ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-FR2A002

A DIGITAL TWIN FOR NEUTRON INSTRUMENTS

S. Nourbakhsh*, Y. Le Goc, P. Mutti
Institut Laue-Langevin, Grenoble, France

Abstract

Research infrastructures and facility users have manifested an increasing interest in data from virtual experiments. Simulated data are an important ingredient for instrument scientists to develop and optimize current and future instruments, for external users to train in the usage of the instrument control system (ICS), for scientists in quantifying and reducing instrumental effects when analysing acquired data. Furthermore large sets of simulated data are also a necessary ingredient for the development of surrogate models for faster and more accurate simulation, reduction and analysis of the data.

The development of a Digital Twin (DT) of an instrument can answer such different needs with a single unified approach wrapping in a user-friendly envelop the knowledge about the instrument physical description, the specific of the simulation packages and their interaction, and the high performing computing setup.

In this article we will present the general architecture of the DT prototype developed at the Institut Laue-Langevin (ILL) in the framework of the PaNOSC European project in close collaboration with other research facilities (ESS, European XFEL). The communication patterns (based on ZeroMQ) and interaction between the NOMAD control system, simulation software (McStas), instrument description and configuration, process management (Cameo) will be detailed.

The adoption of FAIR Principles (Findability, Accessibility, Interoperability, and Reusability) for data formats and policies in combination with open-source software make it a sustainable project both for development and maintenance in the mid and long-term.

INTRODUCTION

The development of a DT at the ILL can play a significant role in better preparation of experiments ahead of time and hence either reduce the amount of beam time needed for each experiment or an improvement in the acquired data with better suited instrument settings for the specific experiment. The objective is to enrich the offer of user tools with the possibility to run a virtual experiment with an ILL instrument. ILL's DT is designed to be used by users with no knowledge about simulations, providing a user experience almost unchanged with respect to what they get when running a real experiment. All technicalities are hidden from the user behind the familiar ICS interface where both the instrument configuration and acquisition workflow are set up. Simulated data are provided by the ICS in the same format as the real data, with no additional step to be taken before undergoing the reduction and analysis. A further requirement, in order to have a long term sustainability of the DT, is to use of state-of-the-art simulation software adopted by the neutron community and to make publicly available the instrument description. The simulation experts' feedback and contribution can play a major role in the improvement of the accuracy of the simulations, while keeping orthogonal the development and maintenance of the DT implementation.

The DT of an instrument would be able to serve for several purposes: training new ILL users to the ICS or current users to a particular instrument configuration settings and capabilities; studying and optimizing of instrument settings for specific figure of merit; improving analyses with better understanding of some background sources and uncertainties; or enriching proposals for demanding beam time with results from simulated data with the specific instrument.

DIGITAL TWIN IMPLEMENTATION

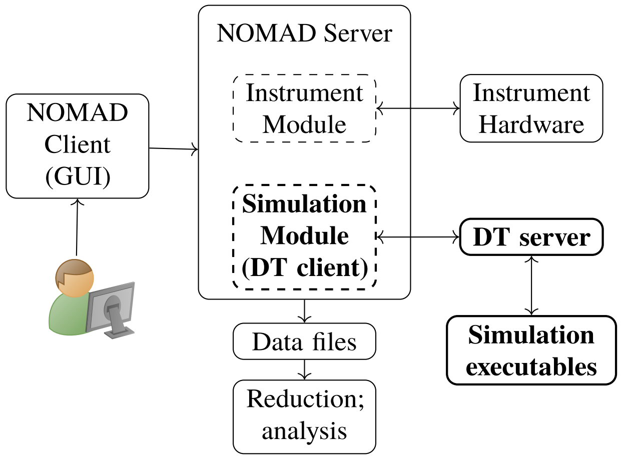

Figure 1: Overview of the logical elements and their communication links. Highlighted in bold are the DT specific parts.

The DT is composed by three logical components: a simulation executable provided by the chosen simulation software, a client application programming interface (API) used by the ICS (acting as a DT client) and a central unit linking those two parts (DT server). An overview of the logical elements and communication links in the ILL's architecture is shown in Fig. 1. NOMAD [1] is the instrument control software developed and used at ILL for the data acquisition and instrument control. Its core is represented by a C++ server that communicates with the instrument hardware with specific modules. The interaction with the user happens via NOMAD client (JAVA application). The integration of the DT into NOMAD is achieved adding a module to the ICS to communicate with the DT server and replacing the com-

* nourbakhsh@ill.fr

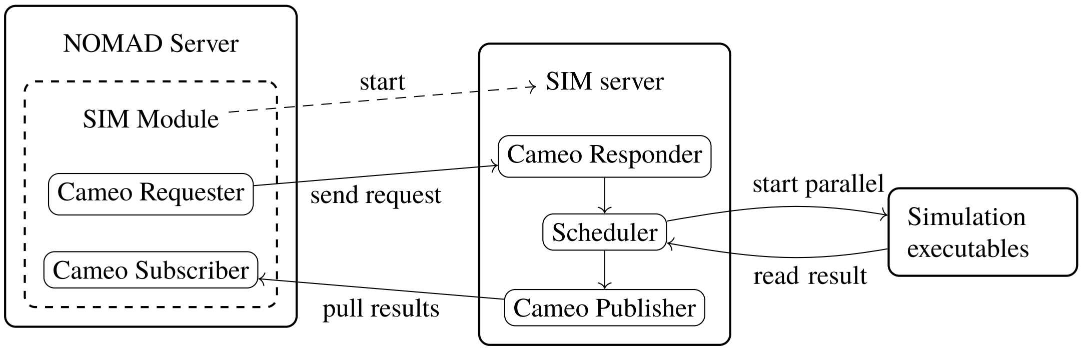

Figure 2: Schematic of the communications.

munication with the instrument hardware. The DT server is seen by the ICS as the instrument hardware would be. As such, the simulated data returned by the DT are then treated by the ICS as the real data.

Simulation Software and Instrument Description

At the time of writing the simulation software used is McStas [2–6], but the architecture allows the expansion with different simulation packages. The instrument has to be described according to chosen simulation software. McStas requires its own meta-language, that then it transforms into a C program that is further compiled as single executable. It generates single neutrons with their properties (position, velocity, polarization, travel time) from the source and traces them to the final detector through the different components undergoing absorptions or modifications caused by reflections, polarizations, scatterings. When reached, detector components record in text files the neutron counts and properties. The process is repeated for each neutron. Instrument components (sources, optical elements, samples, detectors) are configurable. When describing an instrument, component parameters can be set as function of command-line arguments of the executables.

The very first instrument description requires a close collaboration between a McStas expert and the instrument responsible at the institute. This collaboration provides both the deep insights of the specific instrument and the expertise in the way to simulate it. It is published in the “instrument database” github repository[1] set up by the PaNOSC [7] WP5. The repository and its python API allow the storage of multiple versions of the same instrument, representing real instrument configurations at different periods in time. The version control of the description is provided by Git [8] and github allows the setup of continuous integration tests, issue tracking and public contribution from the community of simulation experts that will have access to the most up-to-date version of the description used by the DT. The description of multiple instruments is also simplified by using the McStasscript [9] python library instead of the McStas meta-language. One immediate advantage of McStasscript is the possibility to run the simulation in a jupyter notebook, allowing a larger user base to potentially run simulation not only from their machines but also from the cloud. When compiling the DT components, the latest version of the instrument descriptions are retrieved from the public repository, compiled and the executables are packaged, ready to be installed and executed.

The Client

The client API has been designed to standardize the request and result messages sent to and received from the simulation server. On the ICS side, the instrument parameters required by the simulation are known to the API and hence a simulation request is enforced to provide all the necessary information. The message is then formed by the API ready to be sent to the server. Messages from the DT server are decoded by the API providing a standard interface to obtain the status of the request, the status of the simulation and the data. The implementation of the communication is left to the client using the Cameo [10] API as described in more details in the following.

The DT Server

The simulation server is the core of the DT. It waits for simulation requests, queues them, it manages the simulation execution and returns the results once available. The simulation server acts as a smart simulation scheduler, re-using previous simulation results as much as possible. An instrument configuration file, loaded by the DT server, contains the information about the matching between the ICS parameter names and the simulation executable inputs, making possible the replacement of McStas with other simulation softwares. The DT server can be configured to be started by the ICS or can run as a service. In either way, multiple ICSs can connect to a single DT server and place a simulation request. It is possible to configure the DT to queue them, or run in parallel. For each request, a dedicated scheduler is created in a separate thread, managing, in a totally independent way, each simulation request.

Communication and Interactions

The DT server is started by the ICS when the user activates the simulation module. The communication between

[1] `https://github.com/PaNOSC-ViNYL/instrument_database`

ICS and server is performed using two different communication patterns: requester-responder for the simulation request, publisher-subscriber for the simulation results. Both communication and application management are provided by the Cameo middleware. It consists in a JAVA server, managing a set of applications. Processes can be started by the user from its command-line interface or by other applications via its JAVA, C++ or python APIs. Multiple servers can run on different machines and interact with each other allowing management and communications between remote machines. Cameo is based on ZeroMQ [11] and provides among other things a high level C++ classes simplifying the implementation of the requester-responder and publisher-subscriber communication patterns between applications. The communication process between the ICS and the DT is shown in Fig. 2. In the DT design, the ICS server, the DT server and the simulation executables can run on different machines, increasing significantly the deployment flexibility by using bare metal or virtual machines or any combination of the two. The process is started by the ICS that composes the simulation request using the DT client API. The ICS use a requester that sends the request and receive the acknowledge message from the server. If there is a positive answer about the simulation starting, a subscriber is put in waiting for results from the publisher on the server side. The DT server publishes not only the final data, but also the status of the ongoing simulation and data with partial statistics. The user is then able to see the simulated data with improving statistics and decide to eventually stop the simulation if satisfied or if a mistake in the configuration is detected.

Smart Simulations

The DT's user experience would be a success only if the virtual experiment acquisition time is comparable or even faster than the real one. The sample's scattering cross section has no impact on the simulation execution time, thanks to the design of McStas, making it very interesting for the user to use the DT for experiments that might require very long acquisition times. A limiting factor is represented instead by the number of optical elements between the source and the detector, affecting the number of effective neutrons reaching the detector, reduced even by several order of magnitudes. A strategy for parallel processing and for reducing the re-simulation of lost neutrons has been put in place.

Parallelization Neutrons in a ray tracing simulations are completely stochastically independent objects. A given number of neutrons can be simulated as multiple execution of the same simulation, with appropriate initialization of the seed of the random number generator. As such, no special library is needed for higher parallelization of the simulation with the execption of correctly merging the results. The scheduler plays a central role in dispatching multiple execution of the simulation and merging the results. The simulation process is split into jobs with fixed number of neutrons and known execution time. They are queued for parallel processing based on the available resources in order to obtain larger statistics. As soon as new jobs with neutrons reaching the detector are finished, the results are merged and sent by the publisher as an update. This mechanism allows the user to obtain incrementally more detailed results as the simulation proceeds. To ensure the re-use of previously simulated jobs, a pre-defined sequence of seeds is used ensuring the full reproducibility of the data in subsequent simulations and the stochastically independence of different jobs.

Staging When describing an instrument, particular care has been put in splitting an instrument in different segments. The segment executables are run one after the other in order to get the full path of a neutron from the source to the detector. Neutron states are saved at the end of each segment in an MCPL [12, 13] file and then read back from the next segment. This way, the simulation is split into multiple stages, each simulating one segment of the instrument with the associated instrument parameters. The scheduler is aware of the number of stages, the name of the corresponding executables, and the list of parameters affecting the specific stage. This information is provided by a JavaScript Object Notation (JSON) file read by the DT server at startup. The staging allows the re-use of previously simulated stages whenever the relevant instrument settings are the same, mitigating the issue of long beam lines. For instruments where the experiment requires scanning of one or a set of parameters, it is possible that only few settings of the optical elements at the end of the beam line are changed. This strategy becomes extremely effective in these cases, reducing the simulation time of the second and following scan points by order of magnitudes.

CONCLUSION

In this article a Digital Twin that allows to run virtual experiments with ILL's instruments has been presented. It is a tool that does not require any knowledge about neutron physics nor simulation. The modular design allows the integration in any instrument control system, the adoption of different simulation software and the execution on remote machines of the different components. The user can start using the tool almost immediately if they are already familiar with the instrument control system that is their single entry point. Sustainability of the project is guaranteed by the involvement of the instrument scientists in mantaining an up-to-date and accurate description of the instrument and the wider simulation expert community in testing, debugging, improving the simulation description accuracy and performance. The release of the instrument description in a central and public repository plays a crucial role in this respect. Ideas about parallelization (specific for ray tracing simulations) and instrument simulation sectioning have also be described and could be adopted also in other contests for Digital Twins for photon sources.

REFERENCES

[1] P. Mutti *et al.*, “Nomad – more than a simple sequencer”, in *Proc. ICALEPCS’11*, Grenoble, France, Oct. 2011, pp. 808–811.

[2] K. Lefmann and K. Nielsen, “McStas, a general software package for neutron ray-tracing simulations”, *Neutron News*, vol. 10, no. 3, pp. 20–23, 1999. doi:10.1080/10448639908233684

[3] P. Willendrup, E. Farhi, and K. Lefmann, “Mcstas 1.7 - a new version of the flexible monte carlo neutron scattering package”, *Phys. B: Condens. Matter*, vol. 350, no. 1, Supplement, pp. E735–E737, 2004. doi:10.1016/j.physb.2004.03.193

[4] P. Willendrup, E. Farhi, E. Knudsen, U. Filges, and K. Lefmann, “McStas: Past, present and future”, *J. Neutron Res.*, vol. 17, pp. 35–43, 2014. doi:10.3233/JNR-130004

[5] P. K. Willendrup and K. Lefmann, “McStas (i): Introduction, use, and basic principles for ray-tracing simulations”, *J. Neutron Res.*, vol. 22, pp. 1–16, 2020. doi:10.3233/JNR-190108

[6] P. K. Willendrup and K. Lefmann, “McStas (ii): An overview of components, their use, and advice for user contributions”, *J. Neutron Res.*, vol. 23, pp. 7–27, 2021. doi:10.3233/JNR-200186

[7] ,https://www.panosc.eu/

[8] Git, https://git-scm.com/

[9] McStasScript, doi:10.5281/zenodo.6560751

[10] Cameo, https://code.ill.fr/cameo/cameo

[11] ZeroMQ, http://www.zeromq.org/

[12] T. Kittelmann, E. Klinkby, E. Knudsen, P. Willendrup, X. Cai, and K. Kanaki, “Monte Carlo Particle Lists: MCPL”, *Comput. Phys. Commun.*, vol. 218, pp. 17–42, 2017. doi:10.1016/j.cpc.2017.04.012

[13] MCPL, https://mctools.github.io/mcpl/