.

[134] Ambroise D., Rodolfo C., Irina G., and C. Sachse. SPRING – an image processing package for single-particle based helical reconstruction from electron cryomicrographs. *J. Struc. Biol.*, 185(1):15 – 26, 2014.

[135] P. Emsley, B. Lohkamp, W. G. Scott, and K. Cowtan. Features and development of coot. *Acta Crystallographica Section D - Biological Crystallography*, 66: 486–501, 2010.

[136] G. N. Murshudov, P. Skubák, A. A. Lebedev, N. S. Pannu, R. A. Steiner, R. A. Nicholls, M. D. Winn, F. Long, and A. A. Vagin. REFMAC5 for the refinement of macromolecular crystal structures. *Acta Crystallogr D Biol Crystallogr*, 67(Pt 4):355–367, 2011.