IMPROVING BIODYNAMO BUILD SYSTEM

AUGUST 2019

AUTHOR(S):
Giovanni De Toni

SUPERVISOR(S):
Lukas Breitwieser
Fons Rademakers

CERN openlab Summer Student Report 2019

ABSTRACT

When developing new programs or scientific libraries most of the efforts are focused on providing efficient algorithms, the state-of-the-art techniques and maximum flexibility. However, in order for a new project to be effective, one of the top priorities must be how the final user will obtain our software.

Sure enough, the first approach of the users usually is how to build/install the provided software. If the necessary steps/prerequisites to install the project are missing or not working, then we will lose a possible new user (or even many more). This happens especially if we are targeting users who are not experienced with DevOps operations (e.g. how to develop software and which tools are needed). For this purpose, the documentation must also be clear and concise, it has to provide code examples and possible use cases and it has to state clearly the installation procedure.

This report details the work on the BioDynaMo project (a biological large-scale simulator) which targeted their entire build and testing system. The project was about improving the previous automatic build procedure to increase usability and to provide a more maintainable codebase. The updates targeted the mechanism which takes care of installing the required packages and the complete build process, such to make the operations more robust and flexible for the final users.

Some final conclusions were drawn from this experience to be useful to define future refactoring processes. Moreover, future works and improvements are also discussed which can be used to direct the next development steps of BioDynaMo.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 What is BioDynaMo?

BioDynaMo is a framework to create, run and visualize **3D agent-based biological simulations** [1]. It was devised to be efficient, modular and able to scale smoothly to sustain large scale simulations. Since biological tissue dynamics require extensive computing resources, this library is able to fully exploit different hardware to produce the best results. Ultimately, BioDynaMo will be able to run on cloud computing infrastructure as well as on standard HPC or desktop computers. This will be possible because BioDynaMo harnesses the power of the most recent technologies and software engineering techniques. Moreover, it is constantly developed and updated to match the state-of-the-art scientific know-how. It offers also data visualization capabilities to produce images, interactive graphs which can be useful to show the results of a given simulation.

BioDynaMo can be used to model many kinds of biological process. Its user-friendly interface enables the researchers to easily develop customized simulations. For instance, some of the current BioDynamo's applications are focused towards **oncology research** (e.g. predict how a certain kind of tumour will evolve) and **brain research** (e.g simulate neurons and their growth). Figure 1 shows two examples of simulations and visualizations which can be done using BioDynaMo.

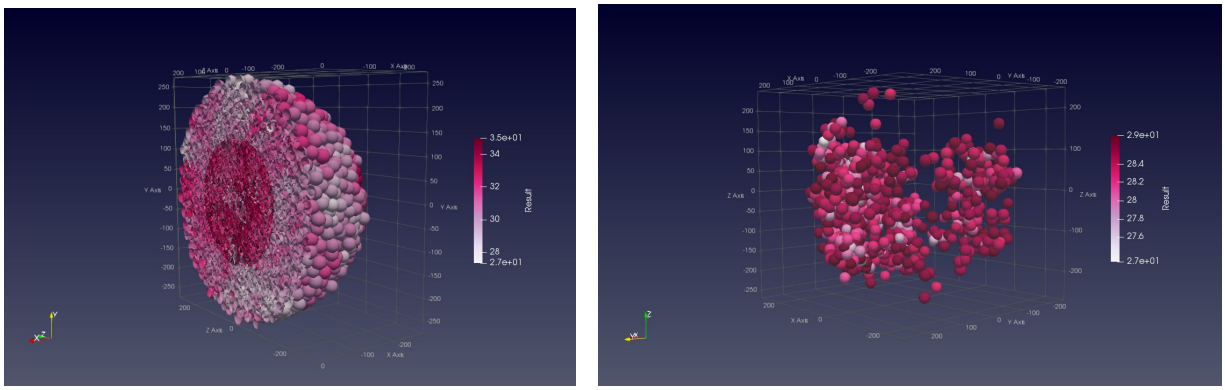

Figure 1. *(Left) This is an example simulation realized with BioDynaMo. It represents a tumour growth inside healthy tissue. Here we do not see the entire tumour, because it was split in half to better observe the phenomenon. (Right) Cell division example.*

1.2 Technical Details

BioDynaMo's core is written in **C++** and it provides a useful Python **CLI (Command Line Interface)** which abstracts the user from several low-level details. Moreover, the code is written to take advantage of parallelization when possible. External libraries as OpenMP are used to reach this goal. Data visualization features are offered using **ParaView** [2]. Moreover, BioDynaMo exploits also the **ROOT** library, CERN's official scientific toolkit [3], for saving simulations to disk to provide backup and restore functionalities [4].

The library is developed using the most recent techniques. **Automatic Continuous Integration** (e.g. Travis CI) is used to increase the code quality and to find possible bugs or issues which could impact user experience and the effectiveness of the simulations.

BioDynaMo is an open-source project and all the codebase is freely available on **Github** [5], one of the largest software hosts globally. Therefore, the project is open to peer review and it also benefits from direct contact with the user base. The users are able to contact directly the developers to report bugs or new features they want to be added to the framework. Moreover, thanks to the Pull Request mechanisms, end users may have a direct impact on the development of the library, since they can easily contribute to the codebase.

2 OBJECTIVES

When I first started working on the project, in order to build BioDynaMo, the procedure was the one described in Figure 2. It is possible to see that the process is made by several steps which each corresponds to several different tasks. The main purpose of this project was to refactor the build system of BioDynaMo such to improve the user experience. I mostly focused on the inner steps of the overall procedure and I tried to improve them by refactoring the existing code and by employing modern CMake techniques. Listing 1 also shows the exact commands the user had to perform to install BioDynaMo.

The practical objectives were many. I will present a few of them here to give an idea of the kind of task I have been working on:

1. Improve the automatic procedure to satisfy BioDynaMo prerequisites (e.g., packages that are needed to correctly build the library);
2. Enable building BioDynaMo without sourcing an environment script first;
3. Improve the automated procedure to install the library BioDynaMo by making it more robust;
4. Provide better documentation to the final users;
5. Make the make install step optional to be able to use the framework from the build library;

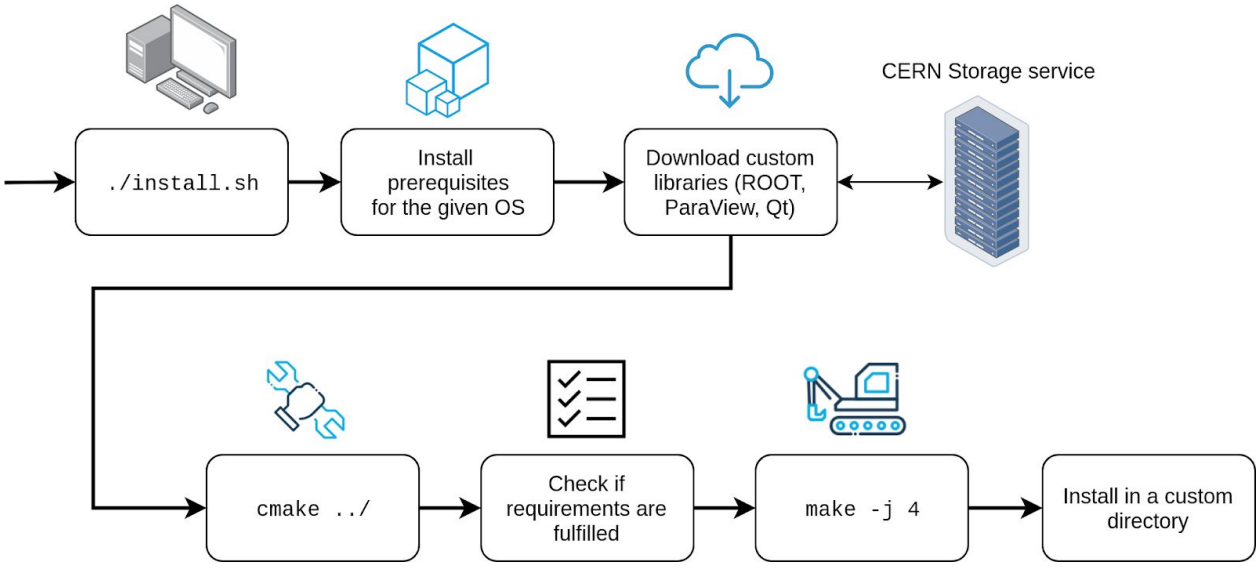

Figure 2: *The sequence of install steps for BioDynaMo. Once the install script is run, all of them will be done automatically. The script will also take care of installing the software in the user machine.*

```
git clone https://github.com/BioDynaMo/biodynamo
cd biodynamo

./prerequesites.sh
# follow the instructions of the script:
source <path-to-bdm-installation>/biodynamo-env.sh

mkdir build && cd build
cmake .. && make -j4
make install
```

Listing 1: *Original procedure used to install BioDynaMo. We can immediately see that the user has to run several scripts before the actual make command. Moreover, the user has to source a bash script which will provide information about the location of certain packages. We will show how this can be done automatically using only CMake.*

3 IMPLEMENTATION

3.1 Technology used

The build system of BioDynaMo is written using **CMake** [6], which is a set of tool mainly used to test, build and install the software. CMake was chosen since it specifically targets C++ software projects. The testing architecture is done using **CTest** and other tools, for instance, valgrind [7] for memory analysis or clang-tidy [8] for static code analysis. We employ also bash and python scripts to perform additional tasks which CMake was unable to do.

The software development process uses a **Kanban-like approach** and **Trello boards** are used to keep track of the tasks (e.g. user stories) and to share information with the rest of the team. Moreover, weekly meeting (both formal and informal) are held to discuss the improvements which were being done and also to gain feedback directly from the users. A **Slack chat** is also used to quickly exchange information (e.g. code, images, links) and potentially to talk with remote software developers.

Git and **Github** are used as Version Control Systems and to share the produced code with the other software developers. All the code is tested both locally using **Docker containers** and both with the complete BioDynaMo test suite on **Travis CI**.

3.2 Initial Approach

Initially, the build process was devised as a **semi-automatic task**. The user would have still to install manually most the prerequisites, but BioDynaMo would have given some hints about what was potentially missing. The devised workflow was the following:

1. The user downloads BioDynaMo from the Github official repository page;
2. The user runs `cmake` and the procedure will check if all requirements are satisfied;
3. If some of the requirements are not satisfied, CMake will produce two bash files (called `prerequisites-required.sh` and `prerequisites-optional.sh`) which can be then used by the user to install the missing packages;

4. Once the packages are installed, the user must run `cmake` again and then he/she will be finally able to build BioDynaMo.

This procedure had to be compatible with all the supported operating systems. Therefore, for each of them, a list of prerequisites was specified. CMake would have read from these lists to suggest the user the appropriate scripts which need to be executed.

3.2.1 Shortcomings

This method had several shortcomings which we discovered while actually implementing the system. First of all, this new procedure may have confused the end-users, which would have had to execute many more commands with respect to the previous one. Secondly, we felt as if we were “fighting with the tool”. By using just CMake alone it was not so easy to implement the automatic generation of the two bash scripts. This happens because the default CMake behaviour when a dependency is not found is to halt and return an error. We finally managed to produce a feasible solution, but the code itself was not exactly readable nor maintainable in the long term.

Therefore, we decided to stop developing this new solution and we tried to devise a new way.

3.3 Final Approach

The final approach removes unnecessary operations and simplifies the build procedure as much as possible. The main difference is that the user will first install all the necessary packages and then will run cmake.

3.3.1 Relocatable Installation Directory

Initially, it was not possible to use BioDynaMo directly from the build directory without issuing the `make install` command first. With the new procedure, the install step has become optional and BioDynaMo can be used directly from the directory where it was built, thus keeping together all the files into a unique location. This will make easier to update our software and also to uninstall it since it will be enough to remove the build directory. This was done by organizing the build directory with the same structure used when it is installed in its final location. Moreover, the `RPATH` of the generated executables was modified such to not contain hard-coded paths of runtime libraries. Everything will be read from the shell environment instead (for macOS a dynamic `RPATH` was employed).

3.3.2 Dependencies and Features Detection

CMake provides some primitives which enable us to find specific packages inside the system (e.g `find_package(omp)`). It has two ways to search for packages: the **Module Mode** and the **Config Mode**. The first mode uses specific files called `Find<Package>.cmake` which specifies where to look for the given `<Package>`. The location of them is defined by a variable called `CMAKE_MODULE_PATH`. CMake is shipped with already some of these files which covers the most famous dependencies. For instance, if we were looking for the Python interpreter, we will use the standard `FindPythonInterp.cmake` file. The second mode search for files called `<Package>Config.cmake` instead. These files are provided directly by the packages themselves and they are not located in the directories specified by `CMAKE_MODULE_PATH`.

For some of the dependencies of BioDynaMo, CMake does not provide an “official” `Find<Package>` file and therefore we had to write them on our own. We also updated to the `CMAKE_MODULE_PATH` variable to search also for our custom files. Moreover, we also tested the detection on all the supported OS. As a matter of fact, a certain package might be installed in a different location on different OS. Therefore we

needed to make sure that we had covered all the possibilities. For instance, on macOS, the user can employ different kinds of package manages (e.g., `fink`, `brew`, `port`, etc.) and each of them stores the installed packages in different directories with a different directory structure.

With the new implementation, CMake also provides the users with a list of all the dependencies it was able to find and the ones which are missing. This is useful because it tells exactly what went wrong and it simplifies the bug fixing procedure (e.g., by asking for the output of the cmake command, a developer can immediately see if the problem was caused by a missing dependency or by the user himself). Listing 2 shows an example of this CMake output.

```
############################ SUMMARY ##############################
-- The following packages were found:
 * PythonInterp (required version >= 3.5), Python executable. (REQUIRED)
 * pip, Package Manager System for Python. (REQUIRED)
 * MPI, OpenMPI, an Open Source Message Passing Interface. (REQUIRED)
 * GLUT, Open Source alternative to the OpenGL Utility Toolkit (GLUT) library.
(REQUIRED)
 * OpenMP, API that enables multi-platform shared memory multiprocessing
programming. (REQUIRED)
 * Threads, GNU C library POSIX threads implementation. (REQUIRED)
 * Git, Open Source Distributed Version Control System. (OPTIONAL)
 * ROOT, CERN's Modular Scientific Software Toolkit. (REQUIRED)
 * ClangTools, Standalone command-line tools that provide developer-oriented
functionalities. (OPTIONAL)
 * Numa, Simple API to the NUMA (Non-Uniform Memory Access) policy supported by...
```

Listing 2: *Sample output from the CMake procedure. Here we can see the (partial) list of packages which were found. Moreover, there is an indication which specifies which packages are required (or optional).*

Moreover, the CMake procedure will show to the user which features were enabled. For instance, if we are building BioDynaMo with the tests enabled, if we enabled ParaView, etc. The CMake procedure will also disable certain features of BioDynaMo if it detects that some dependencies are not met. For example, if the user wanted to use ParaView (which will be enabled by passing `-Dparaview=ON`) but ParaView was not installed in the system, then CMake will detect it and the framework will be not built with the visualization support.

3.3.3 ROOT and ParaView

BioDynaMo requires both ROOT and ParaView to provide some of its functionalities. However, the two libraries need to match a specific version to work correctly. Therefore, we added to CMake some additional checks to ensure correctness.

ROOT is checked against its version (if it is greater than a certain constant) and against the compiler used. BioDynaMo requires C++14 standard, but if ROOT was built with a compiler which does not support C++14 standard, then we would have a series of errors during the build process. Therefore, if ROOT does not satisfy the requirements, a new ROOT will be downloaded automatically from CERN's repository.

The same applies to the detection of ParaView. If ParaView is not compliant with BioDynaMo requisites then a custom version will be downloaded (together with the Qt5 code). This is because the official ParaView release is not compatible with BioDynaMo and therefore we provides our custom build.

To increase flexibility we kept the possibility for the final user to provide its personal ROOT and ParaView versions. This behaviour is achieved by providing two environmental variables (`ParaView_DIR` and `Qt5_DIR`) which can be set to point to the current location of the custom packages. In order to use your custom ROOT, you will need to just source it from its directory (by executing `source this root.sh`).

3.3.4 Compiler Detection

Different operating systems have different standard compilers. Generally, most of the Unix-based distributions come with the GNU Compiler suite (`gcc` and `g++`). Apart from macOS which uses the clang compiler.

CentOS does not provide by default a compiler which can support C++14. In addition, clang compiler shipped with XCode on macOS do not support native OpenMP, thus leading to compilation errors.

Therefore, the challenge was building a system within CMake which had to be able to determine the correct compiler for the project. Moreover, on macOS, this system must account for the fact that the compiler may not be the standard one. For instance, if we used `brew` to install the latest g++, then it will install it inside a custom directory.

By asking our users we discovered also that some of them need flexibility when we are talking about compilers. This means that they wanted to be able to specify their different compiler. This can be easily done by exporting some specific environmental variables (e.g `CXX` and `CC`). Listing 3 shows an example of how it can be achieved.

```
cd biodynamo
export CXX=/opt/local/bin/clang++-mp-8.0
export C=/opt/local/bin/clang++-mp-8.0
./install.sh
```

Listing 3*: BioDynaMo can be built by specifying the compiler. Here we are using clang 8.0 instead of the standard one.*

4 RESULTS

This project helped to improve the overall usability of BioDynaMo during the first phases of its usage. More specifically, it made easier for a first-time user to access to the library and to compile it. The complete refactor of the CMake infrastructure generated a less convoluted and more maintainable codebase, which can be easily extended, for instance, to support several other operating systems. The CMake code was updated to more modern techniques and some anti-pattern where removed. Everything was also extensively tested and validated both with the users and with the continuous integrations tools on the four supported operating systems: Ubuntu 16.04, Ubuntu 18.04, macOS and CentOS 7. In addition, the documentation was updated to reflect the changes and more code examples were added to easily show to the end-users which are the steps required. Figure 3 shows the new BioDynaMo build system.

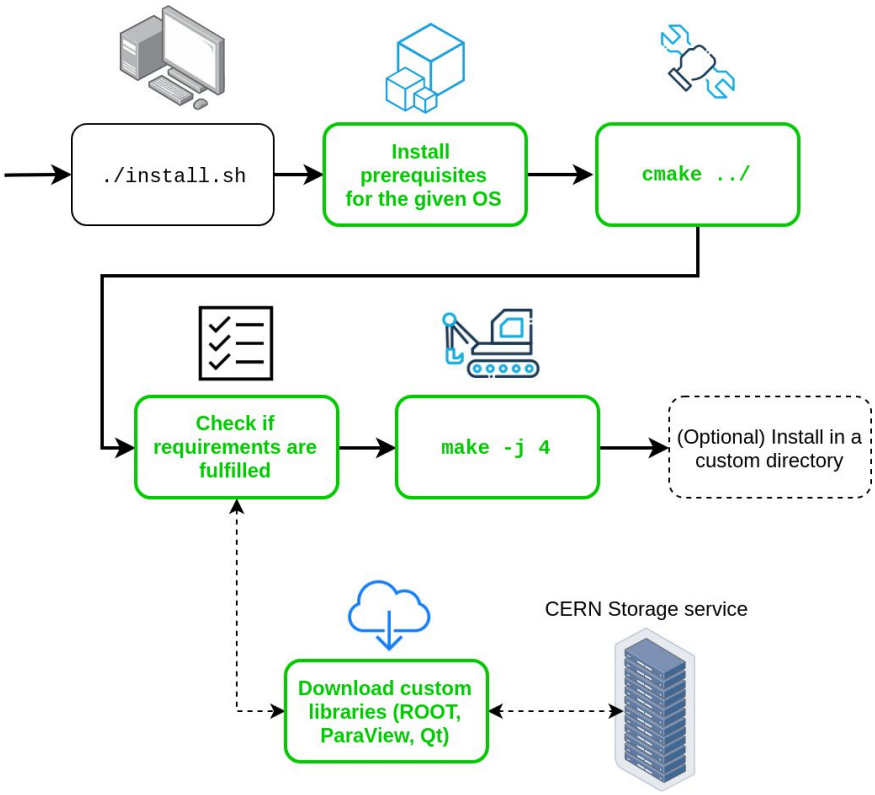

Figure 3: *The new BioDynaMo build system. As you can clearly see, it is not entirely different from the original one, but all the inner steps were reworked such to be more robust, flexible and less prone to errors.*

5 FURTHER AND FUTURE IMPROVEMENTS

5.1 Extensive Testing

Despite our efforts, there are still quite a few tasks which would improve the quality of BioDynaMo even further. One of these basic steps will be testing even more extensively all the features realized. We strived in order to rule out all possible errors and to catch most of the corner case which could result in a failure during the build process. However, this is a gigantic task which can be hardly even defined. Each user uses a **different machine** and each of them may have a **different configuration**. Therefore it is indeed difficult to be sure that no more issues will arise in the future.

Fortunately, the solution will come directly from the users itself. Thanks to the fact that BioDynaMo is available on Github, users will be able to send us bug reports and they will autonomously discover possible issues with the new procedure. Obviously, we hope this will not happen so often, but it is the fastest method to obtain useful information.

5.3 Windows Support

Currently, Windows still accounts for **87%** of the market share [9], while MacOS and generally Unix like systems account for the remaining 13%. Therefore, since BioDynaMo is not available on Windows, we are losing a great part of the potential user base. Many researchers still use Windows for their daily jobs and they will be able to leverage the features of the library. As a future plan, we need to seriously consider to make BioDynaMo available for Windows-like system.

6 REFERENCES

1. Breitwieser L, Bauer R, Di Meglio A, Johard L, Kaiser M, Manca M, Mazzara M, Rademakers F, Talanov M. The BioDynaMo project: Creating a platform for large-scale reproducible biological simulations. In: CEUR Workshop Proceedings. 2016, Manchester, UK: CEUR-WS.
2. ParaView, https://www.paraview.org/.
3. Rene Brun and Fons Rademakers, ROOT - An Object-Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch/.
4. Hesam, Ahmad, & Rademakers, Fons. (2016, October 7). Integrating ROOT I/O in BioDynaMo Brain Simulation Platform. Zenodo. http://doi.org/10.5281/zenodo.159523.
5. BioDynaMo (Biological Dynamics Modeller), https://github.com/BioDynaMo/biodynamo.
6. CMake, https://cmake.org/.
7. Valgrind, http://www.valgrind.org/.
8. Clang Tools, https://clang.llvm.org/docs/ClangTools.html.
9. Operating System Market Share, https://netmarketshare.com/operating-system-market-share.aspx.