Deliverable D7.5

Project Title:	World-wide E-infrastructure for structural biology	
Project Acronym:	West-Life	
Grant agreement no.:	675858	
Deliverable title:	A HADDOCK server for EM	
WP No.	7	
Lead Beneficiary:	1: UU	
WP Title	Joint research	
Contractual delivery date:	Month 30	
Actual delivery date:	Month 31	
WP leader:	JOSE MARIA CARAZO	CSIC
Contributing partners:	UU	

Deliverable written by Mikael Trellet and edited by Alexandre Bonvin

Contents

1 Executive summary

The Joint Research Activity of West-Life is aimed at exploring new ways to use existing or close to existing services so that broader user communities will be reached.
Based on the capability of HADDOCK software to handle EM maps as input to drive the docking, and in the objective of interconnecting services that manipulate EM data, deliverable 7.5 is presenting a new version of the HADDOCK webserver. This new version, beyond introducing the processing of EM maps as input, together with EM-related parameters, has also been built on new technologies to improve user experience and administration by UU.
The complete rewrite of the web portal framework started from the observation that adding new features with the former framework was time-consuming and with limited possibilities to improve user experience on the website. Moreover, protocols and technologies used within the former framework started to be outdated and prevented the deployment of webserver instances on new machines with up-to-date systems.
The new web server is based on python Flask framework, a well-known framework (~22.000 questions on StackOverflow) that allows for website fast-prototyping. It is up and running is its development version on a local machine accessible at https://nestor.science.uu.nl/haddock. Main basic together with EM-related features are available through this URL but new features are added on a daily basis.
We detail in this deliverable the workflow of the new webserver, from the technical details of the implementation to the end-product feature as seen by the users.

2 Project objectives

With this deliverable, the project has reached or the deliverable has contributed to the following objectives:

No.	Objective	Yes	No
1	Provide analysis solutions for the different Structural Biology approaches	X	
2	Provide automated pipelines to handle multi-technique datasets in an integrative manner	X	
3	Provide integrated data management for single and multi-technique projects, based on existing e-infrastructure		X
4	Foster best practices, collaboration and training of end users		X

3 Detailed report on the deliverable

3.1 Background

HADDOCK is a data-driven approach for the modelling of molecular complexes [1]. Its capability to use experimental data to drive the docking process interfaces it with many experimental data types. NMR chemical shifts, mutagenesis data, crosslinks, etc. are some of the data that can be used as input to HADDOCK. The last years have seen the cryo-electron microscopy (cryo-EM) field develop into a powerful structural biology technique due to various technological and software developments. The resulting density maps, even if they don't reach atomistic resolution, can indicate the spatial positioning and orientation of several molecules interacting with each other. We have previously demonstrated in [2] that it is possible, given an EM map and an atomistic model, to fit the atomistic model within the density map with a relative high accuracy. A webserver, based on the software developed in the mentioned study is available under the West-Life portfolio (https://milou.science.uu.nl/services/POWERFIT).

Partially based on this work, UU published a protocol dedicated to the use of Electron-Microscopy (EM) maps as input for HADDOCK [3]. This protocol allows to use cryo-EM maps as spatial anchors for the atomistic models of the proteins to be docked. As a first step, the fitting algorithm used in PowerFit identifies centroids' coordinates, corresponding to the most likely location of the center-of-mass of the fitted models. Those centroids are then used in HADDOCK to define distance restraints that will drive the proteins towards their potential position in the map. Later on in the HADDOCK pipeline, the models are directly refined against the cryo-EM map, using cross-correlation values to assess the quality of the orientation/positioning of the proteins into the map.

As of today, this specific protocol was only accessible when using a local version of HADDOCK and not supported by the current web server [4]. However, the majority of HADDOCK users do not use the local version but prefer to use its user-friendly webserver. The server has gathered over 11.000 registrations to date and handles about 25.000 submissions per year, making it the main entrance point for HADDOCK users. In order to support the new cryo-EM protocol described above, the webserver needed to be updated.

The webserver of HADDOCK is built on top of a legacy framework that has successfully and reliably handled HADDOCK submissions for almost 10 years. This framework mainly relies on a homemade and high-level python-based wrapper around other python scripts. Although very reliable since its creation, the modularity and flexibility of the framework is rather limited. The first limitation comes from its intrinsic complexity, each addition to the framework triggers changes to be made at several locations. The second limitation comes from the framework design itself that has been built for serving static templates and render them as html. Interactivity at the user interface (UI) level and communication between the web client and the webserver is not possible or would require significant efforts to be setup. Those different limitations, and the will to deploy several new features that have been postponed for some time convinced us that the complete redesign of our webserver would be a very profitable long-term investment.

The shift towards a new framework for deploying the HADDOCK web portal has been even more relevant after the successful deployment of two new web portals within the West-Life umbrella, namely DisVis and PowerFit. Both web services rely on the Flask framework and are now up and running for a bit less than 2 years. The new HADDOCK server is named

HADDOCK2.4 webserver, to stay in line with the local HADDOCK software version it operates (haddock2.4, which will become the official HADDOCK software release once the new server will be in production).

3.2 New HADDOCK webserver

3.2.1 Motivation

One of the main feature of the new HADDOCK 2.4 webserver is the splitting of a previously unique form submission step into 3 different steps. This splitting was motivated by the desire to enhance the user experience by providing more feedback along the submission process, trying to reduce as most as possible potential errors that would occur during the execution of the HADDOCK workflow. This is done by carefully validating each data provided and warn the user if anything is wrong among the data he provided. Different levels of warning exist, from the simple flash message that pops-up after data processing, often related to a set of data that is rather unusual but went through the validation steps, to an error message, trying to report as accurately as possible the source of the error, preventing the user to go further in the submission process.

This is significantly different from the actual webserver where all data are submitted at once and their validation/processing is distributed over two steps.

3.2.2 Technical details

As mentioned previously, the core module behind the new web server is Flask. The Flask framework (http://flask.pocoo.org/) is a python-based framework broadly used in the scientific community. It offers the possibility to fast prototyping a new webserver with simple layouts without the need, at a simple level, for extensive knowledge of web development's standards. Many tools and pipelines associated to HADDOCK, and HADDOCK itself, are relying on the python language, making it a perfect language for a webserver framework. The implementation of existing pipeline for the validation and processing of data is then straightforward. On top of its default package, Flask offers many extensions that add features commonly found on other websites. Among them, we can cite, in no precise order:

- User data management (Flask-Login / Flask-session)

- User/admin mailing system (Flask-Mail)
- Authentication through SQL database usage (Flask-SQLAlchemy / Flask-Migrate)
- Forms creation and validation (Flask-WTF)
- html/css/js framework (Flask-Bootstrap)
- Admin management (Flask-script / Flask-Admin)

The entire workflow has been tested with python 3.6. In terms of dependencies, the web server relies on standard python libraries available through the Python package manager pip (exhaustive list available in Appendix 1). Only one third-party tool, Reduce (from Molprobity – version 3.24.130724), needs to be installed.

The entire new workflow is started as a set of docker containers (https://www.docker.com/) handled by docker-compose (https://docs.docker.com/compose/). Three different containers are fired to have the workflow up and running, as illustrated in Figure 1.

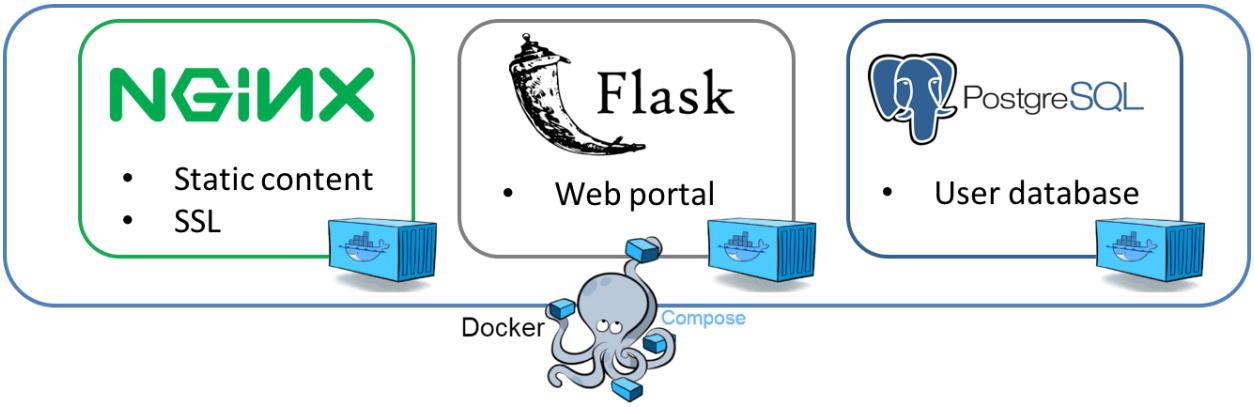

Figure 1 - Illustration of the docker-compose configuration made of 3 independent but interconnected containers.

First container configures and starts nginx (https://www.nginx.com/), an HTTP server and reverse-proxy, that serves static content and handles the SSL connection between the web browsers and the web server. Our PostgreSQL (https://www.postgresql.org/) database is running in a third container. Flask web portal is lying in the middle and relies on both containers to run properly. This management through docker-compose allows for a self-contained web portal setup that can be triggered on any Virtual Environment where docker is installed.

3.2.3 New user database

One of the main feature of the new HADDOCK2.4 web server is its usage of a user database to handle user data and jobs. As of today, HADDOCK user management was done through a secured text file grouping all users' information. Consequently, any action involving the need to identify a user was requiring the parsing of this file. Statistics, data management were therefore quite limited. The evolution towards a user database allows us to extend the information and experience of users by automating many steps and securing most of the location displaying user-related data (jobs, personal details, services subscription, etc.), which is particularly relevant in the context of the General Data Protection Regulations. This also reduces the risk of errors when manually editing, adding or deleting data. Most of our pages are now designed with respect to the connected users, who even get different content depending whether they are logged in or not. In the same way, any user-related content requires login and a permission check is done to match the data owner to the logged in user.

The database allows us to easily assign special permissions to users that impact the quantity of information accessible to them. This mainly concerns the docking parameters as detailed in section 3.2.4.3. We simplified the former *easy/expert/guru* levels to only two levels: *easy* and *expert*. The latter being the most permissive, allowing access to all parameters of HADDOCK. An *admin* level is also present, allowing a limited number of developers and/or operators of the HADDOCK portal to access protected pages and admin interfaces. Those admin interfaces allow modification of the databases including the "User", grouping all user details, and "Job", grouping all jobs submitted to HADDOCK, ones.

Equivalent to permissions, groups have also been introduced in the DB. They do not prevent or allow access to specific "advanced" features but can restrict or lift the sampling range limitations

of a docking run or allows for some priority in the queuing system for instance. As of today, 4 groups have been created: *default/course/team/special*:

- *Default* is the group assigned to all external users
- *Course* is used for training purposes, resulting in an automated reduction of sampling parameters to allow get results in a limited time for demonstration purposes
- *Team* is reserved to the HADDOCK developers and member of the Utrecht group
- *Special* is used for specific, high-priority usage, for example to run CAPRI (the blind docking experiment) targets.

3.2.4 Implementation

As we emphasized in section 3.2.1, the need to improve the user experience by providing instantaneous feedback on input data or parameters that could prevent HADDOCK to run and could lead to misleading results was our top priority (next to implementing the cryo-EM support) for this new version. To do so, we divided the submission pipeline of HADDOCK into 3 different steps:

1. Input data submission
2. Input parameters submission
3. Docking parameters submission

3.2.4.1 Input data

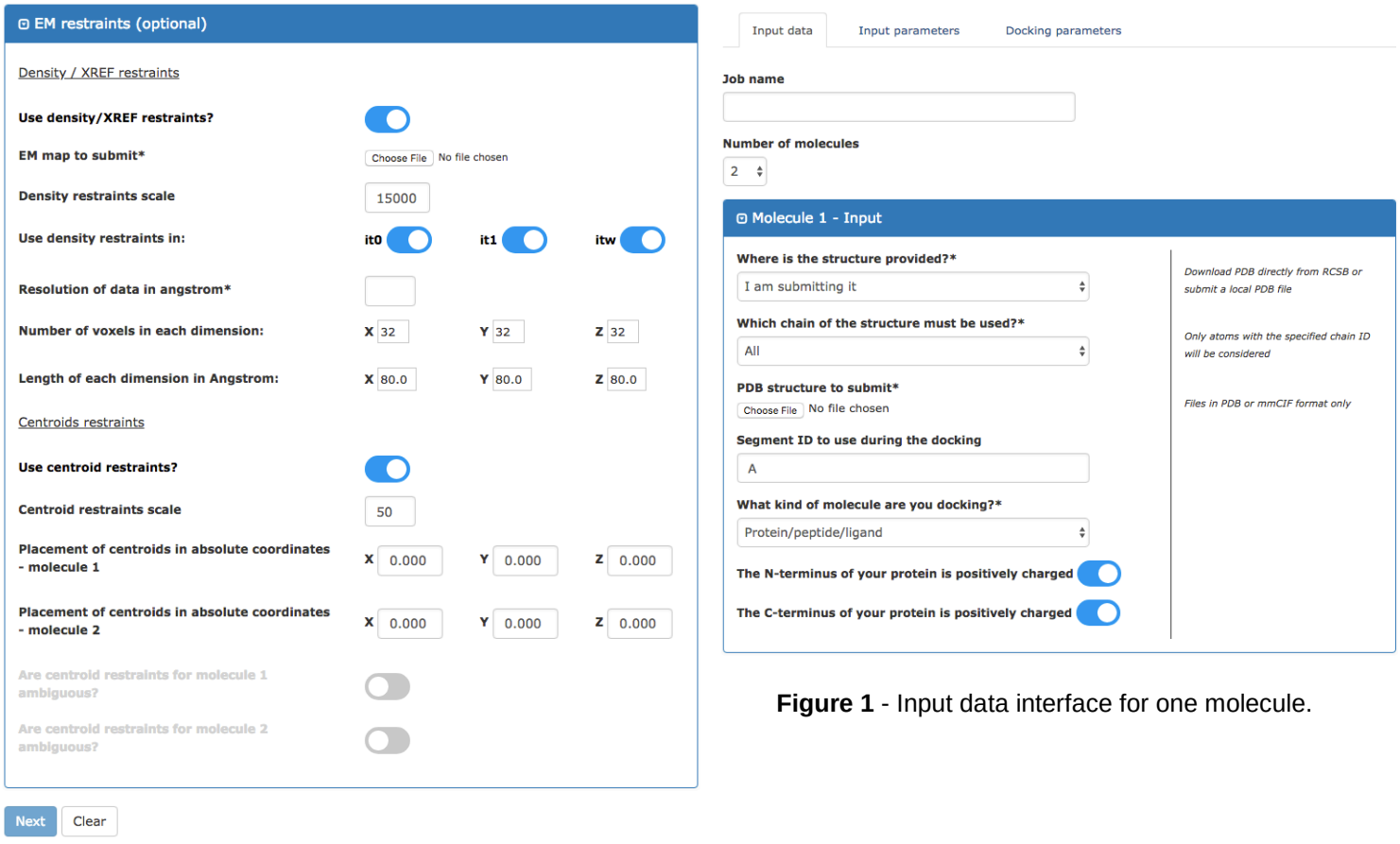

Figure 1 - Input data interface for one molecule.

At this step (Figure 2), the user submits the 3D coordinates of the molecular components she/he wants to dock. This can be done either by providing a local file stored on the user working station or by downloading it from the RCSB/PDB repository. Both PDB and mmCIF formats are supported. Some basic information about the system type (protein/DNA/…), the chain ID to be considered or the charge on the Cter/Nter of the molecule can also be provided. Some fields are mandatory (highlighted in the interface). Moreover, any mandatory field where no data has been input prevents the user to go further (deactivation of the submit button). Aside the 3D coordinates of the molecules, a new section dedicated to EM restraints has been added. This section groups every parameter required to run the EM protocol of HADDOCK. Mandatory fields are the EM map and its resolution as well as the centroids absolute coordinates. These are only mandatory if the density/XREF restraints button is checked. If not, a regular HADDOCK protocol will be performed with the molecular 3D data provided. We limit the size of the map to 200MB. We consider that this size is enough to contain the necessary data HADDOCK needs. It is worth noting that this size cap is reduced for PDB files whose size cannot exceed 10MB.

Upon submission (the “Next” button), the input data provided at this stage undergo several validation and processing steps. Those are listed in Figure 3 where the generic pipeline is also represented. Each of the step is wrapped up within the Flask framework and done sequentially upon submission. The user is informed of the process progression through a

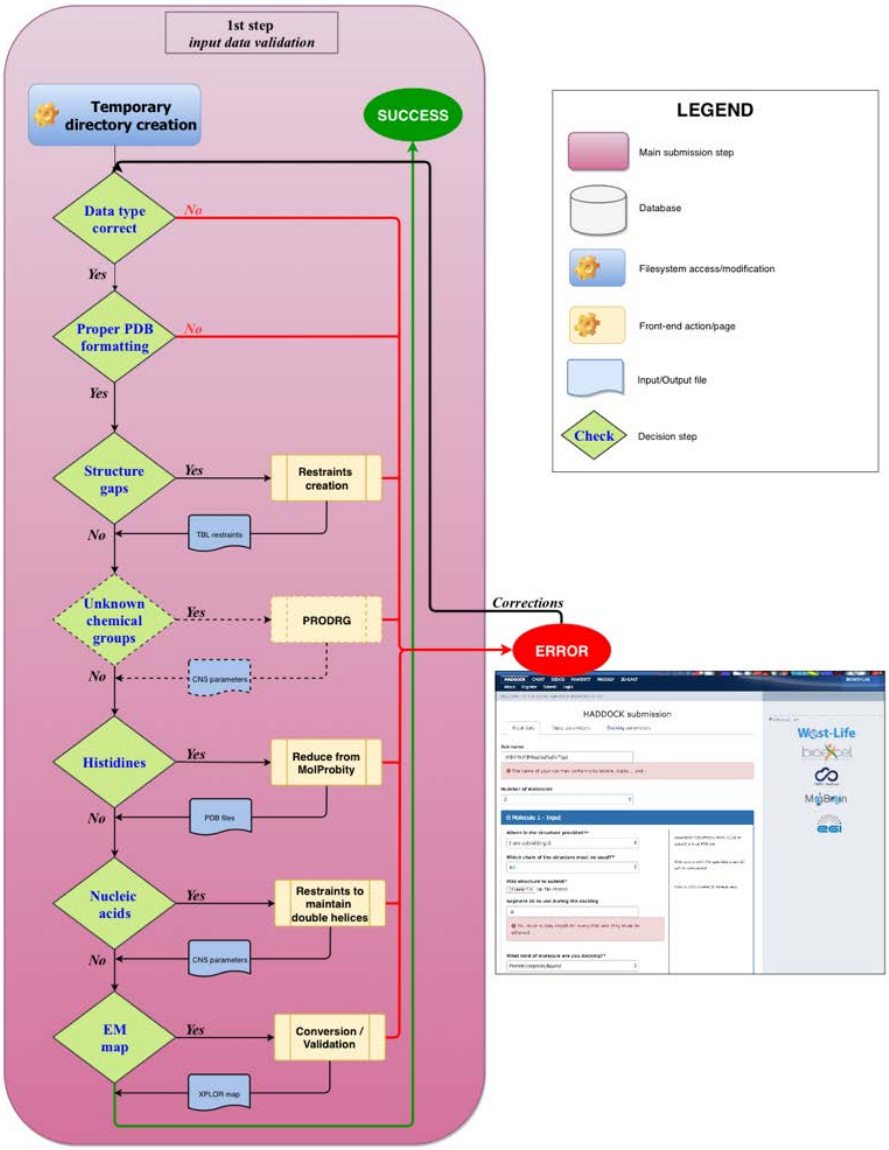

Figure 3 - Input data step pipeline upon submission.

progression bar that appears as soon as he/she presses the submission button. A large part of the data type validation is made internally as part of the Flask logic. Those steps verify the type of the data provided and, depending on the data requirements, validate it or not. Any wrongly formatted data will trigger a stop of the process and will output a pre-formatted error message visually associated to the erroneous field (see bottom right of Figure 3). If all data submitted are properly formatted then HADDOCK-related processing steps are started. First the atomic structures are checked for formatting issues using our own format checking script. Syntactic, coherency and integrity checks are made at this stage to be sure that the PDB will be read successfully by CNS and that the parameters associated (chain ID, molecule type) are consistent with the PDB data. Once data type and PDB format have been validated (if mmCIF, conversion is made to PDB for internal usage, same validation is performed on the new PDB file), several processing steps are performed on the input data. First, some PDB information are extracted: sequence, length, accessibility, etc. Those pieces of information will be either used for the following processing steps or at the UI level to enhance the user experience. Then the structure itself will be checked for potential gaps. If gaps are detected, a set of unambiguous restraints will be created to maintain the structure during the docking. The next step is the detection of ligands and their processing using PRODRG [5] to obtain topology and parameters for those molecules for the docking. This step is currently under development (it is present in the old server but needs to be adapted to the new framework). The next step in the pipeline is the detection of Histidines and assignment of their protonation states. This is done using *Reduce* [6], from the *Molprobity* package. *Reduce* is called from the Flask environment as a synchronous process. If successful, a dictionary of histidine positions and their associated protonation states is made available. If a nucleic acid partially or totally formed as an helix has been provided as one of the docking partner we automatically create a set of CNS-formatted restraints to maintain the helical structure and base-pairing during the docking. This step, which used to depend on an external software called 3DNA (http://x3dna.org/), has now be ported to python, which removed the dependency towards 3DNA. Finally, the EM map is processed and converted, if needed, into a CNS format. The conversion is made at the python level and does not rely on any third-party software.

If all previous steps have been successful, the data are stored in the temporary directory created at the beginning of the pipeline and the 2nd submission step tab is opened.

3.2.4.2 Input parameters

The input parameters stage mainly gathers parameters that either further define the input data or rely on them to be defined. One of the first and main improvement of the UI between the former and new version of the webserver is the use of processed data to guide user during the choice of her/his parameters. Thanks to that, any step involving the selection of residues can be complemented by an interactive sequence viewer that, on top of providing the sequence of the protein, is also displaying some basic secondary structure information. One illustration can be seen in Figure 4. Among the steps that rely on this viewer, the active and passive residue selection and the flexible or fully flexible segments definition. It is of course possible to directly input the list of residue IDs in the text input field associated to the viewer. Any residue ID that is not present in the PDB file will be indicated by a red message that will disappear as soon as all residue IDs are correct. If the user chooses to check the option to automatically define the passive residues from the list of active ones, an option to choose the radius of the search sphere will be presented to the user.

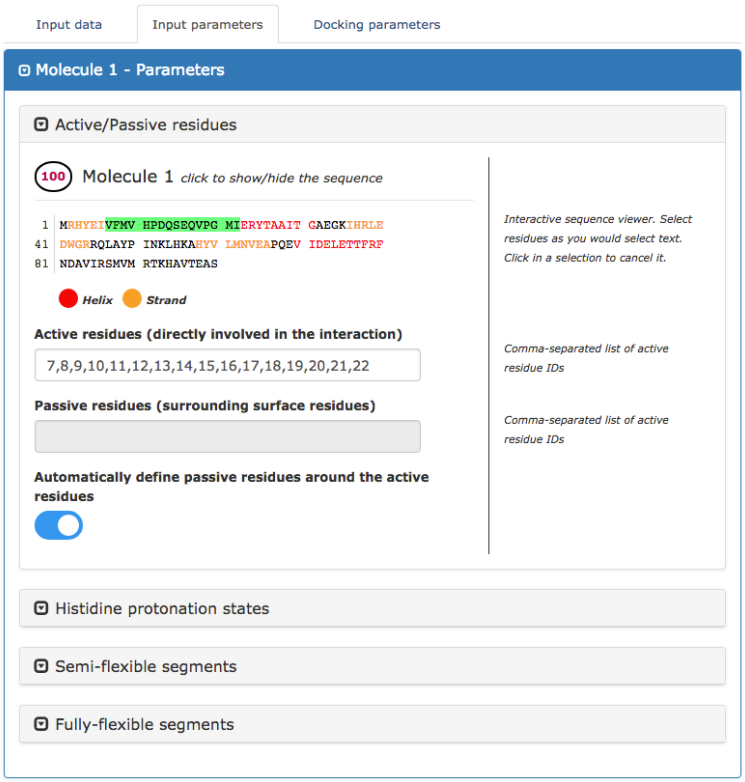

Figure 4 - Input parameters interface with interactive residue IDs selection.

Histidine protonation states, as calculated by *Reduce*, are displayed for each Histidine and the protonation state can be manually changed. If *Reduce* could not assign the protonation state of a specific Histidine, a warning message will be displayed and the user will have the possibility to define it manually.

Only basic validation is made upon submission of the input parameters. Since most of the options have been restricted to the input data, only few mistakes can be done at this stage. It is important to note that users with "*easy*" access will have to provide at least one set of active residues for one molecule and a set of active or passive residues for the second molecule. Some logic validation on the active/passive combination are performed upon this second stage submission to be sure that the docking run will go through. Users with "*expert*" access can go through without providing active and/or passive but will have to provide some restraints input files at the last step.

When all needed parameters have been added, the user can proceed to the step and, upon validation of the parameters, be redirected to the 3rd and last step of the submission.

3.2.4.3 Docking parameters

Docking parameters are all parameters that define the specificities of the docking protocol that will be performed with the input data provided before. They do not directly rely on the input data at this stage of development. Next steps of the development process will involve a dynamic change of those parameters depending on the system studied extracted from input at the previous stages. We published several specific protocols [7,8,9] that deals with different types of molecular complexes: protein-peptide, protein-DNA, protein-ligand, etc. They all use customized parameters to better deal with the particularity of the system. In the future, these will be automatically set depending on the system to be docking.

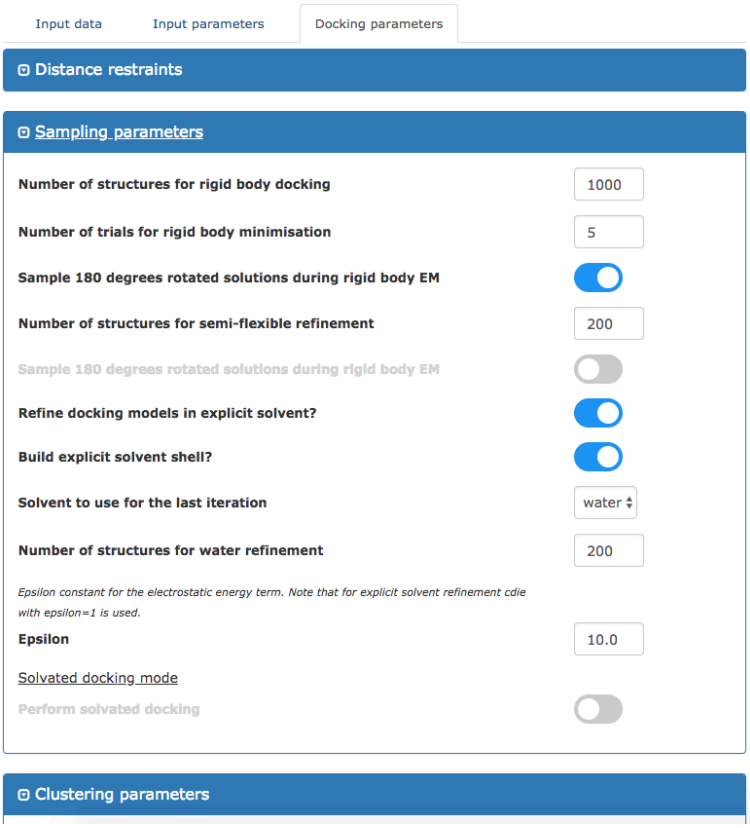

Figure 5 - Docking parameters submission interface**.**

About 300 parameters can be tuned at this stage of the submission. Some of the restraints parameters can be defined multiple times to

repeat their action over different part of the system. For those parameters, a dynamic interface only displays them when users decide to define some. This significantly lightens the interface and allows to define, a priori, a maximum number of restraints that can be manually input. Parameters are split between different categories to ease the navigation.

3.2.4.4 Docking run submission

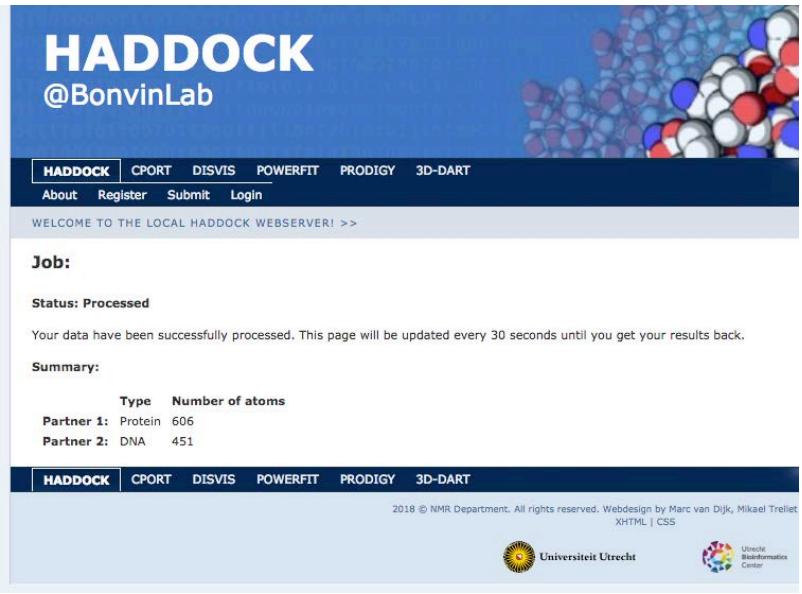

Figure 6 - Status page example with job information summary.

Upon submission, several actions are performed. First, as for the previous stages, the data types are validated. If this step goes through, a logical validation is made to make sure that the combination of parameters is coherent. A significant part of this step addresses the restraints definition to avoid redundancy and/or incompatibility. Several messages can also be sent at this stage, warning the user about unusual combination parameters that won't prevent the submission but might lead to misleading results. If the previous two steps are successful, the input files generated along the submission process are copied to a temporary directory where they will be further processed by HADDOCK software framework. A results directory is also created and some information about the job are written to a database. Among those information, the job id, user id, job submission time, job status ("Processed" at this stage), number of partners, type of the partners and number of residues of each partner. Finally, the user is redirected to a status page where a message indicated him the success of his submission and where a basic summary of the parameters is provided (Figure 6). This page is refreshed every minute indicating the updated status of the job.

3.3 Future developments

With the new version of the server supporting cryo-EM data now available online in a devel version, we will proceed with the implementation of the missing features from the old server (e.g. the support for small molecules). We are also planning to invite selected users from our large user community to test the new interface and give us feedback. This will allow us to further improve the user experience prior to putting the server in production mode. Since the server is building on a new user registration and management system, which will cover all services running at the Utrecht partner site, this is also the logical place where to implement a single-sign-on (SSO). system that will connect to the WestLife or EGI-checking SSO.

Appendix 1: Web server requirements

```
alembic==0.9.8
biopython==1.70
blinker==1.4
certifi==2018.1.18
click==6.7
dominate==2.3.1
eventlet==0.22.1
freesasa
Flask==0.12.2
Flask-Admin==1.5.1
Flask-Bootstrap==3.3.7.1
Flask-Login==0.4.1
Flask-Mail==0.9.1
Flask-Migrate==2.1.1
Flask-Script==2.0.6
Flask-Session==0.3.1
Flask-SocketIO==2.9.6
Flask-SQLAlchemy==2.3.2
Flask-WTF==0.14.2
greenlet==0.4.13
gunicorn==19.7.1
itsdangerous==0.24
Jinja2==2.10
Mako==1.0.7
MarkupSafe==1.0
numpy==1.14.1
psycopg2==2.7.4
pycountry==18.2.23
python-dateutil==2.6.1
python-editor==1.0.3
python-engineio==2.0.3
python-socketio==1.9.0
six==1.11.0
SQLAlchemy==1.2.5
style==1.1.0
update==0.0.1
visitor==0.1.3
Werkzeug==0.14.1
WTForms==2.1
uwsgi
```

References cited

[1] Dominguez, Cyril, Rolf Boelens, and Alexandre MJJ Bonvin. "HADDOCK: a protein− protein docking approach based on biochemical or biophysical information." *Journal of the American Chemical Society* 125.7 (2003): 1731-1737.

[2] Van Zundert, G. C. P., et al. "Fast and sensitive rigid-body fitting into cryo-EM density maps with PowerFit." *AIMS Biophysics* 2.2 (2015): 73-87.

[3] van Zundert, Gydo CP, Adrien SJ Melquiond, and Alexandre MJJ Bonvin. "Integrative modeling of biomolecular complexes: HADDOCKing with cryo-electron microscopy data." *Structure*23.5 (2015): 949-960.

[4] Van Zundert, G. C. P., et al. "The HADDOCK2. 2 web server: user-friendly integrative modeling of biomolecular complexes." *Journal of molecular biology* 428.4 (2016): 720-725.

[5] SchuÈttelkopf, Alexander W., and Daan MF Van Aalten. "PRODRG: a tool for high-throughput crystallography of protein–ligand complexes." *Acta Crystallographica Section D: Biological Crystallography* 60.8 (2004): 1355-1363.

[6] Chen, Vincent B., et al. "MolProbity: all-atom structure validation for macromolecular crystallography." *Acta Crystallographica Section D: Biological Crystallography* 66.1 (2010): 12-21.

[7] Trellet, Mikael, Adrien SJ Melquiond, and Alexandre MJJ Bonvin. "A unified conformational selection and induced fit approach to protein-peptide docking." *PloS one* 8.3 (2013): e58769.

[8] van Dijk, Marc, et al. "Solvated protein–DNA docking using HADDOCK." *Journal of biomolecular NMR* 56.1 (2013): 51-63.

[9] Kurkcuoglu, Zeynep, et al. "Performance of HADDOCK and a simple contact-based protein–ligand binding affinity predictor in the D3R Grand Challenge 2." Journal of Computer-Aided Molecular Design 32.1 (2018): 175-185.