BioExcel Whitepaper on Scientific Software Development

Mark J. Abraham, Rossen Apostolov, Paul Bauer, Alexandre M.J.J. Bonvin, João M. Correia Teixeira, Bert L. de Groot, Vytautas Gapsys, Gerrit Groenhof, Berk Hess, Erwin Laure, Erik Lindahl, Adrien S.J. Melquiond, Dmitry Morozov, João P.G.L.M. Rodrigues, Mikael Trellet[1], Rodrigo Vargas Honorato

Abstract

The BioExcel consortium develops and maintains several software packages, including GROMACS, HADDOCK, PMX and interfaces for QM/MM using CPMD or CP2K. These codes address different problems, are written in different languages by different sub-teams, delivered to users in different ways, and all have unique challenges in identifying good processes and seeking ways to improve them. They are real-world examples of major complex scientific software packages that have adopted more or less advanced formal software development processes.

This white paper presents the experience of the developers and recommendations for development of high-quality software engineering processes. This is an update from our previous white paper[2] to reflect process improvements developed during BioExcel-2.

Introduction

All software projects, in particular those supporting science, share concerns such as:

1. Is the software's output scientifically correct?
2. How will the software change over time?

[1] Now at Fluigent Smart Microfluidics, Paris, France

[2] Mark J. Abraham, Adrien S.J. Melquiond, Emiliano Ippoliti, Vytautas Gapsys, Berk Hess, Mikael Trellet, João P.G.L.M. Rodrigues, Erwin Laure, Rossen Apostolov, Bert L. de Groot, Alexandre M.J.J. Bonvin, Erik Lindahl (2018). **BioExcel Whitepaper on Scientific Software Development.** *Zenodo*. https://doi.org/10.5281/zenodo.1194634

3. Does the code need to be faster or more scalable?
4. How can users learn how to use the software?
5. How to convince people to invest in the software?
6. What happens when the original author moves to a new project?

The BioExcel Center of Excellence (https://bioexcel.eu/) have been tackling these challenges by forming a scientific software development process that is detailed in this whitepaper.

Researchers need software applications and automated workflows for biomolecular simulations that can be shown to work correctly when run on exascale resources. This requires high quality software, but even the most outstanding projects will quickly degrade in quality as they grow more complex – unless you also have a high-quality *software-development processes*.

The BioExcel consortium develops and maintains several software packages, including GROMACS, HADDOCK, PMX, and QM/MM interfaces with CPMD and CP2K. These codes address different problems, are written in different languages by different sub-teams, delivered to users in different ways, and all have unique challenges in identifying good processes and seeking to improve them. They are neither perfect nor necessarily the best examples of software development, but they are real-world examples of major complex scientific software packages that have adopted more or less advanced formal software development processes.

Your software project will benefit from reflecting on your process and developing a plan to improve it. Some areas will be easier, or more urgent, to address than others. Accordingly, in this white paper, we record the current state of the software-development processes within BioExcel, along with plans. These "worked examples" will give you real insight into state-of-the-art scientific software development activities and guide you to make good choices for your team. We hope that these ideas will help you develop correct software faster and cheaper, be portable, attract users, and demonstrate to funders that your software is worth supporting.

Your plan will need to be customized. Your software must reflect your needs, the needs of your other users, and the available resources. But some aspects are common to almost all software projects, so we will discuss these first and suggest external resources for grappling with them. Then we will briefly describe the four software packages whose processes are described herein, so you can then focus on the one that best fits your needs. Finally, the details of the process for each package will be given.

Common elements of software development processes

Most scientific software projects should have a plan for all the following aspects. There is often useful "low-hanging fruit" to consider for improvements.

- Agree on a **license** model for your software, remembering that programmers, their employers, and their funding agencies may have expectations that need to be negotiated. Mention the license in your documentation and provide it in full alongside the software. Licenses are tools, so pick a license that will help the project achieve its goals.

- Make frequent formal **releases**, so that your users can benefit from your new work, and you have an accomplishment to report. Don't wait until it's "ready" – software never can be until real users have used it for real work.

- Use **version control** during development, so that experiments can be rolled back, bugs more easily found and tracked, and your science can be reproduced.

- Document the **dependencies** – what versions of tools, compilers, libraries and operating systems are required? What is supported? What is tested? Which other version, libraries and/or tool might work but aren't tested? How do you decide on what new dependencies can be introduced?

- Plan code changes before rushing in – **gather requirements** from your users first, so that you build a thing that is useful while avoiding designs that block future changes. You won't be able to do everything that they want, but you should listen for ideas and opportunities and prioritize them.

- Use unit tests, and **design for testability**. Before starting implementations, separate features into small methods that can be tested exhaustively to guarantee correctness. Write the tests first – the implementation is finished when it passes the unit tests.

- Design **performance tests** relevant to current real-world usage, and use these to guide profiling experiments, plan optimizations, check for regressions, and support announcement of performance improvements. Keep a balance between the needs of researchers now, and the direction in which HPC platforms are evolving.

- Document the **stability of features** so that users can know what is experimental, and what is reliable.

- While features are assets, **all code is a liability**. No matter how amazing new code is, it will have to be maintained, tested, updated, and documented – which takes time. Is the new feature absolutely worth this time – and who will maintain it next year?

- **Remove less used features** to reduce the complexity of code and simplify maintenance. All members of a team need to share the responsibility for cleaning and maintenance, or the ones doing it will eventually leave the project.

- **Announce deprecation** in advance, before removing support for a feature, so that users are aware that their needs are considered, and perhaps others will contribute to the project to keep something that they need.

- Publish a place to **record issues with the software** – this can be as simple as a shared Google document or wiki page. Users might want to help understand if there is an issue and how it might be resolved.

- Use **formal code review** so all changes are checked and approved by somebody else in the team. This might initially slow down your commit rates, but it will reduce the number of bugs drastically and increase your overall development pace.
- Consider **adopting automated review & integration tools** even for small projects. There are many cloud services that provide this at low cost.
- Adopt a common **coding style**, so that your developers can read and maintain different parts of the code easily, and your code reviews can focus on the substance of the change, and not where tabs and punctuation should go.
- Consider adopting formal user-experience (UX) design processes, e.g. as documented in the User Experience for Life Sciences (UXLS) toolkit,[3] so that your product will best fit the actual needs of your users.

There are several excellent publications and knowledge bases for scientific software development. The UK-based Software Sustainability Institute (https://www.software.ac.uk/) has one of the best and most up-to-date collections of guides you will find useful (see https://www.software.ac.uk/resources/guides).

Software development process and readiness level

There is no unique "correct" level of software, but it is important to be aware of the extremely broad range and differences between early scientific concepts, test implementations, software that can be used internally with tight feedback loops with the developers, limited external usage, and widespread deployment in external production environments with adequate support, training and quality assurance in place.

Both the US Department of Defense, NASA, and the European Commission have formulated "Technology Readiness Levels" (TRL)[4]. The scale ranges from TRL1 ("basic principles observed and reported") to TRL9 ("actually proven through successful mission operations") (Figure 1.). Although not specifically developed for software or scientific applications, it is a good idea to consider what level your software project aims to achieve. The higher levels are difficult to reach without extensive investments

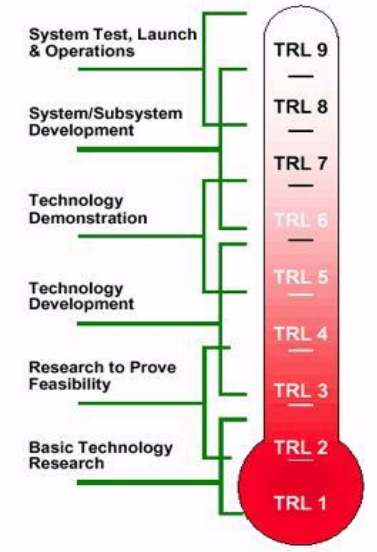

Figure 1. Technology Readiness Levels (TRLs)

[3] UX Toolkit for Life Sciences https://www.uxls.org/

[4] Technology readiness levels (TRL), HORIZON 2020 – WORK PROGRAMME 2014-2015 General Annexes, Extract from Part 19 - Commission Decision C(2014)4995.

in documented development processes. Simple processes suit projects whose scope is small, or lifetime short.

BioExcel aims at constantly improving the guidelines for the development process to help achieve higher levels of readiness. Commercial and academic users should be able to understand the readiness of any technology proposed for use in a project, as part of their risk assessment. For example, deploying simulation software on a problem type different from the test suite accepts a higher risk of failure.

Brief description of the BioExcel software packages

These brief descriptions will help you identify which of the detailed descriptions is likely to have useful process ideas that you might adopt, because they come from a software team working on similar software, languages, libraries, or delivery process.

- **GROMACS** (http://www.gromacs.org) combines a high-performance classical molecular dynamics simulation engine with powerful tools for preparation and analysis. It is released under a liberal business-friendly open-source license (LGPL 2.1) and comprises 500k lines of C++17 targeting all major HPC platforms. Development is led by a core team at KTH in Stockholm and enjoys active contributions from a wide range of other academic and industrial partners. A formal code-review process is accompanied by pre-submit continuous integration testing on several different OS, libraries, and compilers.

- **HADDOCK** (https://haddocking.org) implements integrative modelling of biomolecular complexes based on the CNS engine together with a set of extension scripts that incorporate information from a wide range of experimental or bioinformatics sources. The code consists mainly of a Python layer orchestrating all the computations performed with CNS. A large fraction of the docking protocol itself is written in the CNS scripting language. Users typically access HADDOCK via its grid-enabled web server, supported by several EGI partners in The Netherlands, Europe, and other suppliers worldwide. The team at Utrecht University handles development of both the core software and the web server.

- **CPMD QM/MM** (http://www.cpmd.org) effort within BioExcel is a new combination of CPMD (written in Fortran 90) with GROMACS, implemented at the Juelich Research Center. The new QM/MM interface and its communication library designed to manage the data exchange between the two codes, is being written in Fortran2008 and C++11, respectively. This interface will allow one to couple CPMD to GROMACS (and later other classical molecular dynamics codes) with a minimal core of changes so that both codes can run on different MPI processes on supercomputers.

http://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf

- **GROMACS/CP2K QM/MM interface** enables coupling a molecular mechanics description of a system, with a density functional theory description using two highly parallel codes, GROMACS and CP2K. This is done through the standardized MDModule interface in GROMACS, which was developed in BioExcel. Another part of the project is improving the scaling of the parts of CP2K that are most relevant for QM/MM.

- **PMX** (http://pmx.mpibpc.mpg.de) is a preprocessor for molecular dynamics software that builds efficient molecular topologies for alchemical free energy calculations to predict stabilities and affinities. It is tightly bound to GROMACS development, facilitating massively parallel free energy calculations. PMX can be run both standalone as well as from a webserver. The core and infrastructure are maintained by the Max Planck Institute for Biophysical Chemistry in Göttingen.

Development process details for GROMACS

GROMACS[5] is a molecular dynamics simulation engine that can simulate the Newtonian equations of motion for systems with hundreds to millions of particles. It is primarily designed for simulations of biochemical molecules like proteins, lipids and nucleic acids. Its chief virtue is that it is extremely fast at calculating the expensive non-bonded interactions necessary for such systems. It is a state-of-the-art best-in-class implementation of molecular dynamics, with high-performance implementations for a wide range of commodity CPUs and GPUs, including all current and several anticipated future HPC platforms. It is in use by thousands of scientists, leading to citations of associated scientific articles currently numbering more than 2000 per year.

Figure 2 depicts the current state of GROMACS code development process. Having made appropriate plans before writing code, developers upload proposed changes for review to a new branch on GitLab for code review and automated continuous-integration testing using Docker images on an in house Kubernetes cluster with CPU and GPU nodes, and awaiting review from other developers. Automatic cross-references to the issue tracker on GitLab to verify fixes for bug reports and feature requests are made. Once code is accepted into the master branch of the repository, it is automatically pushed to our Github backup repository.

[5] Abraham, M.J., et al., *GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers.* SoftwareX, 2015. **1–2**: p. 19-25. https://doi.org/10.1016/j.softx.2015.06.001

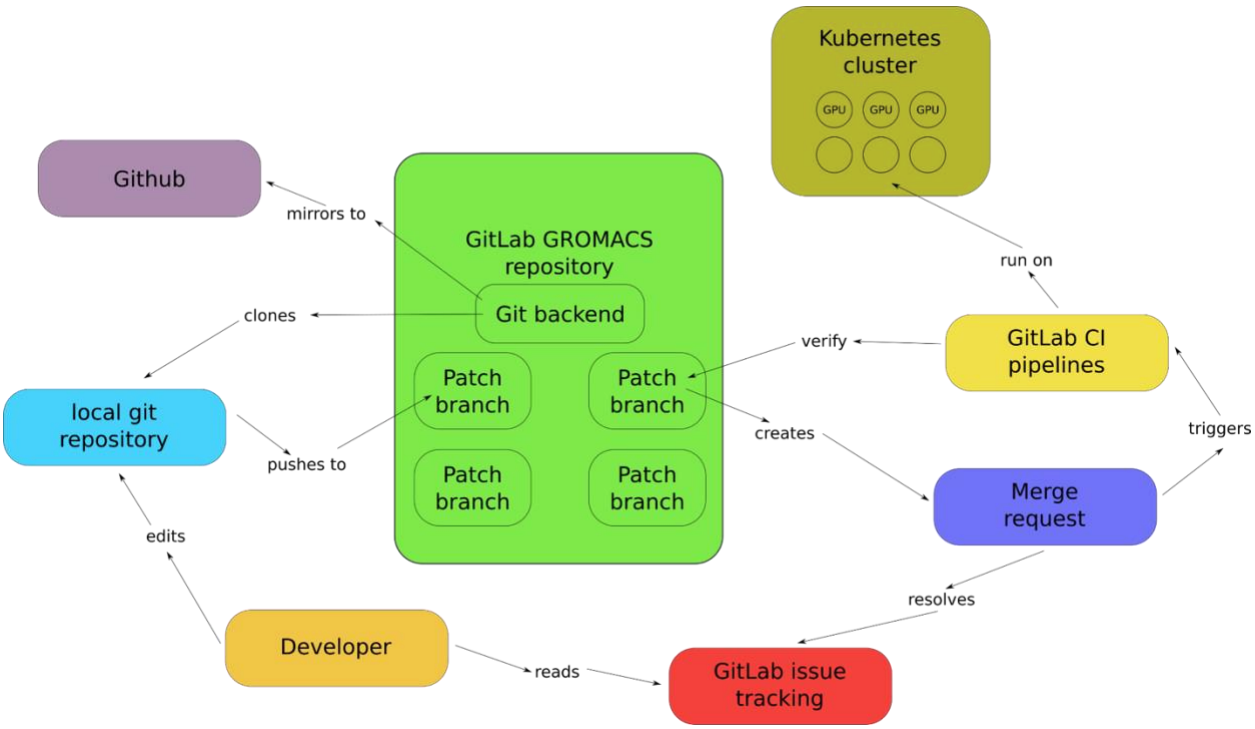

Figure 2. Current GROMACS development workflow description

Copyright and License

GROMACS is released under the Lesser GNU General Public License (LGPL) v2.1 or later. This is an intentional choice to balance our requirements for all extensions to the main codebase to be freely available (which is important for our academic impact), while still enabling commercial applications to link with and use GROMACS as a library. Each source code file has a copyright statement. Source tarballs contain the appropriate COPYING and AUTHORS files, that additionally notes the origin of code bundled along with GROMACS, and the license under which it is redistributed.

Architecture

GROMACS includes a high-performance simulation engine mdrun, several independent tools for preparing simulations, and a suite of several post-simulation analysis packages. Currently all are run on a command-line terminal from a common gmx wrapper binary by naming the required sub-tool. This provides a small surface area for potential conflicts with other software on a user's system, and many convenient features including the ability to deprecate or rename tools while still retaining a way for users to learn whether and where the old functionality is still available. Having a single entry point to the library will also help our future transition to provide runtime dispatch with several different dynamic libraries optimized for different types of hardware. Much of the approximately 500 thousand lines of code is specific to mdrun, and much is shared across all the tools, and the long-term future of the package will require extensive reorganization to

reflect which components support which functionality and thus require appropriate focus when developing and maintaining the software. However, no substantial changes to the user interface are planned, although simplifying the performance-oriented interface to mdrun would offer substantial improvements to user results and experience.

Feature request handling

Current: Anyone, including regular developers, may search and open issues at https://gitlab.com/gromacs/gromacs/-/issues to seek interest, find common ground and identify strategies. A set of structured templates for new issues (e.g. bug, feature request, enhancement) helps users provide sufficient details. Usually proceeding to actual development will require someone actively finding a source of funding and a suitably experienced developer, and enough time and enthusiasm in the rest of the developer community for testing, code review – and a long-term plan for how the feature will be maintained.

Future: In future phases of BioExcel, provide opportunity for clients to discuss e.g. development of new features on a contract basis. Given the complexity and size of the code, we will also start requiring that all new major features are initiated as a discussion using GitLab epics, so it is possible to request significant changes to the architecture before development starts.

Bug tracking

Bugs are tracked centrally using GitLab issues https://gitlab.com/gromacs/gromacs/-/issues also allows any user to register an account and file a bug report, along with uploaded input files. Developers regularly browse for relevant issues and engage in productive discussions with the reporter(s) and each other. When fixes are uploaded for code review, they can cross-reference the bug report, and automatic HTML cross links are created. Release notes can then note that the bug was fixed and provide links to more detail as required. The central integration into GitLab allows automatic linking of bugs, issues, and fixing Merge Requests (MRs) in a clear user interface.

User support

Formerly: The main form of user support used to be the popular and widely used GROMACS user mailing list.

Current: This has been replaced by a Discourse forum on the BioExcel website bioexcel.eu. This unifies the user support setup with the other BioExcel codes. The forum has several advantages over a mailing list. It has a richer interface and good search functionality. When a user types a question, she/he gets presented with a list of similar questions, when present, which might already contain the answer they are looking for. After a short period of similar questions to those posted on the old mailing list, we have observed a clear decrease in the number of commonly asked questions, indicating that users are finding the answers on the forum before posting. This frees up significant time for the support staff, who can now focus their efforts on the more advanced, as well as new questions posted. Features requests and bug reports reported on the forum are redirected to gitlab issues, either by the user or by developers, with links in both

directions, so both users and developers can easily track the progress and request more information if needed.

Version policy

Formerly: GROMACS used a major.minor.patch numbering system (e.g. 4.6.3). Patch releases could only fix things that didn't work. Ad hoc decisions led to a new version being described with a major or minor version increase.

Current: GROMACS numbers each release as year.patch (e.g. 2021.1) so that it is clear to users how old their version is, and that the release was made because it was time to do the annual release. Since software development can generally produce only two out of three of high quality, on time, or with specified functionality, we focus on the former two so that users can benefit from the new functionality that has been completed, without risk of open-ended waiting for other functionality. This provides the wider development team with confidence that an accepted feature will get into the hands of users without having to wait for completion of some other feature intended to go into the next release. However, it also means it is the responsibility of each developer to make sure their feature is not only ready on time but has gone through code review and been accepted by the team before the release deadline.

Future: No changes are planned. Current resources do not permit more than one annual major release, combined with patch releases every few months.

Version control

Git is used for version control. All code development should start from the official git repository hosted on GitLab (https://gitlab.com/gromacs/gromacs/) or one of its mirrors (including https://github.com/gromacs). Developers modifying GROMACS will benefit from being able to use git tools such as grep and rebase to work with the code. Code development should not start from a release tarball, but if this has happened then it can be copied onto a git repository with that version checked out.

Branching

A stable master branch will exist and is generally open for both refactoring and functionality changes, including adding and removing features. Stability will be assured through code review and continuous integration testing. No features can be committed until they pass code review, provide full documentation both of source code and user-facing features, and pass all integration tests – the master branch is not allowed to decay between releases. When it is time for a new annual release, a named branch is made to reflect this. No changes to the scope of functionality are expected after this time, while any issues of module integration are resolved before the actual release. After the release, fixing may continue on the release branch according to policy (no new features are allowed in any release branch, and only critical fixes in old release branches), and such fixes will be merged into any more recent release branches, and then into the master branch.

Modularization

Current: GROMACS is making a transition from C89 to a modular, unit-tested modern C++17 library. This transition is slow and costly, but expected to deliver long-term benefits from lower maintenance, better extensibility and higher portability. The classes within these modules encapsulate behavior alongside the data necessary to implement it, and require full Doxygen[6] documentation of files, classes, members and methods. The modules follow a layered design that is enforced by checking scripts.

Future: Draft a target module hierarchy for the wider development community to use as a mental model during the refactoring process. All new modules will be required to have 100% unit test coverage before they can be committed. This will be verified by means of automated coder coverage testing.

Development process details

Current: Developers are generally autonomous academics who propose ideas on the gmx-developers mailing list, or as new issues on GitLab, and interested other developers contribute ideas. These evolve on an ad hoc basis. Developers progress to develop working code which is then uploaded for peer review to the repository on GitLab (https://gitlab.com/gromacs/gromacs/). Developers can propose code changes that are tagged "request for comment" or "draft" so that reviewers know that high-level feedback is needed now. An ongoing challenge is to motivate others in the developer community to contribute to design and review stages in a timely fashion, because most developers are academics with multiple responsibilities and GROMACS development is only a small fraction of their output (and not always clearly recognized for career progression). However, since all developers are subject to the same review and testing process, all do already recognize that nobody can progress in isolation.

Developers require active encouragement to remember to *separate changes that refactor the code from those that change the functionality*, because the time of code reviewers is a particularly scarce resource, and such separation maximizes the value delivered by reviewer time.

Future: BioExcel developers will lead the way in producing a written plan, in particular for how the user interface will evolve alongside features.

Review Process

Current: All code changes go through two-person review at https://gitlab.com/gromacs/gromacs/, with approval required from at least one core developer, and one other developer. Preferably at least one of those has previously contributed to code in this area, so they are able to make an informed review reasonably efficiently. Review is difficult

[6] http://www.doxygen.org

for both parties, so feedback and responses need to be considered carefully and should be technical and impersonal. Proposed changes do not have to be perfect to pass review, but they should represent a clear improvement in at least some aspect without undue compromise on other aspects. Complete Doxygen documentation is a strict requirement for all new code. Authors may declare a reviewer suggestion out of their intended scope, in which case the discussion should move to a Redmine issue for future consideration. Authors may negotiate that someone else takes over responsibility for the patch.

Future: The most important quality of scientific code is its correctness at implementing the method claimed. Only a subset of kinds of proposed changes can compromise this with GROMACS. It is inefficient to require the same level of developer scrutiny on all changes, particularly when the number of developer hours available to the project is strictly limited. In concert with improved testing, identify areas of the code for which greater risk is acceptable, e.g. refactoring of modules already under high quality unit tests, improvements to user or developer documentation. For major changes to older modules, it is required that it is combined with rewriting the code as proper C++17, with proper unit test coverage before changes can be done.

Integration approach

Current: No long-term development branches exist, so proposed code changes are normally based off a recent master-branch commit, and thus can be readily rebased for integration with the current HEAD commit of the master branch. It is the responsibility of each developer of a feature to track and integrate this feature as the stable master branch evolves in parallel with his/her development.

Future: Some larger development efforts may benefit from a long-running feature branch, which would then need to be integrated back into the master branch. A git merge is perfect from a technical point of view but does not meet the needs of a policy where code review must occur before acceptance into the master branch. Given the limited developer effort available for code review, code needs to be presented for review in small pieces that each passes its own unit tests, and any available integration or end-to-end test. Any move to implement a long-running feature branch must present and justify and process that would lead to effective review.

Testing requirements

Current: All proposed code changes are tested automatically using GitLab CI pipelines, with individual tests run inside Docker images with reproducible build environments on our in-house Kubernetes cluster. It builds the code and runs the tests on multiple compilers, standard libraries, accelerators and CPU architectures. Additional testing on more diverse systems is done after a change has passed these tests using Github Actions on the Github mirror. This caters directly to ensuring long-term portability of the code base to the anticipated features of the exascale landscape. Several static and dynamic analysis tools, such as Address-, Thread- and Undefined Behavior Sanitizer, are also run. All tests must pass before a change can be acceptable. The range

of tests and tools are reviewed and updated continuously[7]. Release versions, e.g. source tarballs, are tested on a range of configurations before being made available to users.

Future: Since the range of testing needs to expand, we wish to use more than one tier of testing, to limit the number of machines required for doing the builds. Fast-running tests will run for each proposed code change on important platforms, and longer running tests will also run on a wider range of platforms once the proposed code has been accepted.

Specific activities to be achieved over the course of BioExcel are:

- expanding test coverage (particularly including rerun, multi-simulation, and replica-exchange functionality),
- replacing old testing Perl driver script with GoogleTest C++ driver,
- deploying automated performance regression testing,
- continuing to add support for warning-free builds for up-to-date compilers, code analyzers, and libraries,
- report annual increases in test coverage.

Release Process

Current: Formal releases will only be made from release branches and will follow the versioning scheme in use at the time that branch forked from the master branch. Generally, at least one release candidate is made available for the community for around a month of informal testing before the final release. During this time users and developers are encouraged to attempt to validate that their simulations of interest work at least as well (or better) than previously and test new features. Debian and Fedora projects already test GROMACS on a wide range of platforms, and they will be invited to help with this effort. The eventual source and tests tarballs for the release are built by using a dedicated GitLab CI build pipeline from a commit in the repository that will later be tagged with the release number. That tarball is automatically tested during this process on a range of installation configurations, to verify that the build works and the tests pass. The documentation that matches the release is automatically generated from the tarball and prepared for easy deployment to web servers. The stages of the release process are shared with the wider community through emails to the users, developers and announcement mailing lists, posts on social media outlets (Facebook, Google+, Twitter), and both the GROMACS and BioExcel websites.

[7] https://manual.gromacs.org/nightly/dev-manual/infrastructure.html

Future: Finalize automation of final details of an automatic release build, including construction and deployment of the release notes, and embedding tarball checksums in the appropriate locations.

Deprecation

When a feature is identified as superseded, is no longer functioning, or is lacking developer support for maintenance then after due consultation with the user and developer community, it will be deprecated. This means it will be announced as deprecated in the next annual release, so that users who run the code have warning that they are at risk of depending on code that will not be maintained in future. Thereafter it may be removed in the master branch at the discretion of the developer community, which means it will not be part of the major release another year later. When code is known to have been broken for multiple years already, and thus cannot be in active use in a maintained branch, it might be removed without a deprecation notice in the software, but this will be acknowledged in the release notes of the next annual release. Support for older hardware platforms is also removed gradually, to make effective use of developer time porting to new hardware; older versions of the code still run on the older hardware for those who cannot upgrade usefully.

Retirement

Current: Each (annual) major release is supported until the next major release (i.e., typically one year) on a best-effort basis by developers to fix any shortcoming in the implementation of functionality intended to be supported in that release. That includes code correctness, accuracy and presence of documentation, functioning of build system, and simulation performance, whether reported by users or developers. The stability of related code paths will form part of the decision about whether, where and how to fix a bug. If a bug fix is not feasible, then we will alter the code to prevent future releases from running that wrong code, and report this to the user in subsequent bug-fix releases. After one year, there will be a further year of best effort to fix any severe issue that would produce scientifically incorrect results (whether in mdrun or tools).

The developers reserve the option to fix a bug only in the master branch if that seems the most practical course. Because large scale refactoring of the code base is underway, it is impractical to remember all the details of several implementations that were used in past years, current implementations, and the proposed future implementations, and bugs and developer conflicts have been introduced because of such lapses in understanding.

Future: It may become feasible to support release branches for longer periods. We have identified several improvements that will make this feasible; the transition to a modular C++17 library needs to be substantially complete, the test infrastructure must be contained in the source-code repository, an automated suite of performance tests must be available and automated, and a wide range of simulation quality tests must be available and automated. When a GROMACS library API is developed (which is a key outcome of a funded NIH project), then we will consider whether the point of long-term support becomes the version (or level) of the API,

rather than the release version, and again the existence of automated testing and the stability of the GROMACS source-code infrastructure will be key components in this decision.

Code commenting

Code is commented with a view to explaining what the code is doing, rather than how. If it is not obvious how the code is working, it should be split into smaller methods with explicit long names for variables and methods, use better control flow, and have examples with working tests. Care should be applied to document the interfaces of methods separately from internal implementation comments. New and newly refactored methods and source files must have Doxygen-style comments that are automatically built into the developer guide[8]. The Jenkins continuous integration environment enforces these requirements, and code that does not follow it cannot be committed.

Coding standards and styles

Current: The specification for code style[9] is part of the codebase, and proposed changes will undergo the same type of review as source code changes. All simple requirements (spacing, indentation, etc.) is checked automatically during continuous integration. The GROMACS style is loosely based upon the Google C++ Style Guide[10], with the notable exception that we permit the use of C++ exceptions. In practice, we will avoid writing code that might throw exceptions in our performance-sensitive kernels because we have great experience in writing these kernels such that there will be no checkable error conditions.

Future: Update these to account for new recommendations in the CppCoreGuidelines[11] following the best practice and wisdom of the larger modern C++ developer community, who largely share our interests in correctness and high performance, by default and by construction.

Major features developed over the course of BioExcel-2

Over the course of BioExcel-2, several high value features have been added to the core codebase, available to all users of GROMACS. A full list for each of the releases during BioExcel-2 can be found on https://manual.gromacs.org. New features include and are not limited to:

- Added support for SYCL to offload calculations to Intel devices and hipSYCL to offload to modern AMD devices

[8] https://manual.gromacs.org/nightly/dev-manual/index.html#developer-guide

[9] https://manual.gromacs.org/nightly/dev-manual/style.html

[10] https://google.github.io/styleguide/cppguide.html

[11] https://github.com/isocpp/CppCoreGuidelines

- Full offload of long-range electrostatic interactions with PME on supported acceleration devices, including support for free energy calculations
- Offloading of the calculation of bonded interactions to all supported accelerators
- Offloading of the integration/constraint calculation to NVIDIA devices
- Improving support for handling GROMACS from Python using gmxapi
- Support for ARM Scalable Vector Extensions (SVE)
- Fitting of atoms based on electron density maps (density guided simulations)
- Support using running multiple time stepping
- Support for new coupling algorithms
- Extended functionality of AWH to include free energy perturbation simulations
- New nonbonded and listed forces library NB-LIB
- New QM/MM interface with CP2K
- Support for more forms of pulling using transformation pull coordinates
- New formulation of soft-core interactions for free energy calculations

Development process details for HADDOCK

HADDOCK is an integrative, information-driven docking approach that supports a large variety of data from biochemical, biophysical and bioinformatics methods such as: mutagenesis data, NMR chemical shift perturbations, cross-links from MS, EPR-derived distances, cryo-EM densities, and bioinformatics co-evolution predictions. The software is made available through a user-friendly web interface, which has attracted a large user community worldwide (27 000+ users[12]) from more than 135 countries. HADDOCK has demonstrated a strong performance in the blind docking experiment CAPRI, belonging to the best performing approaches and is currently the most cited software in the protein-protein docking field. The HADDOCK software is available both for download and as a webserver. The stable release of the software and web server is HADDOCK 2.4 with a new modular version HADDOCK 3.0 currently in pre-release stage.

Internally, HADDOCK is separated in two components, whose interdependence is depicted in Figure 3. A high-level Python layer manages job submission and pre- and post-processing steps. The web server implementation adds several pre- and post-processing steps to this Python layer that are currently not available in the standalone version of the software. The performance-critical molecular mechanics computations and structural and energetics analyses are performed using the CNS (Crystallography & NMR System) engine (http://cns-online.org), which includes its own scripting language. This high-level language means that in most of the cases, development does not require coding in the original language of CNS (FORTRAN77). Some FORTRAN77 routines specific to HADDOCK are provided with the standalone software and require re-compiling the CNS executable. The CNS software comes with its own suite of tests that should be run after recompilation. As such, we trust that if those tests are successfully passed, CNS will provide reliable results.

HADDOCK is deeply interconnected via a series of scripts which, in turn, call CNS protocols. Despite being optimized for performance, this organization is rigid and does not allow easy modification of the current workflow strategy or creation of custom workflows. For HADDOCK to become a platform where users can freely define their custom docking pipelines by leveraging HADDOCK's routines, we are under BioExcel developing HADDOCK3. HADDOCK3 has seen a complete rewrite of the whole Python shell towards an integrated library-like infrastructure. The various CNS stages of HADDOCK2.4 have been split into separate modules so that they can be used independently as building blocks of custom workflows. HADDOCK3's architecture allows users to configure docking pipelines, create new calculation/analysis modules (even CNS-independent), and easily integrate third-party software. Under the umbrella of BioExcel2, HADDOCK3 is being developed as a free and open source (Apache 2) software (https://github.com/haddocking/haddock3). While currently not production-ready, advanced users can already test the latest features and follow the development process live in the repository. A number of example workflows corresponding to different scenarios are already available.

[12] https://wenmr.science.uu.nl/stats

We have introduced changes to enhance the FAIRness (https://www.go-fair.org/fair-principles) of the HADDOCK code. Specifically, the HADDOCK3 GitHub repository in which its whole source is open to the public. Also, the development process by core developers is thoroughly organized in “Issues” and “Pull Request”, granting full traceability of the history of changes. Also, this process is open to the public, for which we can use HADDOCK3 development as a source of training, mentoring, and teaching best practices of open-source scientific software. Traceability applies to code, documentation, and continuous integration processes. The latter are also as open as possible as explained further in the text. Finally, this openness should encourage contributions from third parties (advanced users and other developers). To date, two third party software packages have already been integrated in HADDOCK3.

In HADDOCK3, we prime dependency-free code and pure-python code (except for the CNS scripts). In other words, unless strictly necessary, developers should look forward to reimplementing needed functionalities natively by taking (licensing allowing) code from other sources or reimplementing the needed algorithms using the Python standard library (Numpy is welcomed for numerical operations). We have opted for a dependency-free approach (as much as possible) to facilitate higher layers of software integration to be able to operate with HADDOCK3 easily.

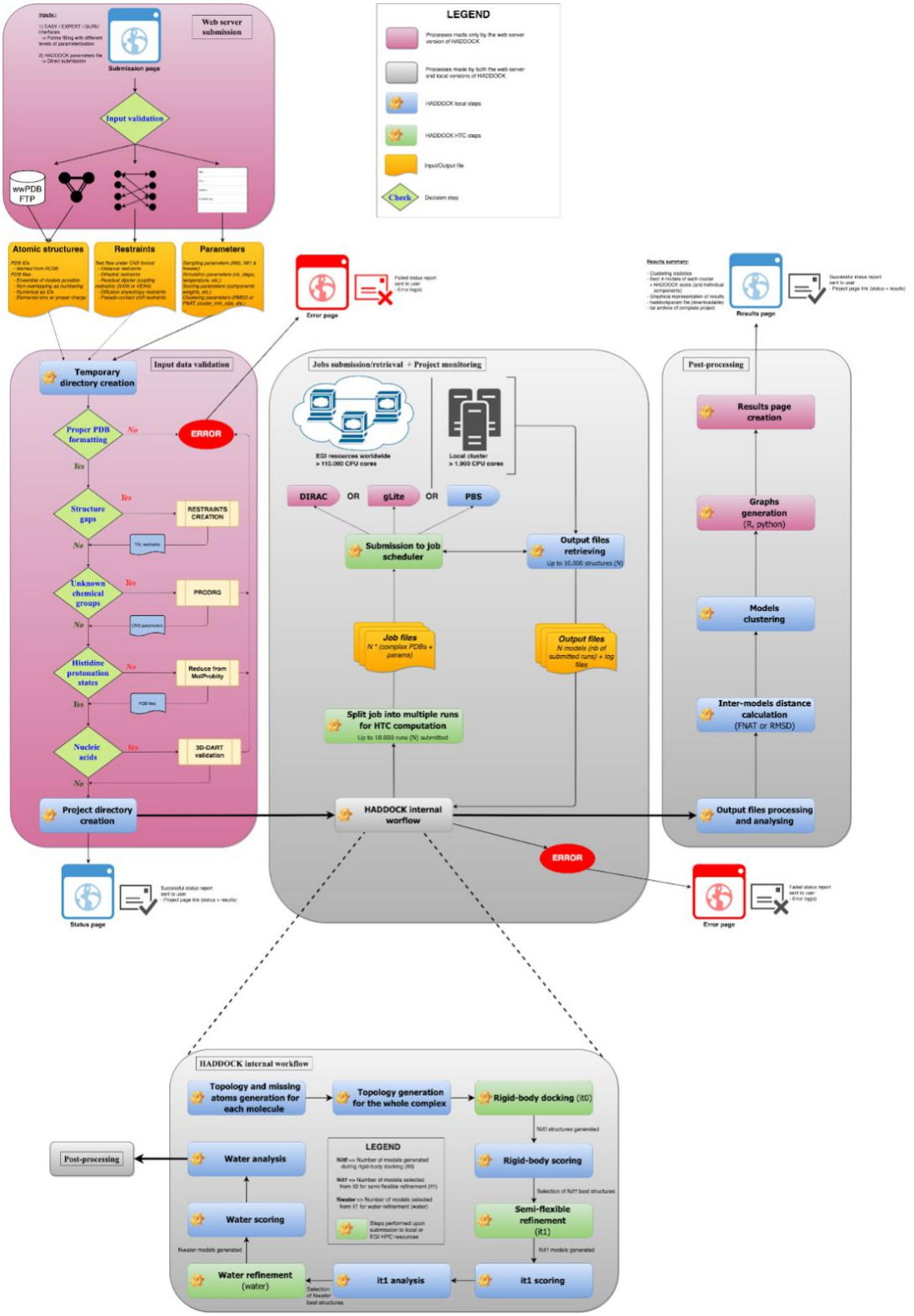

Figure 3. Workflow description of (A) the HADDOCK web portal, and (B) the internal workflow of HADDOCK itself.

HADDOCK2.X

The workflow used by the HADDOCK2.X developers when modifying the code is depicted in Figure 4. The parsing of data, structures, and the actual docking calculations and analysis are coded in roughly 80,000 lines of CNS scripts. A description of the various stages of the docking protocol, with reference to the CNS scripts called, is provided in the online manual. Changes to those scripts require running a suite of tests to ensure that the code does not produce wrong results for pre-existing scenarios (see testing requirements below). Any new feature added to the code requires an associated test case. Changes at the Python level are less frequent and mostly required to accommodate new types of experimental data or changes to the general workflow (e.g. number of molecules supported), as was the case with the release of the 2.4 version.

Most of the developments in HADDOCK over the years have been triggered by scientific questions posed by users and by the core team when working on collaborative projects. Smaller developments occur more often at the level of the force field and associated topologies, namely adding support for new molecules and amino-acids post-translational modifications. HADDOCK v2.X does not follow conventional semantic versioning, but a format that is more suited to represent the development cycle. Releases are made in the format of **2.MAJOR.YEAR-MONTH** once a substantial number of features and bug fixes have been addressed. The next major release v2.5 which is a port of HADDOCK to Python 3.8+ will adhere to the same custom convention.

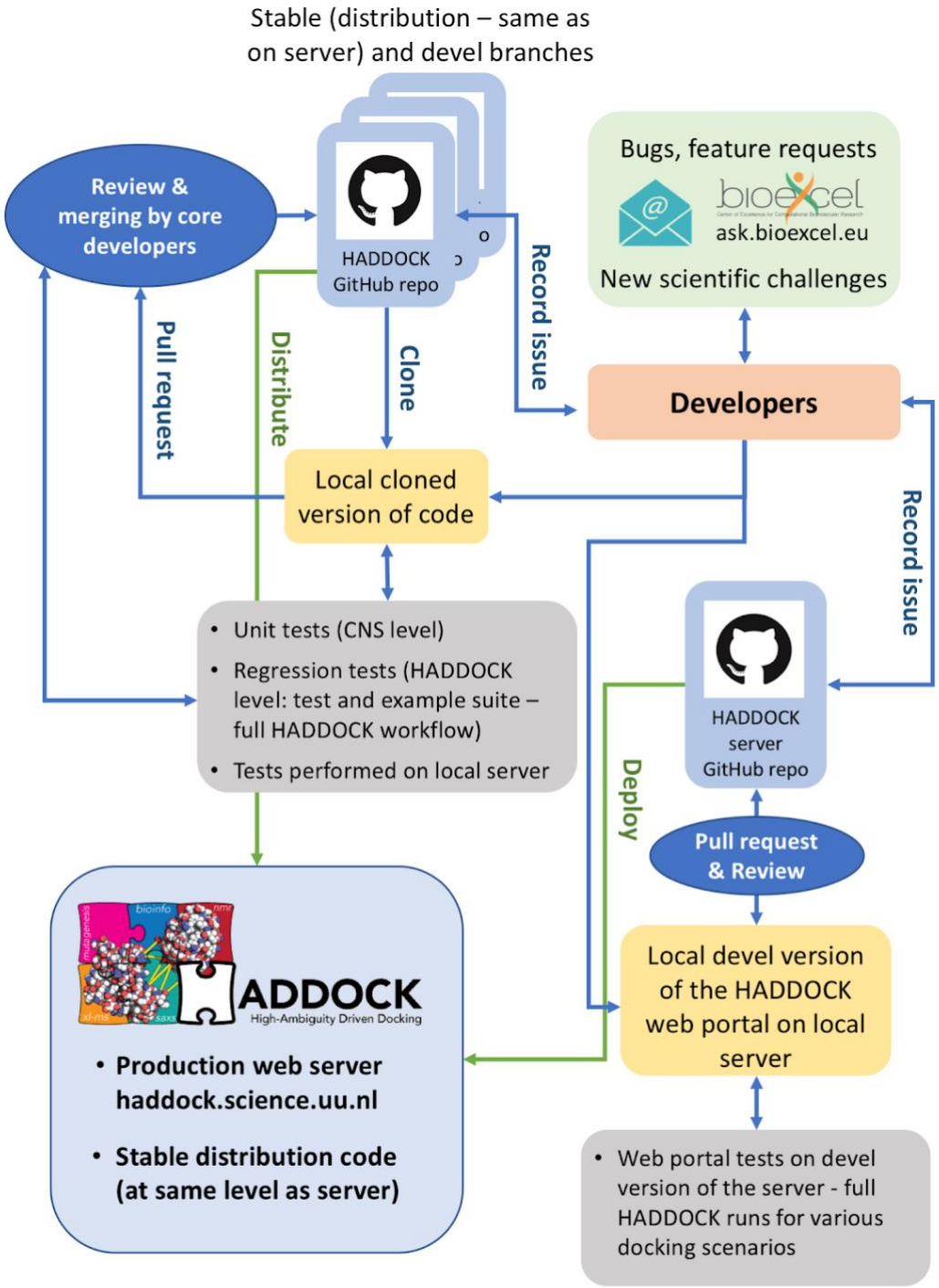

Figure 4. Overview of the HADDOCK development workflow and deployment of new features in the web server

HADDOCK3

The new HADDOCK version is being developed following the architecture of a Python library. Because of this new architecture, HADDOCK3 python code is no longer a combined set of scripts but instead a library of functionalities encapsulated and documented (functions and classes). Higher-level processes orchestrate the smaller and reusable pieces of code towards cooperative behavior. Command-line clients configure predefined operations at the user interface, such as running a workflow. Internal functions are shared and reused across this new library architecture. Unavoidably, new Python-code additions must be placed in their corresponding place within the library packages and python modules. In other words, singular scripts performing specific tasks are no longer accepted. Instead, these should be written in the form of command-line clients that use the internals of the HADDOCK3 library. *This new design aims to enhance code reusing, testing, capacity to add new features without severe code rearrangements, and documentation.*

In detail, we divided the new Python-shell architecture managing HADDOCK3 functionalities into independent building blocks, such as: non-variable physical parameters, command-line interfaces, plug-ins, CNS code for the actual simulations, and the functional modules configuring for each calculation stages. Thus, we can manage each of these blocks and blocks' pieces independently, adding or removing them without breaking the remaining of the software. Our module's parameters are stored within `default.yaml` files, which serve as examples of reusability and scalability. These files also contain the range limitations, types, documentation and serve as a single source of information for all HADDOCK3 associated projects.

Since the HADDOCK3 repository is open, developer contributors can fork our main repository, open a new branch and make Pull Request with their additions for review by core developers if they want to contribute to the main code, else developers are free to implement their features independently (see later in the text).

Retirement

Current: The distributed standalone version (v2.X) is kept as tied to the web server version, since this allows for a much smoother user support by pointing users to the web portal in case of local installation problems or setup issues and reproducible results

For HADDOCK v2.X we use a **MAJOR.YEAR-MONTH** version numbering system, where changes in the year-month do not break backwards compatibility. The current version is 2.4-January-2022. The previous major (released) version, 2.2 for which a web server is still operating at https://alcazar.science.uu.nl/services/HADDOCK2.2 will be discontinued by the end of 2022. Users and third-party developers making use of the HADDOCK2.2 web server will be given a six-month notice before the removal of the version from our web portal.

Future: Our participation in BioExcel has made us aware of the need for a clear policy in supporting software and web server versions. For future releases of the web server and associated standalone version, we will implement a retirement policy that will support and maintain a previous version of the software and web portal for a maximum of two and one years, respectively, after the release of a new version of those. While one year might seem rather long

for the web server, our web servers are not updated that often (except for minor fixes), and this is also needed since the duration of research projects might be quite long and it is important to allow users to complete their research work using one and the same version of the software for consistency.

Deprecation

Once a new version is released, it becomes the production version and the version before that enters in maintenance-mode, with only critical issues being addressed. Third party software developers relying on the web server will be notified directly after the release of a new version of the server.

Users with older versions of the software requesting support are encouraged to upgrade to the most current version. References and web server interfaces are documented with the latest changes of the production version.

Release Process

All components of the HADDOCK software are version-controlled via GitHub. Each major release is stored under its own branch. All code additions generate a new feature/bug branch and then a pull request is made prior to the changes being merged. Acceptance of a pull request requires the approval of at least one other core developer (although minor bug fixes and changes might be accepted and committed without this revision process). We require all changes to be properly documented, in terms of commit messages, and self-contained, to ease making release notes and reverting changes in case of problems.

Likewise HADDOCK2, in HADDOCK3 new code additions will enter the production code through the process of branching and pull requests. However, contrary to HADDOCK2, we won't keep individual branches for the different release versions. Instead, all versions will be archived in the GitHub "release" tab. HADDOCK3 should have a constant, small-increment basis progress, instead of bulk periodic additions.

Version control

The web machinery versioning is rolling release and follows that of the production version, every major release (v2.5, v2.6, etc) will be accompanied by a new instance of the server. This approach has the objective of ensuring reproducibility and usability of the HADDOCK features. For development purposes, we run a separate mirror instance of the web portal that is public and can be accessed to test new features. Additionally, since computations are typically distributed on a grid of heterogeneous servers distributed around Europe and the world, numerical reproducibility of docking results cannot be guaranteed. Users that wish such reproducibility will have to run a local version of HADDOCK. This does mean more manual intervention to perform the computations since the local version does not include all the validation, pre- and post-processing/analysis steps performed by the web portal. Some users have however commented

that they prefer to use the web portal since they know that in that way they always have access to the latest version of HADDOCK.

HADDOCK v3 will follow Semantic Versioning 2.0 conventions (major.minor.patch) that will cover the project as a whole. That is, changes in the Python code, the CNS code, the documentation, or any file in the HADDOCK3 repository will be followed by the corresponding version number increase. Every version will be stored on GitHub for traceability and reproducibility purposes.

Review Process

Several core HADDOCK v2.X developers and industry partners can submit pull requests in Github. Rights to accept commits will be given to a few selected core reviewers as well as the main reviewer (Alexandre Bonvin). The versioning system ensures that, even if a faulty commit would have been accepted, the code can be reverted to a previous commit.

Any new feature added to HADDOCK must be accompanied with an associated integration test and example runs. Further, the complete test and example suite should be executed and demonstrated to lead to similar results. For bug fixes and minor features, successful execution of the test suite should be demonstrated (see testing process below).

The HADDOCK3 code review process is similar to that of HADDOCK2. But, for the latter, we rely on GitHub actions to run the python unittests and the continuous integration (CI) pipelines as configured by tox (described later).

Feature request handling

New features are often added as a result of new challenges and scientific questions. In most cases users directly contact the developers via email (either directly via our university emails, or through our user support email: haddock.support@gmail.com or the http://ask.bioexcel.eu HADDOCK forum that provides another feedback/requirement mechanism. Considering the limited funding and the absence of core and permanent software developers, new feature requests from the users are prioritized based on the scientific interest and expected impact on the user community. These then generate new issues in the HADDOCK GitHub repository for the core software and in a separate GitHub repository for the web interface when necessary (addition of new features/parameters, etc).

Bug tracking

We are making use of the “issues” forum offered by GitHub to add and track bugs. Users can directly write their bug reports or request new features in the main GitHub (for HADDOCK3) which allows to classify issues in various categories. Code additions correcting bugs can refer to the issues raised in the forum, thus facilitating traceability.

Testing requirements

Any bug fixes should be validated by running the test suite in the local version and demonstrating that similar results are obtained.

Any new feature should be accompanied with a well-documented test and example with all required input data and an example output file.

Any feature/bug fix pushed to the web server should be first tested and validated on the development version of the web server, demonstrating that the full workflow is running properly, and similar results are obtained. For new features, a single, self-contained HADDOCK parameter file should be provided to allow simple testing via the "file upload" interface of the server.

Testing process

HADDOCK2.X

HADDOCK v2.X development is done in a private repository on GitHub. The *integration tests* are separated from the code on their own GitHub repository and these run the full set of supported features for the local version of the software with reduced settings to make sure that everything is working properly, allowing to execute the complete test suite in less than one hour on a laptop. *Examples* for various scenarios are separated from the code on their own GitHub repository. These run the full workflow with standard or optimal settings for the various scenarios to make sure that all the molecule types supported, and the various combinations of input data are properly working.

Next to testing the entire HADDOCK workflow, unit tests are defined for the CNS software used as the computational engine interpreting and executing the HADDOCK scripts. CNS comes with its own test suite that checks that the compiled executable properly works, testing all various commands, modules, and energy functions.

HADDOCK3

Regarding CNS and integration testing, we aim at having a similar process for HADDOCK3. All other tests and continuous integration routines are configured under tox and GitHub actions combined. In detail, HADDOCK3 has several testing environments: python unittests, python package building, code style, and (in the future) documentation build. Python unittests assert that all functionalities of the python shell library work as expected, either as individual components or combined elements. HADDOCK3 unittest suite is written with pytest and is defined under the tests folder. The Python package building tests are a series of routines that guarantee Python can convert the repository as a whole to an installable python package. The code-style tests ensure the python code follows a pre-defined writing style. Finally, documentation tests will ensure documentation pages render properly from docstrings and RST files to HTML pages. tox is an automation library, and we use it to ensure all HADDOCK3 developers run equal tests following global configurations defined by the core developers.

Developers can run these tests locally on their machines while developing new features. Detailed error messages are reported in case tests fail.

Finally, when developers propose new code to HADDOCK3 (Pull Request), the same tox-based tests run remotely and openly in the GitHub Action servers. In this way, code reviewers can also assert if tests pass and request changes if needed. With this implementation strategy, we use a single file (tox.ini) to define as many test environments as needed, where each test environment can be as complex (or simple) as necessary, and ensure all developers and servers run the same unified tests.

Development process details

Any new development should be linked to an identified feature documented as an issue in GitHub. This allows for discussion, planning and review as the work on a feature progresses. The following steps will be followed:

1. Identify and describe a new feature/development by creating an issue in the HADDOCK GitHub repository (also possible directly by users for HADDOCK3).
2. Write a development/implementation plan and solicit feedback from the HADDOCK developers (eventually, if needed from other BioExcel experts).
3. Plan tests, implementing either regression tests in case of changes at the CNS script level in HADDOCK (i.e. new tests/examples running the complete HADDOCK workflow), or unit tests if new functionalities are added to CNS (i.e. self-contained CNS test scripts testing the specific new feature implemented).
4. Attempt implementation and testing by creating a new branch of the current version of HADDOCK
5. Finish code, add test and example to the test/example suite
6. Run the full test/example suite for validation
7. Submit to main developers for review (typically two reviewers are required)
8. Merge code upon approval
9. Run the entire test/example suite in the merged version for final validation

Any developments affecting the web server will follow the same mechanism, with the additional requirement that single, self-contained *haddockparameter* files be generated to test the new feature via the “file upload” interface of the web server. Validation using all test cases is first performed on the development instance of the web server. Depending on the impact of a new feature and the number of new features implemented, either a minor update of the software/portal or a major release will be rolled out.

Integration approach

The HADDOCK developers will work on their own separate branch of the code using standard GitHub mechanisms for this. Once all tests and validation have been successfully performed on the local branch a pull request is done in GitHub to merge the changes in the main branch. GitHub automatically checks if an automated merge can be performed, which needs to be approved by the main reviewer. If this is not the case, manual merging will be required, which will involve the developer putting in the merge request and the main/core reviewer(s). This effectively means merging the current official branch and the developer's branch into a new branch that will require another round of tests to make sure the integration was successful. Only then will it be merged into the main branch. This applies to HADDOCK v2.x, v3.x and the web server.

Code commenting/standards/styles

Code is commented with a view to explaining what the code is doing and why a specific part was added, rather than how. If it is not obvious how the code is working, then the naming of variables and methods should be improved.

HADDOCK v2.X consists mainly of Python code and CNS scripts, with some additional C code and a collection of various scripts (csh, sh, awk, perl).

HADDOCK v3 code style is checked with flake8 and isort and the configuration in our repository. Developers can execute `tox -e lint` to run the code style tests on the whole code. Docstring documentation follows the numpydoc guidelines. We will use those combined with Sphinx to automatically generate the library internal documentation in HTML.

For the CNS scripts, proper indenting, commenting and clear variable naming should be followed.

Web server

For the web server, any update should first be successfully tested on the development version of the web server, ensuring that the various scenarios provided in the example set are all successfully running on the web server, including the pre- and post-processing steps not offered by the standalone version of the software. For this, version-specific, self-contained single *haddockparameter* files should be used via the "file upload" interface of the web server. Those files will be added to the test suite on GitHub.

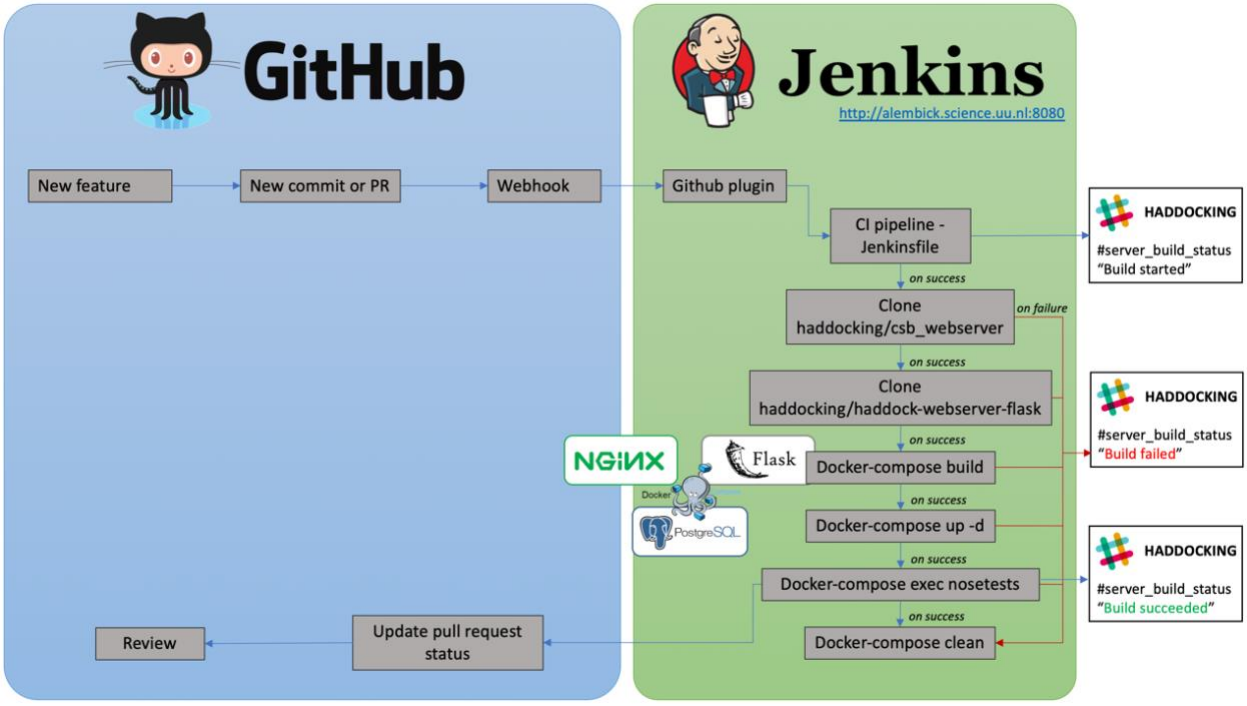

Figure 5. HADDOCK Web Server Continuous Delivery pipeline

The web server has a continuous integration pipeline that links GitHub, our repository hosting service, and Jenkins, a self-hosted automation server (Figure 5). This is all automated and event-driven, triggering feedback to the developers via a dedicated Slack channel and emails.

Jenkins uses Pipelines to set up an environment as close as possible from the production environment. This is possible thanks to the web server containerization via Docker. Deploying the Docker containers that bundle the web server within Jenkins creates an environment similar to production. Then, a test suite runs and finally a status code is sent to GitHub to report the success or failure of the current code.

Web Server Design

The web server (Figure 6) design has since the start been geared toward user-friendliness, providing foldable menus to hide forms when not used. It uses the Flask framework (full stack) with some JavaScript functions for extra usability. Furthermore, several access levels to the portal are offered that only expose a limited or more extended set of input options depending on the complexity of the scenario. Usability and scenario-specific web forms is something that will remain central in any future development.

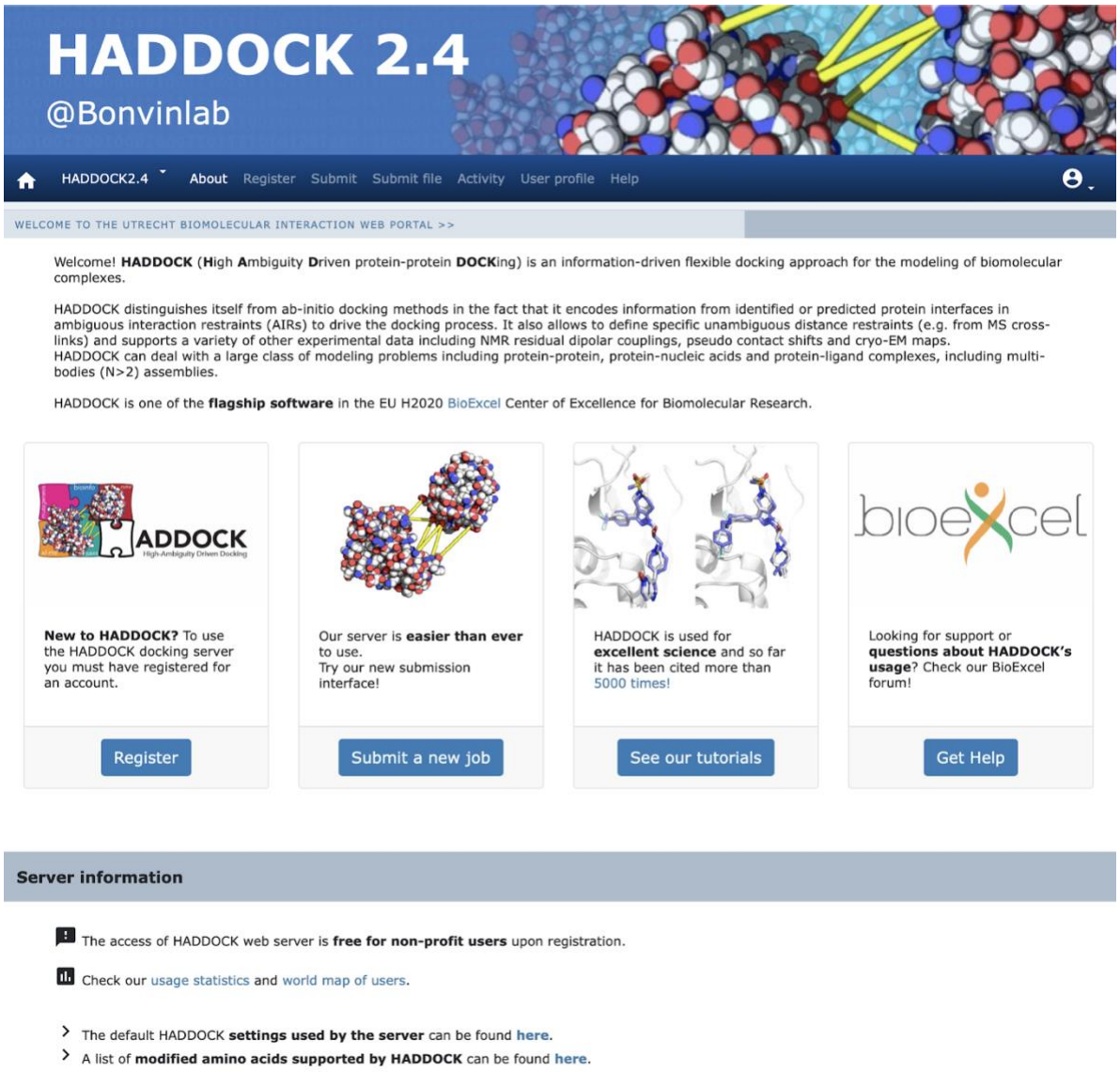

Figure 6. HADDOCK2.4 web server landing page (https://wenmr.science.uu.nl/haddock2.4/)

The next iteration of the web server will follow the next major release v2.5. To this end we are currently developing a newer, modern interface that uses current web development technologies that adapts the single-page application philosophy to the context of our scientific software (Figure 6). The user will be given the following options to submit a run: 1) Upload a parameter file, 2) select from a preset scenario or 3) define all parameters from scratch.

Currently the provided parameter file is a JSON file that the user can edit manually and then submit to the file-interface. At this stage it will be validated. With the new interface we will provide the option for the user to upload a parameter file, which will be read by a javascript function that will use the values of the uploaded file to populate the form, in such a way that the interface will act as an embedded parameter file editor. This feature has been requested by users and is not available in the current version of the web server. These changes will provide the users

with a much friendlier, less error-prone and more productive manner of re-running a previous simulation.

In the current 2.4 server interface we also suggest optimal parameters for a given simulation based on the molecule type identified in the system. These optimal values are based on our own observations and benchmarks. We will take advantage of the new interface and provide the users with preset scenarios based on the most relevant topics we observe in the user support channels (ask.bioexcel.eu) such as, for example, ab initio, small molecule or glycan docking. The current refinement interface will be replaced by such presets, which will simplify the server interface and increase its maintainability. The interface will read files analogous to the aforementioned parameter files that were manually built by us, using the more optimal values for each scenario. This approach will provide the users a more comprehensive and robust set of parameters so that the software can be utilized to its full extent.

Lastly, users can choose to build a run from scratch, similarly to what it is done for the current v2.4, selecting each parameter according to its own research question. Creating simulation setups in this manner will expose users to either the full options of HADDOCK in case of GURU-level users or to simplified views for EASY users. We also aim to improve usability by improving the parameter description tooltips and adding relevant links to the Best Practice Guide.

HADDOCK Virtual Research Environment

The web server is the main way in which users execute HADDOCK. The source code (v2.4) has been distributed to 1.200+ users, whilst the web service contains 27.500+ users registered, executing around 300 simulations per day. Via the web interface users have access to computation resources and to post-processing steps not available via the command line. For HADDOCK3, our goal is to expand on the usability, reproducibility and FAIRness of the web server and move towards a Virtual Research Environment (VRE). This VRE will be divided into 3 main components: a workflow builder, an execution middleware and a data analysis platform (Figure 7). It will use HADDOCK3 as an engine and take full advantage of its modular workflows.

Using the *workflow builder* users will be able to interactively build custom made workflows tailored to their own specific needs and have seamless access to all HADDOCK functionalities. This web interface (in development) uses the ReactJS framework and can be dynamically deployed either via web-hosting or locally, retrieving parameters and modules directly from a pre-generated catalog containing all necessary information[13].

[13] https://github.com/i-VRESSE/workflow-builder

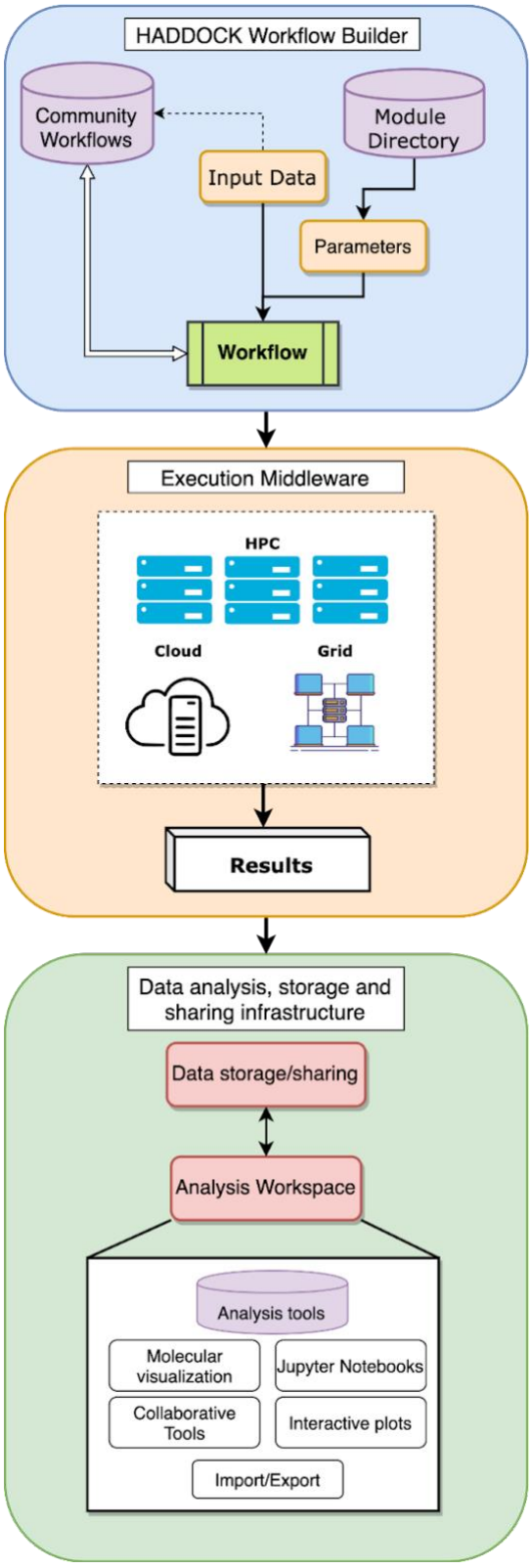

Figure 7. Diagram of the overall architecture of the Virtual Research Environment

The *execution middleware* will build on what is in place for the HADDOCK web server and expand its execution functionalities. GRID/HPC executions will be handled effectively and we will add the option to use cloud resources for computations. These cloud resources can be either private (provided by the user) or shared via one of the partners such as European Open Science Cloud.

Via the VRE users will also be able to run *data analysis*, this will be an expansion of the post-processing step and will include manually curated scripts created by the HADDOCK team as well as routines created by users. The analysis workspace will allow users to share their analysis with collaborators and offer reproducibility tools in such a way that can be added to publications.

The VRE is being developed in the context of the eTEC 2021 grant, together with the Netherlands eScience Center.

Copyright and License

HADDOCK v2.X is distributed under a proprietary license which allows use for academic research without fee, but an agreement of a commercial license is needed for commercial applications. However, the HADDOCK3 version has been released under the Apache 2 License in the context of the BioExcel 2 Center of Excellence. We are currently studying agile ways to carry commercial licenses via a possible paid support

Development process details for CPMD QM/MM

CPMD is an *ab initio* (or quantum) molecular dynamics software package developed and maintained by IBM Research. It employs the density functional theory to solve the many-electron problem and a plane-wave basis set to expand the wave function. To perform the time evolution of the quantum system it implements both the Born-Oppenheimer and the Car-Parrinello molecular dynamics approaches.[14] It was reported to be able to scale up to several million threads on a 16.32 PFLOPS supercomputer with almost 100% efficiency.[15]

However, even such highly-parallel *ab initio* code can treat relatively small systems (few thousands of atoms), while many applications, in particular biological ones, require processing systems containing hundreds of thousands and millions of atoms. Such simulations are not feasible using full quantum methods and multiscale approaches, where different parts of the systems are treated at different levels of accuracy and resolution, are employed. The QM/MM methods are those two-layer multiscale approaches where the (small) region that requires the

[14] Car, R. and M. Parrinello, *Unified Approach for Molecular Dynamics and Density-Functional Theory.* Physical Review Letters, 1985. **55**(22): p. 2471-2474. https://doi.org/10.1103/PhysRevLett.55.2471

[15] Weber, V., et al., *Shedding Light on Lithium/Air Batteries Using Millions of Threads on the BG/Q Supercomputer*, in *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 2014, IEEE Computer Society. p. 735-744. https://doi.org/10.1109/IPDPS.2014.81

most accurate description (QM part) is treated at quantum level, while the rest of the system (MM part) is described with a classical force field.

CPMD has a built-in QM/MM interface[16] designed and developed by the group of Prof. Ursula Röthlisberger at EPFL (Lausanne, Switzerland) that couples CPMD with the old GROMOS96 (van Gunsteren et al. 1996) classical molecular dynamics routines to perform a QM/MM molecular dynamics simulation. However, this implementation suffers from a set of drawbacks. First, the force fields that can be used to describe the MM part are only GROMOS96 and AMBER99. Then, the scalability of the MM part is limited because the version of GROMOS96 employed has only OpenMP parallelization without MPI part. Therefore, only one node can be used for classical part which leads to scaling issues in case of large MM parts on massively parallel architectures (like IBM Blue Gene). Finally, GROMOS96 has a commercial license, which requires a user to buy it before using the QM/MM interface in CPMD.

To address those issues a novel QM/MM interface is being developed. The designed workflow of a QM/MM-enabled simulation based on CPMD is shown in Figure 8.

[16] Laio, A., J. VandeVondele, and U. Rothlisberger, A Hamiltonian electrostatic coupling scheme for hybrid Car–Parrinello molecular dynamics simulations. *The Journal of Chemical Physics,* 2002. **116**(16): p. 6941-6947. https://doi.org/10.1063/1.1462041

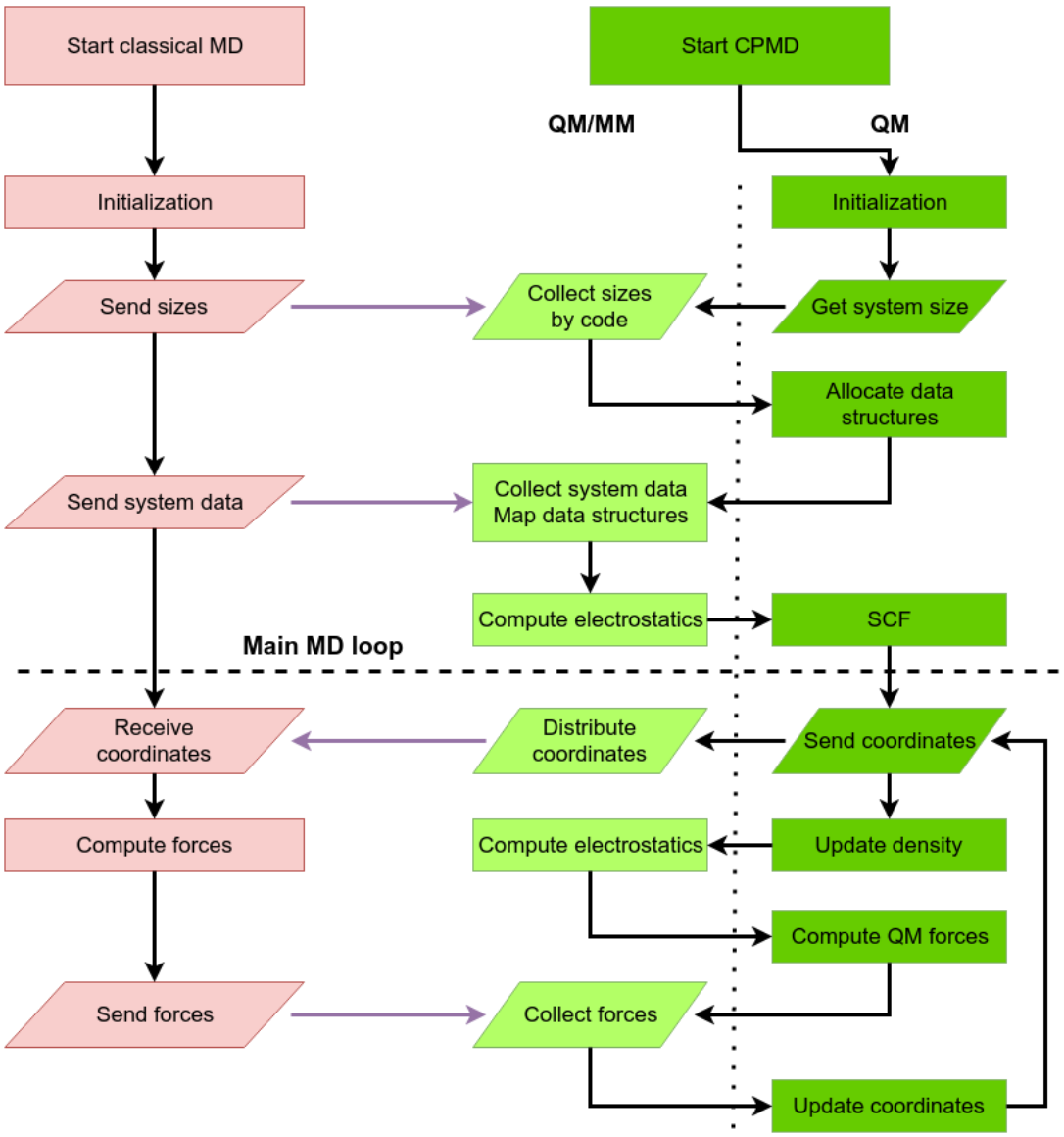

Figure 8. CPMD QM/MM-enabled simulation workflow

Copyright and License

After some discussion within the developer team, it has decided to release the new QM/MM interface under the Lesser GNU General Public License (LGPL) v2.1. In fact, this choice prevents the new QM/MM interface to be distributed inside the tarball of the CPMD source code that can be downloaded from the CPMD website (www.cpmd.org), due to the restrictions of the IBM general license under which CPMD is made available. Different websites, including the BioExcel one (https://bioexcel.eu/software/code-repositories/), has been selected for releasing the patches that allow introducing the needed changes in the code, and full information about the new QM/MM interface, including how to enable the new QM/MM interface of the code will be

inserted in the CPMD manual which is the main source of information about CPMD in the CPMD community.

Version control

Git is used for version control. For the initial stage of the development the merge-based Github workflow is employed. A stable master branch will be used to create builds. All changes are done on a separate branch. Feature branches are merged into master branch through merge requests. Before the review process an automatic build and testing procedure is triggered. A three-level testing procedure is used. The first level of testing is the unit testing checking the validity of code changes introduced in separate modules. After the successful passing of unit tests, a set of integration tests is triggered. Finally, the test coverage analysis is performed using *gcov*[17]. The testing procedure is handled by the Gitlab continuous integration system. An approval of at least two members of the development team is needed to fulfill the merge request.

Feature request handling and bug tracking

Future: An issue tracking system will be used for reported bugs and feature requests when the program will be released for the user test phase. At the moment a Gitlab issues tracker is planned to be used, with a possible fallback to Redmine in case the Gitlab functionality proves to be not sufficient.

Testing requirements

A PFUnit unit-testing framework is used to write automated tests. All functional parts of the code (i.e. not constructors, accessors, etc.) must be covered by unit tests. Moreover, proposed changes should not lower the test coverage percentage. Failing to comply with these requirements results in the rejection of the proposed changes. A feature incorporating multiple code units should have a sensible integration test provided. Lack of an integration test without a valid reason stated will also result in the code rejection. Tests are run automatically using Gitlab continuous integration system on several build-servers, running on different hardware and software to assure the portability of the code.

Development process details

Current: The team of developers is currently a small number of autonomous academics in different Institutions that coordinate the general development lines through short face-to-face or Skype meetings.

The code development process is organized following the Github workflow, based on a Gitlab service. After the code changes are committed to the repository an automatic build is triggered,

[17] https://gcc.gnu.org/onlinedocs/gcc/Gcov.html

running automated tests following the build. If tests are successful a code review process is started. If the proposed changes are approved by two other members of the team then these changes are merged into a master branch. This workflow is depicted in Figure 6.

Future: We are planning to build up a new dedicated website for the QM/MM interface with the release of the next stable and optimized version of code within the end of the BioExcel project this year. This website is intended to be the main reference for the users of the new QM/MM interface. In particular, it will contain and update the list of the new features and bug reports from users collected by the developers (through the CPMD mailing list, ask.bioexcel.eu forum, private emails), together the release version of the code where the feature is implemented or the bug fixed. If the number of users will increase considerably, a webpage where the users can directly report issues will be implemented as well. The website will also be the primary source of information for external developers who want to contribute to the development: since we expect only a small number of external contributors, they will be fully integrated in the current development process.

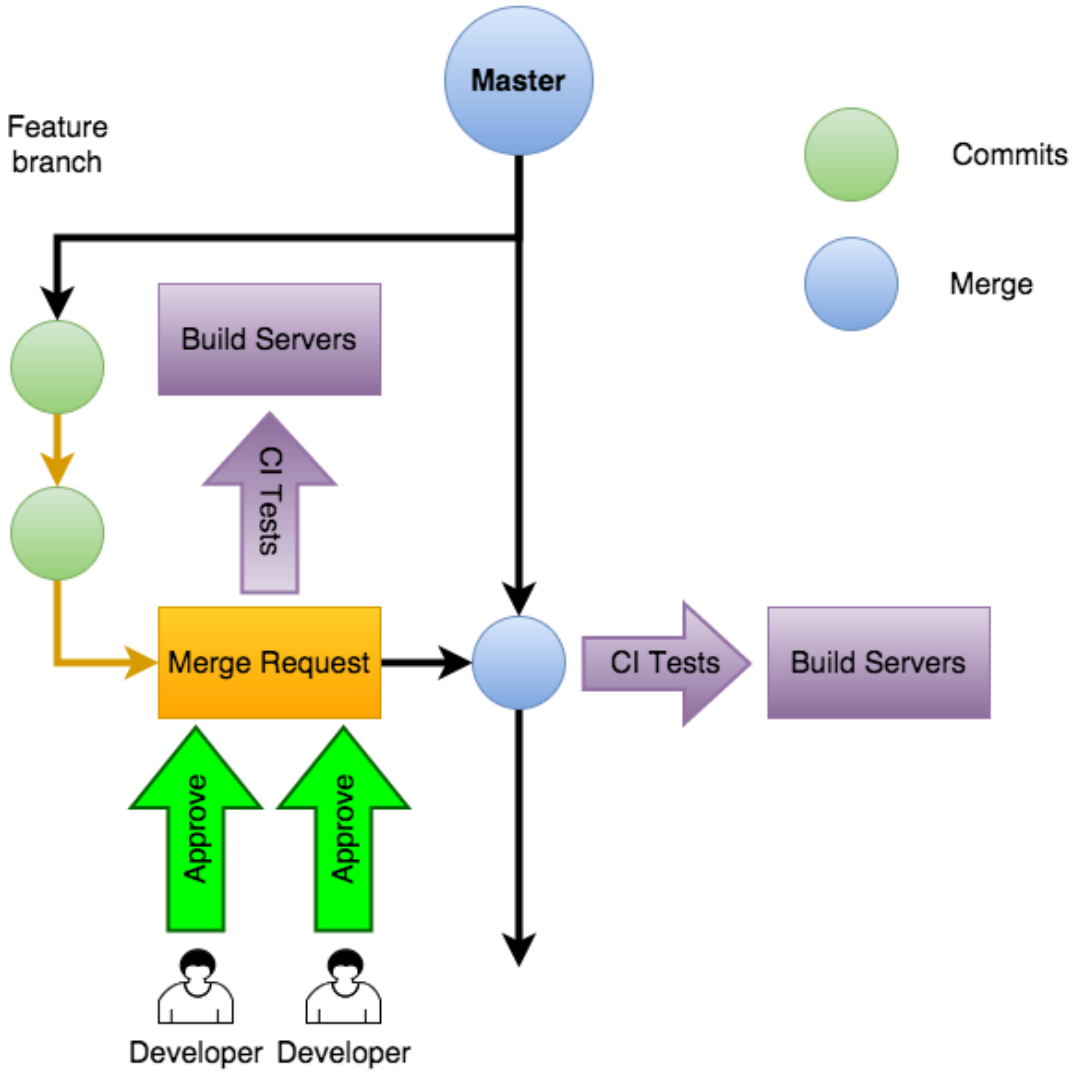

Figure 9. QM/MM development workflow during initial MiMiC development phase

Code commenting

Every type and subroutine (except constructors and accessor methods) has Doxygen-format comments stating its purpose (i.e. explaining what code is doing).

Coding standards and styles

The part of the code integrated in CPMD is being developed to comply with Fortran2008 standard. Coding style is loosely based on the Fortran90 Best Practices[18]. Execution flow can only be controlled using if and do statements - usage of goto statements is strictly forbidden. On top

[18] http://www.fortran90.org/src/best-practices.html

of that a usage of OOP techniques is encouraged (though sometimes is limited due to the language issues or performance considerations) to make the maintenance of the code easier.

The communication library is being developed in C++, with style loosely based upon the Google C++ Style Guide[19].

Finally, the minor changes in GROMACS are implemented by following the GROMACS style described in the corresponding section above.

Design

The design of the novel QM/MM interface has been devised having in mind the execution model based on the multiple program multiple data (MPMD) approach. In this model CPMD and the classical molecular dynamics code (e.g. GROMACS) run independently, occasionally exchanging data. To establish the data connection an ad-hoc communication library is used. The advantages of this approach are i) the possibility to couple CPMD virtually with any classical molecular dynamics code (and consequently to employ any force field in the MM part) with minimal code intervention; ii) benefitting from the parallelization scheme of both the classical molecular dynamics code and CPMD; iii) overcoming possible licensing conflicts between the CPMD's proprietary license and the one of the classical molecular dynamics code.

[19] https://google.github.io/styleguide/cppguide.html

Development process details for QM/MM Interface between GROMACS and CP2K

CP2K is a state-of-the-art free and open-source European package for Density Functional Theory computations and related methods, with a wide range of capabilities and excellent performance. CP2K is used to address problems in materials science, computational chemistry and provides a framework for multiple modelling methods including hybrid calculations for large biological systems. While the performance of CP2K, as well as its permissive licensing terms make it a favorable application for Life Science related simulations.[20] However, the limited documentation, the large number of options available via a complex input file format, and the size of the code (900,000 lines of Fortran 95), make it challenging for new users (and developers) to adopt this code. Because existing expertise is currently isolated in individual research groups, there is an enormous potential to increase the user base by sharing know-how with the community.

To address this challenge and introduce CP2K into the biomolecular simulation community, we have developed an interface to share forces and energies between CP2K and the widely popular and user-friendly GROMACS molecular dynamics program. With the new GROMACS/CP2K interface, it is now possible to perform scalable high-performance QM/MM simulations of biological systems under periodic boundary conditions. To improve user experience and usability topologies are automatically converted from standard MM to QM/MM and default CP2K input parameters sets have been integrated into the *grompp* tool of GROMACS (Figure 1). While these functionalities target new users, who benefit from a smooth and straightforward setup of their systems for QM/MM simulation, the interface design also allows advanced users to modify most of the individual CP2K parameters for improved performance or for employing non-standard methods.

The new QM/MM interface is based on the MDModules framework of GROMACS, which provides a unified, modularized and maintainable platform for introducing additional (external) forces into an MD simulation. A typical GROMACS *mdrun* simulation workflow for QM/MM simulations with the new MDModule-based CPK2K interface is illustrated in Figure 2. Interactions with the core *mdrun* functionality as well as requests for receiving data from other GROMACS modules are implemented via the MDModules notifications mechanism. Interaction with CP2K routines is achieved via the *libcp2k* functionality for transferring data and executing QM calculations, which is linked with *mdrun* into a single executable.

[20] Thomas D. Kühne, et.al., *CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations.* J. Chem. Phys., 2020, **152**, p 194103 https://doi.org/10.1063/5.0007045

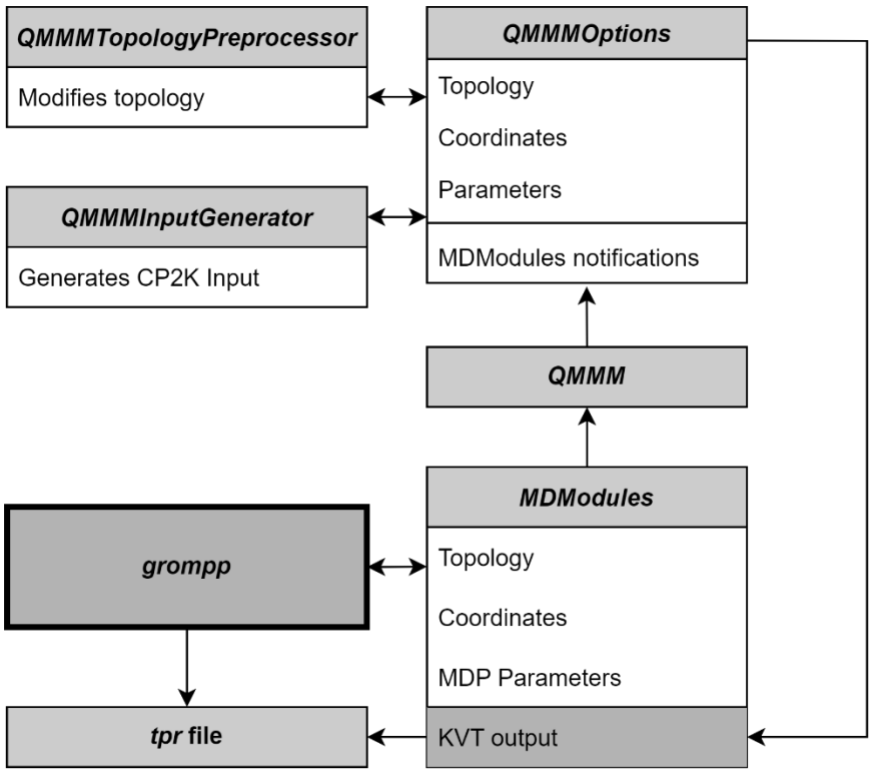

Figure 10. GROMACS grompp workflow with CP2K QM/MM interface

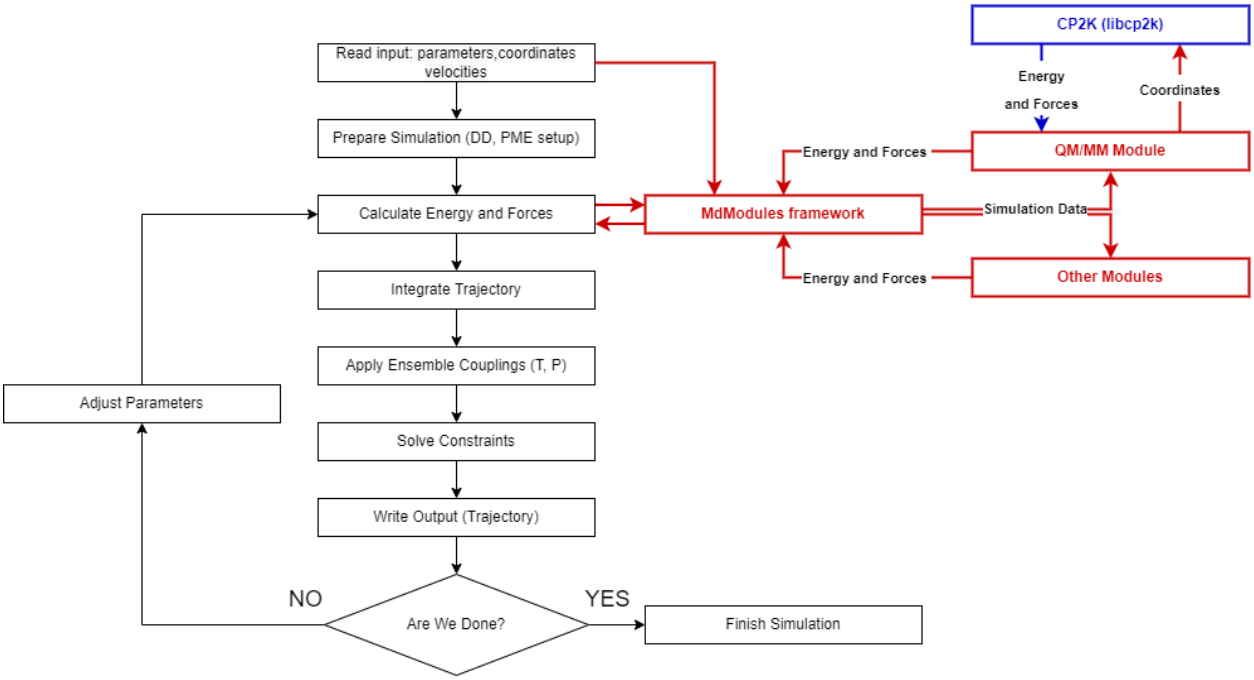

Figure 11. GROMACS MD workflow with CP2K QM/MM interface. Interface part is shown in red.

Copyright and License

The new QM/MM interface is distributed as a part of GROMACS molecular dynamics package under the Lesser GNU General Public License (LGPL) v2.1. This is an intentional choice to balance our requirements for all extensions to the main codebase to be freely available (which is important for our academic impact), while still enabling commercial applications to link with. CP2K is also an open-source software distributed under GPL license, which allows users to give the code a try, and to make modifications as needed.

Version control

Gitlab of GROMACS is directly used for version control. At all stages of the development the merge request based Gitlab workflow is employed. A stable master branch is used to create releases. All changes are done on separate feature branches. These branches are merged into master branch through merge requests. Before the review process an automatic build and testing procedure is triggered. Major testing is performed as unit tests that check the validity of code changes introduced in separate modules. In addition, tests employing linking with the most recent version of *libcp2k* are performed. The whole testing procedure is handled by the Gitlab continuous integration system. Review and approval of at least two members of the GROMACS development team is needed to complete a merge request.

Feature request handling and bug tracking

An issue tracking system of GROMACS on Gitlab[21] is used to track issues, report bugs and create feature requests. In addition, a Bioexcel forum dedicated to QM/MM simulations[22] is used to respond to user requests and problems. In case of bugs or feature requests, information from the forum is forwarded to the GROMACS Gitlab and processed there in a standardized way with Gtilab issue tracking system.

Testing requirements

A GoogleTest C++ driver[23] is used to write automated unit tests. All functional parts of the code (including constructors, accessors, etc.) are covered by unit tests. Moreover, proposed changes should not lower the test coverage percentage. Failing to comply with these requirements results in the rejection of the proposed changes. Tests are run automatically using Gitlab continuous integration system. In addition, code should fulfill all testing requirements for the GROMACS package as stated in the respecting section above.

---

[21] https://gitlab.com/gromacs/gromacs/-/issues

[22] https://ask.bioexcel.eu/c/qmmm-biosim/20

[23] https://github.com/google/googletest

Development process details

Currently, there is only one active developer of the new QM/MM interface and priority is given to fixing bugs in the release version of the interface. Thus, new features will be implemented as soon as they pass testing and the code is ready to be merged into the master branch of GROMACS. There are two main drivers for further developments: (i) expanding scope of scientific questions, which can be treated using QM/MM and (ii) improving usability of the interface based on feedback from users obtained from e.g., forums, e-mail, interviews, conferences, workshops, *etc.* Release of the new features and bug fixes is synchronized with the official GROMACS release schedule.

Deprecation

At the current stage of the development, features are not intended to be deprecated. However, if some of the functionality will be significantly updated, users will be allowed to reproduce an old version behavior of it as well.

Code commenting

Every type and subroutine are accompanied with Doxygen-style comments stating its purpose (i.e., explaining what code is doing). All documentation is written according to the GROMACS standards.[24]

Coding standards and styles

The QM/MM interface is developed in C++, with a style loosely based on the Google C++ Style Guide[25]. As a part of the GROMACS package it follows the GROMACS coding style described in the corresponding section above and online.[26]

Design

The new QM/MM interface between the GROMACS and CP2K packages has been designed to follow the execution workflow of GROMACS. In the first step (Figure 1), the user produces a binary simulation file (*tpr*) with the *grompp* tool based on topology, coordinates, parameters, index and all other necessary files. During this pre-processing stage, the interface (i) transforms parameters into internal representation, (ii) makes topology modification, (iii) generates CP2K input and (iv) saves all data into the key-value tree (KVT), that will be incorporated into the simulation *tpr* file.

[24] https://manual.gromacs.org/current/dev-manual/doxygen.html

[25] https://google.github.io/styleguide/cppguide.html

[26] https://manual.gromacs.org/current/dev-manual/style.html

In the second step (Figure 2), *mdrun* is used to perform the simulation with the data stored on the *tpr* file. At the start, the QM/MM interface reads the relevant input data from the *tpr* file and uses it to initialize *libcp2k* and the MPI communicator (in case of MPI run). After initialization, the interface will wait for calls from the MDModules, which provides atomic coordinates at each step of the simulations. After receiving these coordinates, the interface performs the QM calculation with the relevant routines from the *libcp2k* library and returns the QM/MM forces to MDModules. This cycle continues until simulation is finished.

In our strategy, we try to pre-process as much data as possible with *grompp*. Examples of such data include: conversion of index groups into the arrays, topology modifications, CP2K input processing, *etc.* In contrast, the *mdrun* part of the code should be as efficient as possible in consumption of memory and computing time.

Development process details for PMX

PMX is a python based software package that allows automating the specialized free energy calculation setup for the GROMACS simulation engine. PMX enables calculation of free energy differences in proteins, DNA and arbitrary small molecules. Not only the initial steps of the hybrid structure and topology generation are supported, but also the full simulation workflows for the relative free energy calculations can be constructed based on the PMX classes and modules.

Originally, the PMX software was designed to automate GROMACS based alchemical amino acid mutations in proteins[27,28]. Subsequently, in the further developments PMX was expanded to support DNA mutations[29]. Also, attention was paid to providing user friendly access to the tools: for that purpose, protein and DNA mutation support was enabled via a webserver-based interface[30] [PMX4]. Following these developments, we focused on the development of another major PMX branch – alchemical ligand modifications, to facilitate drug design based on rigorous free energy calculations. In contrast to the amino acid and nucleic acid mutations, it is not possible to pre-generate mutational libraries tabulating all the necessary modifications, because of the vast chemical space that arbitrary ligand molecules span. In Bioexcel-2, to address this challenge, PMX implements an algorithm which allows identifying necessary transformations for any given ligand pair. All in all, PMX readily provides the full toolkit for

[27] Daniel Seeliger and Bert L. de Groot. (2010): **Protein thermostability calculations using alchemical free energy simulations.** *Biophys. J*. **98**:2309-2316 https://doi.org/10.1016/j.bpj.2010.01.051

[28] Vytautas Gapsys, Servaas Michielssens, Daniel Seeliger, and Bert L. de Groot (2015): **pmx: Automated protein structure and topology generation for alchemical perturbations**. *J. Comput. Chem.* **36**:348-354 https://doi.org/10.1002/jcc.23804

[29] Vytautas Gapsys, Bert L. de Groot (2017): **Alchemical Free Energy Calculations for Nucleotide Mutations in Protein-DNA Complexes**. *J. Chem. Theor. Comput.* **13**:6275-6289 https://doi.org/10.1021/acs.jctc.7b00849

[30] Vytautas Gapsys, Bert L de Groot (2017): **pmx Webserver: A User Friendly Interface for Alchemistry**. *J. Chem. Inf. Model*. **57**(2): 109-114 https://doi.org/10.1021/acs.jcim.6b00498

alchemical ligand modifications, including a ligand perturbation map builder, an atom mapper between the pairs of ligands and hybrid topology generation.

Retirement

Currently, two major PMX branches are being supported. The ‘master’ branch is based on the original Python2 implementation: only bug fixes for this version are provided without any new features added. The main development is performed in the ‘develop’ branch which is based on Python3. The Python2 based version will be completely retired once the last dependencies on this branch are removed: to this date those include PMX webserver and ligand perturbation map builder.

Deprecation

Up to this point, no feature deprecation in PMX was required.

Release process

PMX follows a feature driven release cycle. With the critical mass of new features implemented, the major version is released. This is accompanied with a manuscript in an academic journal describing the advancements in the new release. All the bug-fixes, new features and other modifications are tracked by the GitHub’s version control process.

Review process

A single developer reviews PMX code. Additions to the code by a few external developers are also reviewed by the main developer.

Feature request handling

New feature development is guided by the scientific interest of the developers and collaborators. External requests via GitHub, Ask BioExcel Forum and e-mail are considered on an individual basis.

Bug tracking

Users report bugs on GitHub, Ask BioExcel Forum and via e-mail.

Testing requirements

The correctness of the ligand hybrid structure/topology generation is ensured by performing free energy calculations across a thermodynamic cycle where each branch can be computed explicitly, with the sum of the individual legs providing a rigorous and absolute reference of zero free energy. Any deviation from zero is a sensitive and robust indication of topology or protocol deficiencies. A thermodynamic cycle for hydration free energies of two ligands fulfills

such a requirement. It is important to ensure that the tested hybrid structures/topologies include diverse chemistry and varying numbers of mapped atoms.

Testing process

The first step in testing ligand hybrid structure/topology generation was by computing hundreds of free energy differences across thermodynamic cycles constructed from the FreeSolv database. The database comprises ligands of varying chemistry and size, thus allowing to test different atom mapping scenarios.

Subsequently, we have performed numerous large scale alchemical free energy calculation for protein-ligand binding[31,32,33]. The calculations allowed identifying edge cases, leading to software improvement.

Development process details

In addition to the formerly defined main forces driving PMX development – scientific questions and changes in GROMACS – we have also paid particular attention to the non-equilibrium free energy calculation workflow development. Prompted by the user requests, we have released a detailed description of the alchemical calculation setup for protein mutations[34]. [pmx7] Also, specialized classes within PMX were implemented to facilitate convenient interaction with the standard GROMACS based molecular dynamics simulation setup commands. In turn, this allowed constructing workflows for alchemical free energy calculations of ligand modifications in protein-ligand complexes that scales readily to use cases of hundreds of calculations.

Integration approach

After the review by the developer the code is integrated into the 'develop' branch of PMX. The bug-fixes are also integrated into the 'master' branch.

---

[31] Eddy Elisée, Vytautas Gapsys, Nawel Mele, Ludovic Chaput, Edithe Selwa, Bert L de Groot, Bogdan I Iorga (2019) : **Performance evaluation of molecular docking and free energy calculations protocols using the D3R Grand Challenge 4 dataset**. J. Comput. Aided Mol. 33(12) 1031–1043 https://doi.org/10.1007/s10822-019-00232-w

[32] Vytautas Gapsys, Laura Perez-Benito, Matteo Aldeghi, Daniel Seeliger, Herman van Vlijmen, Gary Tresadern, Bert L de Groot (2020): **Large scale relative protein ligand binding affinities using non-equilibrium alchemy**. *Chemical Science*. 11: 1140-1152 https://doi.org/10.1039/C9SC03754C

[33] Vytautas Gapsys, David F. Hahn, Gary Tresadern, David L. Mobley, Markus Rampp, Bert L. de Groot (2022): **Pre-exascale computing of protein-ligand binding free energies with open source software for drug design.** *J. Chem. Inf. Model.* 62(5) https://doi.org/10.1021/acs.jcim.1c01445

[34] Matteo Aldeghi, Bert L de Groot, Vytautas Gapsys (2019):. **Accurate Calculation of Free Energy Changes upon Amino Acid Mutation**. in *Computational Methods in Protein Evolution*. 19-47. https://doi.org/10.1007/978-1-4939-8736-8_2

Code commenting/standards/styles

Over the period of BioExcel 2 we have considerably improved on the details of comments within the code base. During the transition to Python 3 we have also aimed to adhere to the corresponding programming style for Python[35].

Architecture

The main PMX classes for handling structures and topologies remain unchanged. In addition, with the particular focus shifting towards alchemical ligand modifications, new large classes for ligand atom mapping (LigandAtomMapping) and hybrid ligand topology (LigandHybridTopology) construction were created. The overall workflow for ligand modification requires both classes used sequentially. Firstly, atom mapping between a pair of arbitrary ligands is established (Figure 12). This step follows two main pathways: graph based maximum substructure matching and distance-based atom matching after alignment. An external RDKit library is required by this class. Also, numerous rules are encoded to ensure atom mapping which would allow for a subsequent stable molecular dynamics simulation. Afterwards, the information about the mapped atoms is processed by the LigandHybridTopology class to create a hybrid structure and topology.

[35] Guido van Rossum, Barry Warsaw, Nick Coghlan: **PEP 8 – Style Guide for Python Code**. https://peps.python.org/pep-0008/

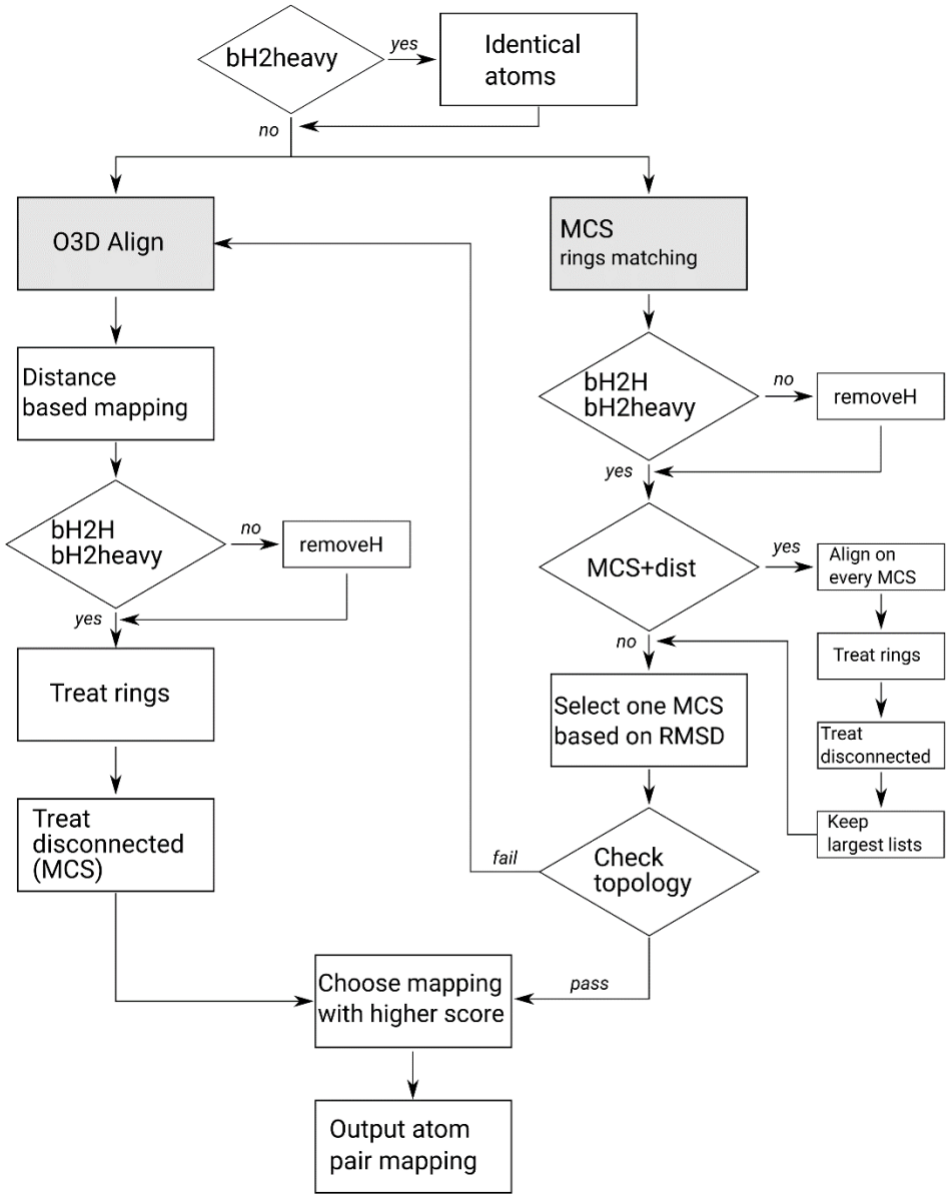

Figure 12. Atom mapping (LigandAtomMapping class) for pairs of arbitrary molecules. The procedure allows creating a list of atom pairs which are further used in generating hybrid structure and topology to be used in alchemical free energy calculations.

Design

There are multiple ways to interact with PMX. Firstly, the PMX binary can be called from the command line interface to perform to make use of the main PMX functionalities. Alternatively, the PMX webserver can be used to perform amino acid and nucleic acid mutations. Finally, all the PMX classes can be directly imported into a python script allowing for an efficient

construction of customized workflows. This presents a convenient method to incorporating PMX into computational chemistry workflow manager systems, e.g. Icolos[36].

Concluding Remarks

We hope that these snapshots of the development processes for the BioExcel codes have been useful. These real-world projects are using tools and methods that work, but they have evolved over time and will continue to change to suit the individual projects' needs. Ongoing incremental investment into code development process is needed for any project to provide software that continues to meet its users' needs, while remaining correct, portable, and performant.

Author List

Contributing authors from the BioExcel Centre of Excellence, along with their academic affiliations:

- Forschungszentrum Jülich (Jülich Research Centre), Germany:
  Emiliano Ippoliti
- KTH (Royal Technical University), Sweden:
  Mark J. Abraham, Paul Bauer, Rossen Apostolov, Erwin Laure, Berk Hess, Erik Lindahl
- Max Planck Institute, Germany:
  Bert L. de Groot, Vytautas Gapsys
- Utrecht University, The Netherlands:
  Alexandre M.J.J. Bonvin, Rodrigo Vargas Honorato, João M. Correia Teixeira, Mikael Trellet[37], Adrien S.J. Melquiond, João P.G.L.M. Rodrigues[38]
- University of Jyväskyla, Finland:
  Dmitry Morozov, Gerrit Groenhof

[36] Moore JH, Bauer MR, Guo J, Patronov A, Engkvist O, Margreitter C. Icolos (2022): **A workflow manager for structure based post-processing of de novo generated small molecules.** *ChemRxiv.* https://doi.org/10.26434/chemrxiv-2022-sjcp3

[37] Now at Fluigent Smart Microfluidics, Paris, France

[38] Now at Schrödinger, New York, USA