Towards a systematic requirement-based approach to build a neutronics study platform

Alberto Previti,[*,a,b] Alberto Brighenti,[a] Damien Raynaud,[a] and Barbara Vezzoni[a]

[a] *Framatome - Codes and Methods Department*
1 place Jean Millier, 92084 Paris-la-Défense cedex, France

[b] *ENEA Centro Ricerche Frascati*
via Enrico Fermi 45, 00044 Frascati (RM), Italy

[*]Email: alberto.previti@enea.it

Number of pages: 50
Number of tables: 0
Number of figures: 12

Alberto Previti https://orcid.org/0000-0003-3479-8981
Alberto Brighenti https://orcid.org/0000-0003-4775-9375
Damien Raynaud
Barbara Vezzoni https://orcid.org/0000-0003-3934-3505

Abstract

The design and safety assessment of nuclear reactors relies on a combination of calculations performed by several simulation packages, each dedicated to modeling a specific ensemble of phenomena. To treat the complexity of the physical problem, appropriate software architectures and methodologies to trace and implement user needs are of paramount importance to fulfill the needs of all the possible stakeholders. This work proposes a systematic approach to break the complexity of constructing a lattice neutronics platform, that is one of simulation package needed in nuclear reactor analysis. After reviewing the state-of-the-art of current methods applied in reactor physics engineering, the work concentrates on identifying the applicable software architecture strategies and on discussing advantages and drawbacks. While the specific target is the neutronics code APOLLO3®, the subsequent categorization and analysis of user needs written in the form of formal requirements allows for defining a unified approach to design an effective, industrial-grade, and future-proof calculation platform. Subsequent presentation of typical use cases involved in developing deterministic lattice calculation schemes allows linking the formal definition of use cases and software architecture with the actual application to a specific calculation setting. This work aims, therefore, at proposing an innovative viewpoint to tackle large software developments applicable in the nuclear industry. The research presented in this paper has been developed at Framatome in the context of the lattice neutronics work package of the H2020 CAMIVVER project.

Keywords — Neutronics calculation platform, Software architecture, Systems engineering, Industrial use cases

I. INTRODUCTION

Designing innovative nuclear reactor concepts, periodic safety qualifications, and efficient operation of existing power plants are nowadays increasingly rooted in complex multi-physics simulations. Coupled core calculations are meant to simulate the reactor behavior under all operating and accidental conditions to provide accurate solutions capable of assuring that the safety criteria are respected under design basis accidental conditions and in general for all possible scenarios. Core neutronics, heat conduction in fuel rods, and thermal-hydraulic of the cooling system are the three main disciplines that dominate the physical phenomena in a nuclear fission reactor. Since they show their effects in different spatial and temporal scales, they are often managed by different simulation codes. Indeed, since the neutronic behavior of the reactor core is strongly dependent on the feedback received from the interconnected disciplines, input and output fields need to be exchanged via well-defined interfaces.

Given the number of different disciplines and tools involved, the ensemble of simulation codes and associated methodologies employed in the analysis of a nuclear reactor is often called *calculation chain* to underline the relationship between the interconnected nature of the physical phenomena with the resulting exchanges needed at the software level. Building an effective calculation platform is a great challenge from both the technical and organizational points of view because of the roles and relationships among multiple competencies. This is already the case for a single component of the calculation platform, as shown in the paper for the lattice part.

Nowadays, industrial applications require using these software stacks to perform large numbers of reactor calculations. These simulations provide, on the one side, the means to justify the fulfillment of the strict safety standards demanded by the regulatory agencies for the design and certification of new and existing reactors and, on the other side, the tools to operate current reactors efficiently. In addition, the follow-up of network requests and considerations based on the actual operational history sometimes involves the redefinition of cycle strategies, resulting in very short response times requested by the users (one night).[1] The optimization of these expensive calculations while preserving the accuracy of results is, therefore, of paramount importance in the nuclear industry.

More specifically, reactor neutronics calculations are typically performed since the 80s employing numerical solvers based on discretized representations of the diffusion and/or simplified

transport operators on 3D geometries.[2]

To respect the need for a reasonable response time, reactor core calculations are performed on coarse meshes, relying on physical data previously computed at the assembly level.[3] Lattice neutronics calculations are therefore meant to provide refined multigroup flux solutions at the fuel assembly level which are used to prepare multi-parameter libraries of few-group cross sections and pin-power form factors that enable full-core neutronics calculations.[4] This approach, commonly known as *two-step calculation*, is a *de facto* standard in industrial applications.

This paper concentrates on the lattice calculation part to propose a more generic development approach. When focusing on lattice calculations, each cross section model is tailored to the fuel type in the reactor under study and the corresponding core solver employed. In fact, the few-group cross sections are used to reproduce the feedback of the state parameters that primarily affect neutronic reactivity, for instance: fuel temperature, moderator density, soluble poison concentration, and isotope concentrations. More specifically, isotope concentration modifications during the reactor cycle are accounted for via depletion calculations at imposed nominal or perturbed conditions, while the variation of the other state parameters is considered in branch calculations for each imposed burnup point. At the same time, real reactor conditions may differ from pre-calculated state points: efficient storage strategies and interpolation functions must accurately reproduce cross sections and pin-power form factors for the actual core conditions. These methodologies may range from the simple multi-linear interpolation, based on hyper-cubes containing the relevant quantities of interest for the full Cartesian products of the state parameters, to compressed and more efficient storage strategies.[5, 6] Consequently, multi-parameter neutron data libraries preparation should be cast via software architectures flexible enough to accommodate several output formats.

When looking at the actual calculations employed in the generation of assembly data, traditionally two-dimensional simulations are employed; three-dimensional methodologies still need to mature before being customarily applied in the industrial sector.[1] Also, one may note that lattice code protocols typically rely on obtaining reference flux thanks to self-shielding procedures and accurate flux solvers. Several calculation protocols, i.e., the ensemble of recipes to drive the self-shielding and flux solvers to determine the final transport solution, may be needed, ranging from *best estimate* schemes devoted to providing very accurate results for a wide range of potential reactor and fuel types to *industrial* schemes tailored to specific applications optimized for

faster response times. Then, a final spatial and energy homogenization allows for the generation of few-group cross sections with coarse meshes.[7, 8] Geometrical output regions indeed depend on the actual capabilities of the core solver, i.e., whole fuel assembly or quarters, for squared LWR fuel assemblies, and hexagonal sectors, for VVER and fast reactors. More accurate computational protocols may require data homogenized for every single pin-cell (cell by cell), and for a number of energy groups larger than 2, albeit at an increased cost of storage.

Assembly calculations are often done using an infinite lattice model, where reactor conditions are imposed via critical leakage, e.g., via the classical homogeneous B1 or via more sophisticated models.[9] New approaches are, however, emerging, for instance, the dynamic homogenization technique that accounts for the actual localization of the fuel assemblies in the reactor core by an iterative solution comprising successive exchanges of boundary currents.[10] It means that flexible exchanges of data fields via open interfaces at the software level will represent, in the foreseeable future, a key factor in the success of innovative and flexible calculation chains.

Finally, in the context of the *two-step* approach, a typical simulation usually takes into account calculations of combinations of several assemblies, often called *color-sets*, and specialized 1D and 2D motifs aimed at representing the radial and axial reflector geometries. As with the classical single-fuel assembly setups, these geometrical patterns are often needed to prepare nuclear data for core simulations. In addition to multi-parameter neutron data library preparation, lattice codes are routinely employed for specific studies, such as designing new assemblies or validation activities connected with experiment modeling. In these cases, the engineers need accurate solutions with reduced methodology biases to optimize their design or to tune protocols employed in batch calculations. Consequently, the development of advanced methods is strictly linked with the capability of current and future codes to support potential non-standard workflows.

Previous generation calculation chains, like SCIENCE v2 currently used at Framatome,[11] rely on software processing comprising either the coupling of its codes via input/output files or dedicated on-purpose sockets. Indeed, recent trends in the nuclear industry push towards a more advanced coupling between physics, allowing to perform studies in which the analyst is able to interrogate and modify the state of the calculation code kernel. In fact, open interfaces allow the dynamic composition of the calculation processing while maintaining strict respect for the single responsibility principle among simulation tools. As a plus, substituting some bricks of the chain

with different codes and/or options eases the comparison among solver kernels and the available simulation strategies. A modern architecture comprising different software layers dedicated to the underlying numerical methods and the engineering objects has already been implemented in COCAGNE,[12] the neutronic core code under development by EDF and Framatome. This modern approach showed a significant gain in the flexibility of the whole calculation toolbox while preserving high computational efficiency thanks to the careful definition of the software implementation methodology for each layer.

In the H2020 CAMIVVER project,[13] a task is dedicated to defining and implementing a multi-parameter neutronic library generator based on the deterministic neutronic code APOLLO3®[14] and its related meshing software ALAMOS[15]. APOLLO3® lattice part may be considered as the successor of the APOLLO2 code[16], presently used as a well-established standard in the industrial calculation chains used by Framatome and EDF. The underlying idea of the lattice neutronics work package in CAMIVVER is to analyze the possibility of advancing the industrialization of the APOLLO3 deterministic calculation kernel, so that the advances in modeling, numerical methods,[17] and software architecture of the last decades can be leveraged to provide state of the art results for VVER and PWR applications.

The paper aims to present the work performed at Framatome (France) in the context of the lattice neutronics work package of the H2020 CAMIVVER project, paying special attention to explain the full process by presenting the method, the implications, and the technical impacts.

As briefly outlined, the interconnected physics of a nuclear reactor and the need for more powerful simulation toolboxes capable of managing various scenarios in a comprehensive and well-organized manner force to strike a balance between very specific codes, which are tailored and optimized for a given application and target machine, and more versatile software stacks, which are potentially slower. A systematic way to tackle this complexity to build efficient and flexible calculation platforms is the primary subject of this work. Therefore, this work presents a systematic approach based on the systems engineering paradigm, which has been customized and designed specifically for lattice neutronics applications. We will show how a structured requirement-based approach can better identify the user and the developers' views, allowing for collaboration and exchange to overcome the limitations of previous-generation simulation codes. Following these principles, especially the idea of having the stakeholders as the central points of the definition of

the simulation toolbox, the analysis starts in Section II with the identification of the target users, discussing both the actual people involved and the target computational platforms. Indeed, production releases of these simulation toolboxes may be employed in industry for decades, because of the considerable amount of work for development, verification and validation, and certification. It is, therefore, critical to clearly identify as much as possible current and future potential needs through an open exchange with all the different actors. The analysis continues in Section III with the decomposition of the main software components, followed by the determination of the principal skills of the developers and a clear separation of responsibilities. The software architecture strategies are discussed as well, comparing advantages and drawbacks. At this point, Section IV proposes a Requirements Management Plan showing the interconnection among actors and components and a systematic way to define the overall calculation platform, either from the numerical and organizational point of view. The categories analyzed for use cases applicable in lattice neutronics are detailed in Section V, linking the consideration drawn so far with the actual scenarios of interest. Section VI proposes a case study with the deterministic codes APOLLO2[16] and APOLLO3®[14] and the Monte Carlo code TRIPOLI-4®[18] to show the actual connection of the procedure outlined with the classic advanced workflows applicable in the industrial sector. Finally, conclusions follow in Section VII.

II. SYSTEMS ENGINEERING APPROACH TO LATTICE NEUTRONICS

There is a multitude of neutronic codes, either deterministic or stochastic: they are capable of solving the transport problem coupled with the evolution of isotope concentrations. These solvers are the results of intense research spanning several decades spent to address the complexity of the transport phenomena in nuclear reactors.[1, 19] In addition, being able to solve the problem open the doors to the careful analysis of the stability of the algorithms and their capability to provide accurate solutions.[20] This quite intensive activity is focused mostly on the mathematical and numerical aspects of the transport problem: the prerogative of the solvers may thus be considered together with the accompanying pool of competence as a founding building block that needs to be established as a prerequisite.

At the same time, as seen in Section I, industrial and research needs require to go beyond the concept of neutronic code to provide a neutronic platform. Systems engineering is presented here

as the ideal framework to build, in a more effective way, a neutronics platform, because it involves all the activities related to the management of complex systems over their life cycle. The main objective of systems engineering is to deliver a product that fits customer needs. During the production of a neutronics lattice calculation platform, an effective collaboration among stakeholders and disciplines considering multiple requirements and interfaces is mandatory. The first step is, therefore, to recognize the actors that should identify the user scenarios:

1. *nuclear plant managers*: they are mostly confronted with fuel loading pattern determination for reactor cycle optimization and routine calculations to define operating conditions according to the request of the transmission control operator of the electric grid;

2. *safety analysts*: they employ the calculation toolboxes to assess the safety of the reactors under design or during periodic assessment; their jobs involve the simulation of all possible operational and accidental scenarios to guarantee the respect of the safety criteria in all cases;

3. *nuclear fuel and core designers*: they are devoted to the design of fuel assemblies and reactor geometries and compositions for new and more efficient nuclear fission reactors;

4. *research agencies*: they are confronted with calculations involving well-established reactor technologies as well as new experimental mock-ups with completely unexplored geometrical and material features;

5. *training specialists*: they are mainly involved in the preparation of courses to teach students and employees how to analyze nuclear reactors;

6. *students*: they are confronted with the physics of a reactor and its analysis;

7. *algorithm researchers*: they conceive the numerical deterministic and/or Monte Carlo methodologies to tackle the neutron transport problem and the depletion of isotopes under neutron irradiation in the geometries of interest;

8. *calculation scheme and verification and validation experts*: they assemble the numerical solvers of the basic equations governing the physics of nuclear reactors in calculation protocols tailored to the given type of reactor and perform verification and validation of these methodologies by using comparisons with different numerical solution strategies and experimental data available;

9. *codes and methods developers*: their main responsibilities concern the optimization of the numerical algorithms for the actual applications of interest and the development of calculation ready-to-use software.

Following systems engineering practices, this classification is based on the actual roles of the actors. Indeed, the platform may be subject also to the guidelines of the safety authorities, e.g., [21]. All those actors gravitate around the calculation platform and share only partially their needs. At first, it is necessary to stress the fact that the above-mentioned actors can be classified into two main groups: those who actually focus on the physics of the nuclear reactor machine (1, 2, 3, 4, 5, 6), and those who concentrate on the numerical algorithms from the point of view of their software implementation or their use via specific calculation protocols (7, 8, 9). We will employ this consideration in the definition of the software architecture applicable in a neutronics calculation platform. Of course, the same person may correspond to different actors at different times depending on the specific activity. For instance, a *codes and methods developer* himself is an end-user of the simulation codes he produces. Nevertheless, in general, a single person may not represent all the possible actors. The first key to succeed in the development of a lattice neutronics calculation platform is, therefore, to take into account that quite different competencies and centers of interest exist: each of them may have its own requirements and constraints. Indeed, building the calculation platform requires an organized methodology to let all the actors share equal rights and responsibilities over all the software life cycle.

Having understood *who* interacts with the calculation platform, the analysis continues to *where* it is used. Indeed, a simulation toolbox is just a collection of software, and following the considerations made so far, it may serve different purposes. This translates to various potential target computing machines. Two sub-concepts are thus needed:

1. *distribution*: it refers to several distinct calculations related to different input parameters executed at the same time;

2. *parallelism*: it refers to a given single study that requires the execution on multiple processors and/or computing nodes because of the number of degrees of freedom (memory-bound) and/or expected number of computing operations (CPU-bound) needed to complete the algorithm with the required accuracy in an acceptable time.

For instance, a research agency may need to support large and complex simulations of complicated geometries in custom experimental setups, while an ordinary safety analyst may need to perform many routine simulations changing some relevant parameters. Many intermediate scenarios exist as well. For this reason, the target computing machines applicable to a neutronics calculation platform may range from ordinary workstations to high-performance computing (HPC) clusters. Indeed, the software toolchains, the processor families, the storage technologies, and the network interconnections change during the platform's lifetime and among the supported target computing machines. To assure conformity for all the potential applications, it is necessary to trace the corresponding non-functional system requirements in strict collaboration with *information technology specialists*, who are additional actors involved in the development.

In CAMIVVER lattice neutronics activities, four main pillars have been considered:

1. *flexibility of modeling and analysis options*: i.e., the possibility for the user to employ the calculation platform for a broad set of applications ranging from specific design and verification and validation studies to routine generation of neutronic data libraries for core codes;

2. *consistency of requirements specification*: i.e., the clear identification and fulfillment of the user expectations in terms of modeling options and interface of the calculation platform;

3. *innovative algorithm and improved precision*: i.e., the need to move a step forward in the accuracy and performances with respect to the calculation platform currently employed in the industry;

4. *state-of-the art platform and architecture*: i.e., the goal to achieve a modern platform capable of implementing the required functionalities and attain the expected performances with the possibility of building on an open and extendable platform for future needs.

The classical starting point to systematically breakdown project activities also applied to software development is the so-called Waterfall model.[22] Under this approach, the activities are decomposed in a series of linear sequential phases: the first step is the generation of requirements of the system. The requirements analysis is the heart of the design phase, in which most of the technical choices are made. Afterward, the development phase takes place until the code is ready for testing and then released to production.

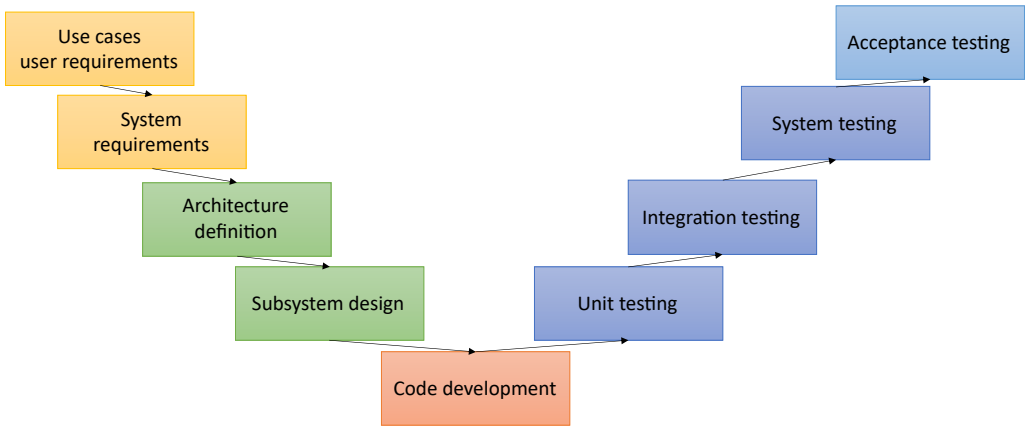


Fig. 1. V-cycle phases

It should be stressed that a requirement is supposed to be a clear and concise statement to define a singular unit of the demand for the system.[23] Requirements specification is thus the basic initial activity to identify the system goals to fulfill user needs clearly, and thus a starting point to feed the design phase. More specifically, the coding phase cannot be pursued before having clearly identified the requirements and performed the basic design following them.

One of the main drawbacks of the Waterfall model is its linear sequential approach and a somewhat missing direct link between the project's initial and final phases. This is the reason why the more advanced V-model[24] is preferred, since it enhances the relationships between each step of the engineering life cycle and its associated phase of testing, thus reinforcing the fulfillment of the requirements at each level. Figure 1 depicts the main phases of this approach: at first, the teams go down through the left side of the V by starting with the identification of the needs and their translation to corresponding requirements (yellow boxes) and by completing the design according to them (green boxes) first at a higher level and then for each software component. Then the solution is put in place during the development (orange box). Once the code has been completed, the team proceeds upwards in the right side of the V, dedicated to progressively testing the code produced against the requirements, starting from unit testing for each corresponding abstraction layer (blue boxes) to broader acceptance tests (azure box). It should be noted that the V-model, according to the above-mentioned description, would require a complete identification of all use cases and system requirements before any development. While this would be the ideal case, in real-world scenarios code development and testing often reveal the need to update use cases and

system requirements, which indeed will propagate to design adjustments and code modifications. As such, an iterated approach is typically employed between the two main branches of the V-model. Section IV will present the use of traceability matrices to support these iterations of the V-cycle, which may occur multiple times until attaining consistency.

III. ELEMENTS OF SOFTWARE ARCHITECTURE FOR LATTICE CALCULATIONS

Before decomposing the specification phase on its interconnected levels, we need to analyze the applicable software architecture approaches. Calculation codes are often *monolithic*: the underlying solver and its data model are tightly coupled with the user frontend. Under this approach, the computational kernel allocates the memory needed for processing in its memory space after parsing an input deck or after receiving the relevant information through a specific application interface. The execution of the simulation proceeds according to the inner mechanics of the code, which may create intermediate software structures opaques to the user to carry out the instructions included in the input deck or received via dedicated interface calls. In any case, the simulation is based on the internal state of the computational kernel, which has been initialized based on the initial settings and is modified internally during the course of the numerical algorithms. The outcome is generally dumped in the standard output and/or in output files requested by the user, as depicted in Figure 2(a). This methodology has several drawbacks since the analyst has limited capability of performing online branching decisions and/or cannot reuse some memory structures to accomplish additional calculations. Moreover, the debugging or the elaboration of complex yet efficient calculation schemes may be severely limited without direct modification of the source code, which may or may not be accessible.

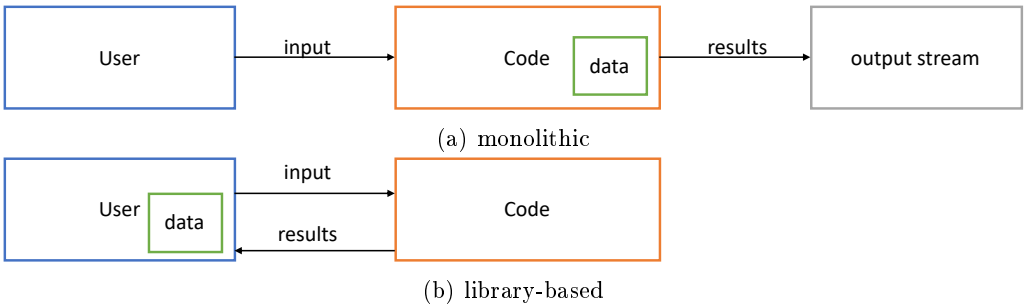
To better understand this point, we may consider the case of a user who defines a flux problem. According to this paradigm, the computational code stores the modeling and data structures internally after reading the input deck. At this point, the user launches a flux calculation, and the code performs the calculation and shows outputs in the listing. If there is no convergence, the user needs to manually analyze the listing, stop the execution, and relaunch everything from scratch after modifying the input deck. This is not only inconvenient because of manual processing, which may or may not be managed by external wrappers, but especially for the waste of the

initialization that shall be run again and again. In addition, the same user cannot employ the same memory structure for a similar yet different problem: this is the case of parametric studies in which vast parts of the setup are identical with the exception of a few specific options. Since the user is not the owner of its data, multiple runs require multiple instances of the same computational kernel, in order to be sure that they work in different memory spaces guaranteed by the underlying operating system. This approach imposes additional limitations in the framework of large calculation chains, in which the same dynamic shared library of the computational kernel may be loaded in different software modules, leading to potential conflicts because of unexpected internal state modifications triggered in seemingly separate source code sections.

Another approach involves the construction of the computational kernel as an ensemble of modular pure functions, i.e., software components performing their operations without side effects and without having to modify the internal state of the library to which they are belonging. The advantage is depicted graphically in Figure 2(b): since the simulation data belongs in this case to the user, it is way more simple to introspect the processing workflow and to perform branching decisions via an external scripting language. For example, following the analysis of some intermediate state of the neutron transport solver, the user may program the calculation scheme to adapt itself with different options and/or convergence accelerations, without being forced to directly modify the implementation of the underlining code. At the same time, the construction of outputs may go beyond the rigid predefined file formats, since it can be tailored to the specific applications and requested post-processing. As a plus, a potential orchestration of multiple multi-physics calculations may profit from a given computational kernel in completely different time-frames of the processing, being sure not to have side effects.

In this context, this library-based approach supports the definition of calculation workflows at the user level via interpreted scripting languages, e.g. Python, Perl, etc. In fact, if the software components in the simulation code expose their functionalities via well-defined interfaces, it is possible to easily construct bindings to interpreted languages, thus leveraging the modular approach to obtain a completely flexible platform. The same idea implemented in monolithic codes would instead allow exclusively to manage triggers that modify the internal state, that remain opaque to the interpreted script and therefore less compatible with custom workflows.

If we consider the very same example of a user who defines a flux problem, according to



(a) monolithic

(b) library-based

Fig. 2. Software architecture concepts: objects property and life cycle

this paradigm, the computational code returns the modeling and data objects to the user. At this point, the user launches a flux calculation and the code manages input objects, performs the calculation, and returns outcome objects. In this case, if there is no convergence, the user may analyze the results and perform adjustments dynamically in a programmable way. In fact, since the data is on the user side, the user may call the same solver routine again with the input data having different options, or potentially modify the input data employing the unconverged flux as the first guess for the solver. This means that the major limitation is the experience of the analyst and not the capabilities already coded in the computational kernel. This paradigm allows, therefore, naturally interactive processing and by-design dynamic branching decision capabilities. While this advantage may be at first sight considered limited to very specific applications, it should be stressed that having the state at the user level allows employing the same data objects at the same time for different calculations without needing to regenerate everything from initialization. The support of parametric studies, to be performed potentially in parallel, is no longer managed by the process separation algorithm of the underlying operating system, but directly by the computational kernel itself.

Indeed, large calculation chains may freely employ the same dynamic shared library in different source sections without the risk of side effects. This is quite an important feature, since complex simulation toolboxes may be developed by completely different teams: the responsibility of each software stack is clearly separated realizing complex workflows in a much more flexible way.

When performing lattice calculations, having the possibility to fine-tune the data-flow options, numerical techniques, and input databases is of the utmost importance.[25] A disruptive innovation was made available in 1987 with the release of APOLLO2,[26] that provides an internal scripting language GIBIANE that allows the analyst to program the calculation backend. In

this way, the calculation kernel is fully modular and the advanced user may design appropriate calculation settings for many reactor types. However, the GIBIANE language is a very low-level API, too difficult for routine studies, that motivates upper-level input generators, as is done in APOLLO2-A[27].

The main design choices in APOLLO2-A come from the consideration that generating multi-parameter cross section libraries comes as a loop of single atomic calculations, for which the end-user normally has little interest in the inner mechanics of the solution strategy but is instead concentrated on the physical properties of the fuel assemblies. This is the reason why a larger effort has been put in place to define the APOLLO2-A input deck as an easy-to-understand syntax. It is based on a user-friendly keyword-based document, where keywords are self-explanatory and default options are provided where possible. Keywords are grouped in blocks, coupled hierarchically via their identifiers, describing various physical objects (for example, a rod, an assembly, a detector, etc.). At the same time, calculation options are defined in another keyword-based document so that the code utilization is decoupled from the definition of inner numerical methods. To perform the input processing in APOLLO2-A, a brand new C++ frontend has been built to drive the APOLLO2 kernel written in Fortran 77/90. The APOLLO2-A frontend is capable of analyzing the simplified user input already described and producing the corresponding GIBIANE calls to drive the APOLLO2 calculation.

After considering this experience, we thus may identify the two main software categories in the lattice calculation platform:

- *backend*: collects the numerical solvers and provides accessible interfaces to conceive calculation schemes;
- *frontend*: aims at providing the analyst the engineering view to study the nuclear reactors by profiting from the backend for the mathematical and numerical evaluations.

We may note that the two groups of actors identified in Section II naturally interact with either the frontend only or with both frontend and backend.

Considering this point, we note that user-friendliness in APOLLO2-A is managed by the internal handling of most of the complexity: while the APOLLO2 backend maintains basically a modular stateless library approach, the APOLLO2-A frontend architecture features a monolithic structure. The goal of the industrialization of the next generation lattice codes, such as

APOLLO3®, is thus to develop interactive high-level user experience, thanks to fully modular and flexible backends and frontends. Achieving this goal may require partially rethinking the existing backends, and reorganizing their internal processing to obtain a fully modular architecture free of internal states.

IV. REQUIREMENTS MANAGEMENT PLAN: THE SPECIFICATION TREE

Following the analysis of the previous sections, the specification tree applicable to lattice calculations may be articulated in three main levels, as shown in Figure 3:

1. at the *highest level*, there are the use cases for the calculation platform (azure boxes), alongside the applicable reactor design options and core code needs (yellow boxes) and calculation methodologies (gray box);

2. at the *system level*, there are the requirements for the calculation platform (green box), which satisfy user needs;

3. at *sub-system level*, there are the lower-level requirements that satisfy the system requirements: from one side, the requirements for the calculation backend (blue box, i.e., the code system providing the basic computation primitives) and on the other side, the calculation frontend (orange box, i.e., the software with which the end user interacts), and the input/output interface of the calculation backend and the output multi-parameter library interface (i.e., the format used to communicate with the core codes) (white boxes).

While these three levels complete the specification tree, the activity continues with the design and development: the dark blue and orange cases in the lower part of the picture relate to the specific requirements of the components of the actual platform, implementing the functionalities of the backend and frontend, respectively.

This specification tree has been conceived to reinforce the collaboration among the actors and enhance the system's conformity with respect to user needs. The orange arrows in the figure depict the relationship between *children* and *parent* requirements, represented in the following as the link called *is satisfied by*. The green arrows represent the relationship between applicable calculation schemes, reactor operation standards, state-of-the-art calculation methodologies, and the platform requirements: this additional information *justifies* the generation of further system

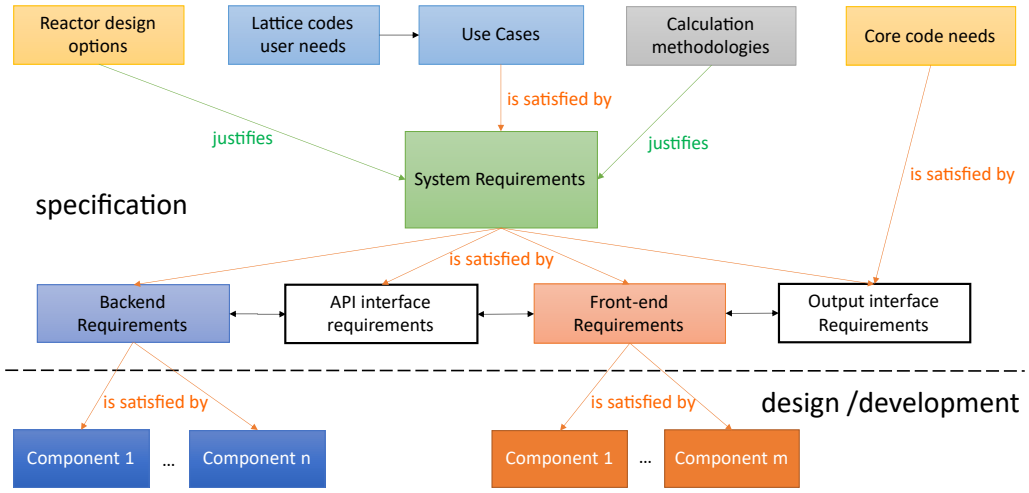


Fig. 3. Specification tree for lattice neutronics, employed in CAMIVVER

requirements (i.e., *derived requirements*). It should be noted that during the specification phase, additional sources of rationales may be identified as a justification for additional requirements at the sub-system level.

In fact, as we can see in the picture, the identification and collection of user needs and the generation of requirements for each defined level (i.e., System, Sub-System) follow both the two following procedures:

- propagation of requirements that are allocated from the upper level to the lower level of the specification tree (i.e., the *parent* requirements): these are the *basic requirements*;
- generation of new requirements arising from the design activities based on upper-level requirements and external rationales: these are the *derived requirements*.

In order to follow the specification tree, the capture of relevant information shall start from the relevant experience of all the actors involved in the calculation platform and shall be systematically traced down to the use cases and requirements specification database. Both the described top/down propagation and the requirement allocation should be monitored. In particular, the propagation to lower levels depicted with the relation *is satisfied by* between parent and children requirements allows to generate a traceability matrix, which is a tool to verify compliance. The traceability matrix shows in a compact way the links of the requirements tree, and it should be established in order to appropriately keep up to date each level of the specification, in particular

when a given feedback or consideration in a lower level triggers a modification to upper levels. In this way, they permit the following of the potential iterations of the V-model caused by the update of use cases or system requirements triggered by the later analysis during the development or test phases.

Then, the specification activity starts from considering user needs for lattice code applications. Here, the focus is on the *user* as the actor with a given role willing to obtain a result. The use cases, therefore, arise as specific statements depicting the user and its actions with the expected outcome, mostly in terms of functionalities. The collection of use cases emerges from capturing relevant information from the current industrial applications already outlined. Some common needs are shared among the main applications of the lattice neutronic platform allowing for the collection of use cases to take into account the possible synergies. A systematic breakdown of use cases and their relationships applicable to lattice neutronics applications will be analyzed in Section V.

Once the user needs are clearly stated in the form of an organic set of use cases, the specification activity may continue with its central part, i.e., the system requirements specification. Here, the focus is on the *system*: the requirements describe its capabilities and constraints. They are thus a formal specification (and design at the lower level) of the system in order to implement a specific feature or action. In this context, the system is meant, as the overall platform, to perform all the possible lattice neutronic activities. Two main categories of requirements exist:

- *functional*: related to the business logic of the functions to be performed;
- *non-functional*: related to other system properties not bound to functions, e.g., performance or quality.

Note that the direct propagation of use cases to the system results mostly in basic requirements. However, further considerations apply, and thus, the generation of additional derived requirements is possible, provided that a rationale is given as justification, according to two main sources:

- *reactor design options*, as a means to restrain the scope of the lattice calculation platform and as a source of further considerations on the applicable calculation parameters;
- *calculation methodologies*, as a means to impose the need for specific functionalities to com-

bine numerical methods to implement a desired calculation scheme, yielding as well non-functional requirements based on performance considerations.

While the central system requirements specifications and the use cases are not bound to a given calculation code, the subsequent allocation of functionalities to backend and frontend depends, to a certain extent, on the actual code system to solve the neutron transport and isotope evolution problems. In the context of this work, we support the opinion that all the actors shall cooperate in the identification of the needs, while the actual distribution of the workload depends on the peculiar competence of each team. This means that when considering, for instance, the APOLLO3® code system, the allocation to frontend and backend is driven by these considerations in view of a clear separation of responsibilities of the two sub-systems:

- capabilities already existing or subject to potential implementation in APOLLO3® and/or ALAMOS or strictly connected to the underlying numerical methods, that act as justification for the allocation and as a rationale for further requirements of the backend;
- capabilities related to the actual applications of the calculation tool, rather than internal numerical routines, that act as justification for the allocation to the frontend.

While the considerations of this analysis drawn so far may be considered as of general application, the allocation to the frontend and backend is strictly related to the specific competencies of the teams working in the construction of a given lattice calculation platform. Indeed, according to the desired software architecture, it may happen that some functionalities could appear in either the frontend, backend, or both. Although a universal rule does not exist, the systems engineering workflow provides a standardized way to solve the distribution of responsibilities jointly. Regardless of the actual decomposition and of the actual subsequent software implementation, the formal identification of use cases provides a systematic way to write acceptance tests that guarantee the fulfillment of user needs.

Alongside the sub-system requirements, the interface definitions let each part of the system communicate with the other and the external actors. The nature of a component is fundamentally different from that of an interface: while the former results in the development of a software product, the latter corresponds exclusively to a document that allows for decoupling of the responsibilities.

According to the specification tree proposed here for lattice neutronics applications, the former interface to be considered is between the backend, i.e., computational kernel, and the frontend, i.e., the industrialized entry point of the platform. The industrialized frontend shall drive the computation backend taking care of the supervision of the typical workflow applicable to attain a calculation platform respecting the system requirements. For this reason, the interaction between backend and frontend is mediated throughout the interface specification, representing the coupling between these two sub-systems. It should be stressed that this interface shall be formalized not only as the objects exchanged but, most importantly, also as the triggers and feedback of one sub-system because of the actions performed by the other sub-system. It means that the decision of the actions to be performed after a requested atomic calculation should depend on its outcome, which is transmitted by the interface. For instance, if the solver options foresaw n iterations for the convergence at the solution, the convergence status is transferred by the interface from the backend to the frontend, which, based on this information, takes the necessary actions, e.g., changing the solution algorithm in case of non-convergence, performing m additional iterations in case of partial convergence or, finally, continuing with the following step of the calculation.

The latter interface to be considered is between the lattice neutronic system and the external neutronic core codes. This specific interface is applicable for only one of the possible scenarios of the employment of the calculation platform but may be implemented differently according to the given reference neutronic core code to be fed. The communication between lattice and core codes occurs typically via multi-parameter nuclear data libraries, which usually take the form of hierarchical data files containing homogenized few-group cross sections, pin-power form deposited energy factors, and accompanying data needed by the specific core code like kinetic data and discontinuity factors. In this case, interface requirements originate not only as basic propagation of system requirements but also as derived generation considering the specific core code needs as a rationale. This means that this interface is bound to the request of the actual neutronic core code. From the software point of view, we stress that if the frontend can obtain the results of each state point, it may construct the resulting multi-parameter data library compatible with potentially every core code respecting the *two-step* formalism. This means that, while the interface between frontend and backend is unique given a specific backend, the interface between lattice and core neutronics can be create in a completely generic way. The relationship from the systems engineering point of view

is always the same, and this separation of responsibilities allows the definition of new lattice-core interfaces by writing only the respective output driver in the frontend. Moreover, it should be stressed that these interface requirements mandate from one side the contents of the resulting data files and from the other side the actual format: these two aspects shall be consistent with other system requirements arising from other studies than multi-parameter data library production, e.g., the validation activities employing resulting data files as output to perform parametric studies.

V. REPRESENTATIVE USE CASES

Following the systems engineering practices described in previous sections, the first step of development is the identification of the user scenarios applicable to the system to be produced. As outlined in Section IV, the user scenarios are formalized to use cases, which map directly to the acceptance tests that characterize the platform capabilities.

As already pointed out by Royce[22] and later analyzed in subsequent studies[24], this is not a once-through procedure. In fact, system analysis and software programming are very difficult to completely separate from each other since implementing a large non-trivial system often results in discovering unexpected issues and needs with respect to those identified at the initial stage. This is why systems engineering provides for a truly iterative procedure: specific procedures are prescribed to guide the final product's design. They provide a common framework in which all the stakeholders may formally exchange ideas to carry out a thorough collaborative analysis of needs before starting software development and to perform adjustments of the needs during the life cycle, assuring consistency.

More specifically, the ISO 29148 standard[23] gives the characteristics to which the requirements shall comply. They shall be necessary for the objectives of the final product, complete in their definition, unambiguous with respect to every actor involved, consistent one to the other without conflicts, and, in principle, design-free, i.e., focused on the need and not on how to achieve the need. In addition, each use case and requirement include unique identifiers that allow performing the iterative design of the system to trace at all times the modifications and their impacts on all the software components. For instance, the Easy Approach to Requirements Syntax (EARS) developed by Rolls Royce[28] provides a well-defined style that simplifies the fulfillment of the above-mentioned characteristics.

The experience acquired in lattice neutronics applications shows that sometimes calculation code kernels are developed initially with the mindset of an algorithm developer. These calculation codes are later adapted for the analysts' needs via a stack of abstraction layers that may or may not be sufficient for the final use cases. Instead, the approach proposed in this work is based on a different perspective, i.e., being able to define the design goals without considering, during the initial stages, the implementation details, but focusing on user scenarios.

When this approach is applied to a neutronic lattice calculation platform, possible use scenarios go from the classical generation of multi-parameter nuclear data libraries to specific studies for the fuel design, maintenance, verification and validation, and/or experimental setups. This procedure has been successfully applied in the CAMIVVER project for the development of the multi-parameter libraries generator prototype.[29] A systematic approach starting from the definition of Use Cases (UCs), declined with respect to the principal breakdown of platform elements outlined in Section IV (i.e., the APOLLO3® code system backend, frontend, input, and output interfaces) has been carried out and involved all actors identified in Section II. The use cases identified during this task have been formulated to collect most of the functional requirements of a neutronic lattice calculation platform.

While each item of this collection of needs may be considered a standalone identity, a preliminary analysis has been carried out to determine a general classification applicable to a generic lattice neutronics application. This classification has mainly been driven by the experiences gained using lattice codes, and it has been helpful for collaborating on tasks and for organizing discussions with stakeholders. Moreover, the formal phrasing has been useful for guaranteeing consistency.

The categories identified and employed in the CAMIVVER project are indicated as follows.

1. *Problem settings*: under the category problem settings, the basic features the user needs to state the problem to be solved are collected. They are related to the methodology employed by the analyst to instantiate data objects in the frontend so that it can create the corresponding entities in the calculation backend, which provides the numerical solution. The UCs included in this category are common to all the users of the platform.

2. *Calculation Scheme*: this category includes the needs identified by the engineer in charge of developing the calculation schemes and the ordinary reactor analyst performing routine calculations via the already defined schemes. The features identified during the CAMIVVER

work are based on the experience gained in implementing the schemes used in the French lattice platforms, but more in general for setting up an accurate calculation protocol and tuning the performances of the numerical methodologies.

3. *Assembly and Reflector design*: in this category, the needs for fuel assembly and reflector analysis in terms of modeling and calculation types are identified. More specifically, these categories correspond to the high-level representation of the fuel assemblies and reflectors and their mapping to lower-level bricks later sent to the calculation backend to provide a solution.

4. *Verification and validation* several UCs have been identified to take into account the features necessary for the engineer in charge of code and model verification and validation. These features allow for the assessment of the quality of the calculation tool with respect to accuracy and efficiency.

5. *Multi-Parameter Output*: this part is collecting the UCs related to the multi-parameter output data library structure and related tooling. The multi-parameter data library is the interface data object between a lattice calculation and a core investigation. The multi-parameter data libraries are used by reactor core analysts in neutronic core simulations, but these libraries are generally produced by users of the lattice platform that may provide all the information required by the core solvers.

6. *Processing tools*: this category includes the UCs related to the features expected in the toolset used by the reactor analyst for preparing the input of the simulation and assessing the results.

As already said, the UCs in categories 1 and 6 are user independent, i.e. they are needed by all the possible platform users, while UCs in categories from 2 to 5 depend on the applications of each actor. The joint collaboration of all the possible users allowed to have the widest possible view of all the needs and to be sure the UCs cover the different perspectives guaranteeing a uniform and coherent software implementation. This allows guiding software development to ensure building the platform that meets expectations.

At the same time, all the UCs need to be later translated to requirements to be allocated to the main components, i.e., frontend and backend, which may be further specialized in sub-

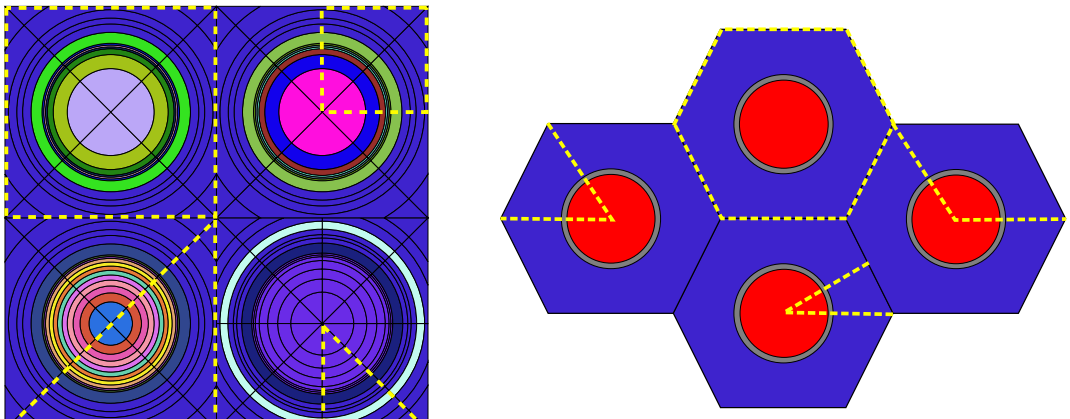
components according to the allocation of specific functionalities. This main high-level allocation to frontend and backend is up to the choice of the team in charge of the development. A clear distinction is necessary at the early stages of the life cycle to distribute work efficiently. As a general guideline, the approach followed in this work is to allocate to the frontend functionalities connected to the interaction with the user and the creation of high-level data objects close to the analyst viewpoint capable of defining the calculation sequence and generating low-level instructions related to the numerical algorithms. On the other hand, a functionality has been affected to the backend when it refers to the mathematical and numerical representation of the transport and isotopic depletion problem and their actual solution. In the CAMIVVER project, the needs and capabilities of a neutronic lattice calculation platform have been declined throughout 140 use cases. The principal concepts collected for each identified category are presented in the following sub-sections.

V.A. Problem settings

Starting from category 1, the platform shall provide the user with a uniform interface to describe the geometrical representation of configurations to investigate. This means that the frontend shall allow the user not only to define the actual engineering components, like pins, materials, cells, and their arrangement into assemblies, clusters, or reflectors but also to transform these engineering components into a model for providing a numerical representation for the solver. This means that, for instance, a fuel cell can be described using its pitch and the different radii describing its different components (fuel, gap, clad). Also, meaningful elements like detectors or custom-shaped control rods could be part of specific building blocks available to the user.

On the other hand, the requirement for the backend is to manage the data provided in input into an actual numerical representation, e.g., the radius of a pin shall be transformed into a circle with that given radius, see Figure 4(a). If an assembly also shows a repetitive pattern, the user shall also be able to define the symmetry associated with the domain, see Figure 4(b).

Regarding the material assigned to each geometrical pattern, the frontend shall offer the user the possibility to define the composition with a high-level interface close to engineering views, i.e., by providing access to concepts like isotopic mass vectors, enrichment, temperatures, and applicable source nuclear data libraries (e.g., JEFF, ENDF/B, JENDL). It is then the frontend's



(a) (b)

Fig. 4. Possible geometries and symmetries on (a) Cartesian and (b) Hexagonal cells

responsibility to determine the actual isotopic atomic compositions to fill the corresponding data objects needed by the backend to perform its processing.

In addition, the frontend shall also allow specifying options and procedures necessary to perform calculation types such as depletion and branch calculations. The frontend shall allow for performing simulations for all the configurations and state parameters identified by the scenario uses and to map them to actual conditions applicable to the numerical solutions of the backend. This transformation allows the presentation layer of the problem settings of the frontend to be universal for each neutronics lattice simulation, while the actual driver of the backend is tailored to the chosen calculation kernel. The developer should therefore perform a deep analysis of the interface between these two platform components.

Another example is the need of the user to retrieve the Quantities of Interest (QoI) on a geometry different from the one used for the calculation, which depends on the solvers used. This means that the frontend shall allow for the description of an additional geometry, superposed to the calculation geometry, on which the backend homogenizes/condenses the quantities computed from the calculation geometry, which must be provided in the output. One example of a case at the cell level is indicated in Figure 5, where the geometry for calculation used on the left is much more refined than the one selected for extracting the QoI. This means that, for instance, when the user asks for a macroscopic reaction rate on a supporting geometry that considers each pin as a separate output region, the frontend shall trigger the corresponding homogenization in the

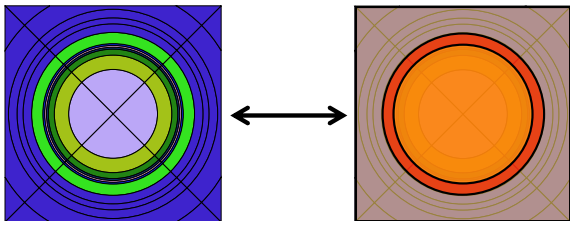


Fig. 5. Example of the change from the calculation geometry (left) to the output geometry (right) on which the backend has to compute the QoI.

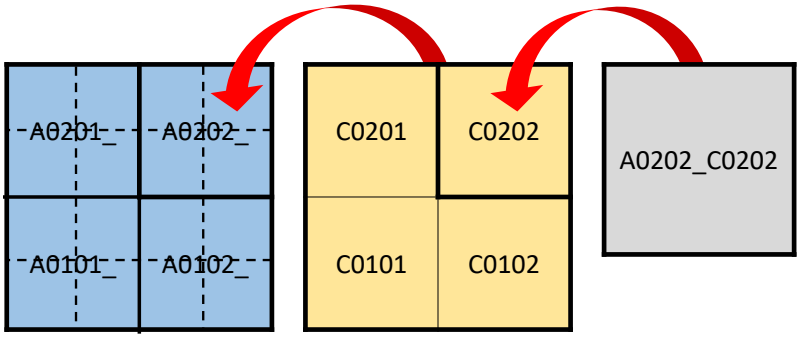


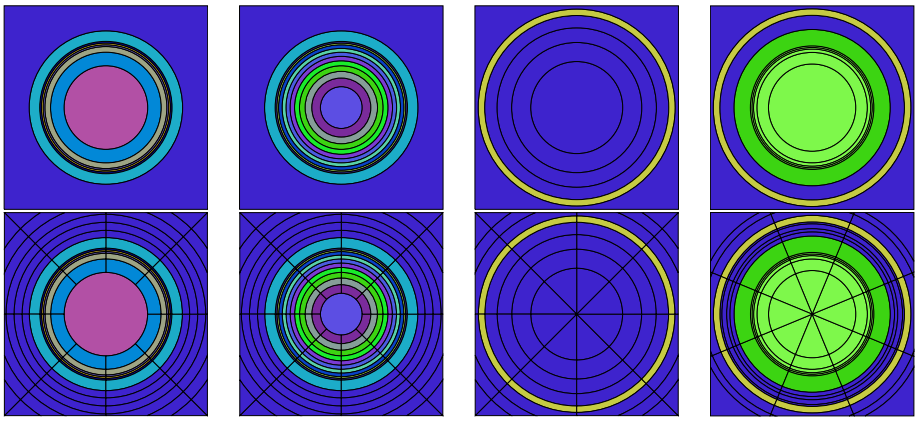
Fig. 6. Example of nomenclature concatenation for a Cartesian grid for (left) a 2×2 assembly cluster, where each (center) assembly is composed by a set of 2×2 (right) cells.

backend, which in turn shall be able to retrieve the solutions in the form provided by the actual solver employed and then perform the necessary integration for the specific output region. In fact, the specified reaction rate on the output region is provided by the integration of the reaction rate, computed with the flux, isotope concentration, and cross section of each solver cell.

To simplify this procedure, the user at the frontend shall be capable of assigning labels to each region on which the QoI shall be calculated to later perform data analysis, having the possibility to easily associate each result with the original locations. Such characteristics shall also be available when combining repetitive patterns, via, for instance, the concatenation of the naming convention chosen, as shown in Figure 6.

V.B. Calculation Scheme

One of the aspects that is taken into account for defining the calculation scheme is the definition of solver options that will be accessible to the users. The choice of the solver and the options allow the user to perform several types of calculations such as self-shielding, neutron and photon transport, and isotopic depletion. It should be stressed that different actors, as outlined



(a) UOX (b) UOX+Gd (c) Guide tube (d) B4C

Fig. 7. Different meshes for various pin types for self-shielding (top) and flux calculations (bottom).

in Section II, may have quite different requirements, ranging from the need for very fine-grained tuning of computation options to just the access to predefined black-box recipes.

As depicted in Figure 7, self-shielding and flux calculations may employ different spatial meshes, linked to the basic cells representing the physical features of the lattices under analysis. In addition, homogenization may correspond to the need of integrating results on a separate, but superposed, geometrical region, as shown in Figure 5.

The user shall be capable of defining, via the frontend, the different refinements applied to self-shielding and flux calculations, assuring coherent mapping between them. It may happen that the two geometrical supports on which self-shielding and flux calculations are executed do not coincide: for instance, the self-shielding may be run on reduced representative patterns, while the flux calculations are bound to the whole geometry.

When dealing with self-shielding calculations, the user shall be able to employ different models, e.g., equivalence/fine structure and/or subgroup. It should be underlined that different energy ranges of the same isotope may be linked to different models. Moreover, advanced calculation schemes may require the definition of different spatial patterns in which carrying out self-shielding depending on each isotope. For instance, it should be possible to self-shield ^{238}U and ^{235}U on different geometries.

At the same time, flux calculations often require combining multiple solvers, as already introduced in Section I. For instance, spectral reconstruction is a key feature of multi-level schemes like REL2005[30] and should be available as a backend functionality. Finally, the fine-tuning of the

calculation scheme may involve the SPH equivalence steps[31] and/or the calculation of Assembly Discontinuity Factors (ADF)[7].

The choice of the solvers employed, and therefore of the numerical algorithms under the hood, is a delicate equilibrium in which the calculation scheme specialist needs to strike a balance among robustness, accuracy, and response time. Therefore, a comprehensive toolkit of options is paramount in both the industrial and research sectors to conceive fast yet accurate calculation schemes.

It is quite delicate to decide whether such fine-grained computation settings shall be managed by the frontend via a high-level orchestration or by the backend via its inner mechanics. In this context, we support the idea that the algorithm developers are best suited for deciding the optimal numerical strategies, provided that a coherent definition of the presentation layer of the backend could offer the advanced analyst the possibility to tune the above-mentioned modeling options. An example of the flexibility employed by the engineer in charge of calculation scheme definition is presented in Section VI, where the focus is oriented to solver options investigations.

For each application, the frontend should offer a way to serialize the prescribed calculation scheme strategy for later use, for instance, by combining relevant sequence scripts and dictionaries of options of each underlying numerical method, to be sent to the backend for processing. In this sense, the backend is responsible for modeling flexibility, while the frontend is responsible for driving simulations via calculation recipes.

V.C. Assembly and Reflector design

The assembly and reflector calculations have been grouped together in the category 3 due to similar features required by the users, even though the geometries and the physics to be treated are different.

The engineer in charge of the assembly design shall be able to assess all the fuel assembly typologies included in the loading pattern, as well as the configuration of control/safety rods and detectors, and the reflectors. More specifically, reflector calculations involve combining multiple calculations, as prescribed, for instance, by the Lefebvre-Lebigot method[32] or more advanced modeling as seen, for example, in [33]. In addition, specific studies may require the analysis of patterns of multi-fuel assemblies, as depicted in Figure 8.

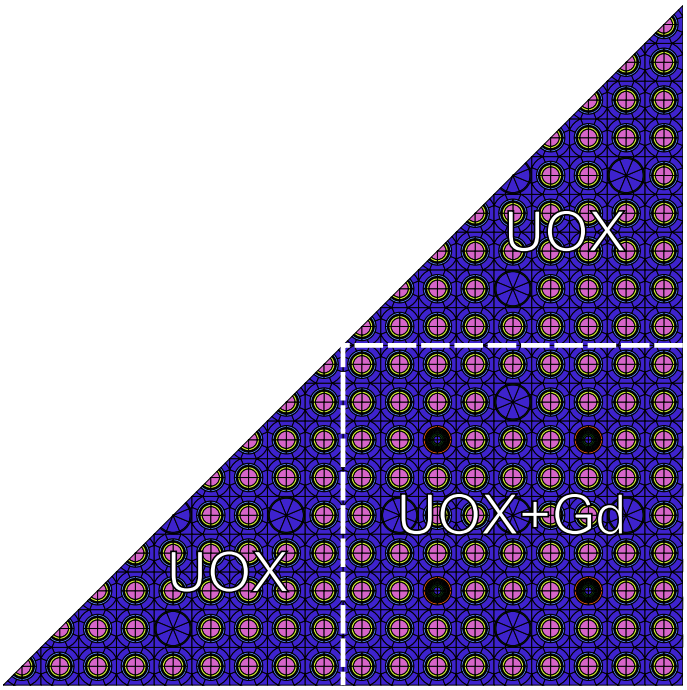


Fig. 8. A possible layout of a *cluster* configuration for the calculation of the UOX/UOX+Gd interface for a 9x9 assembly with one-eight symmetry.

Access to these representative patterns is an important feature for nuclear fuel and core designers: for instance, when dealing with 3D objects, the user shall be able to specify the amount of 2D slices. In this case, the frontend potentially sees the real fuel assembly and then it assembles the 2D slice, thanks to the basic bricks representing the actual 3D object in a numerical form. From a generic point of view, the frontend shall be able to provide high-level data objects connected to the concepts of fuel assemblies and reflectors. They shall provide the associated configurations related, for example, to control rod insertion or extraction and modification of specific fuel pins after a reparation. Also, studies like reflector response calculations or interfaces among different fuel assemblies may require composing the very same physical configurations with different numerical settings considering or not other elements in the environment. Looking again at Figure 8, it is possible to consider the case in which the UOX+Gd fuel assembly is at some stages of the calculation considered alone and at other stages surrounded by other UOX fuel assemblies. These rather complex calculation recipes involve the dynamic modification of the geometrical and material supports on which the solver is run. The frontend is, therefore, responsible for constructing the actual geometry employing the basic bricks as seen in Section V.A, while guaranteeing the coherence

among the physical quantities sent and received via the interface with the calculation kernel.

From these considerations, it becomes clear that the platform is fully responsible for translating engineering modeling into numerical protocols, and an interactive processing is needed for advanced and specific applications.

V.D. Verification and validation

This category 4 refers to UCs related to the verification and validation of the neutronic code and calculation scheme performed. While defining the calculation schemes involves fine-grained tuning of the calculation options, verification and validation are more connected to assessing already existing recipes. Two types of activities are commonly performed:

- the *numerical verification* aims at quantifying the accuracy of the neutronic calculation schemes by comparing them with continuous-energy Monte Carlo simulations (TRIPOLI-4® for example) with the same nuclear data library (e.g., JEFF) or with an already validated deterministic code (APOLLO2 for example);
- the *experimental validation* corresponds to the comparison of the results of the global simulation package against experimental results from integral measurements (mainly in reactors).

For these purposes, the *verification and validation experts* need to compare the results provided by the considered neutronic code to those obtained with other references, including experimental setups or already validated codes when using different solver options and calculation schemes.

It means that the analyst, on the one side, freely composes geometrical basic elements and/or defines new ones starting from an unstructured representation to accurately match the experimental setups and, on the other side, inspects interactively the results returned by the calculation kernel. The latter is necessary to allow constructing meaningful representations of QoIs for further assessment, as shown, for instance, in Figure 9.

In addition, the verification and validation of a coupled transport and depletion solution may be connected to the capability to introspect the current isotopic concentration of trace elements such as ^{148}Nd and to compare it with available experimental data. When making a code to code comparison, it should be possible to retrieve relevant information from an already executed calculation that should trace its specific setup, that in turn may be modified to perform sensitivity

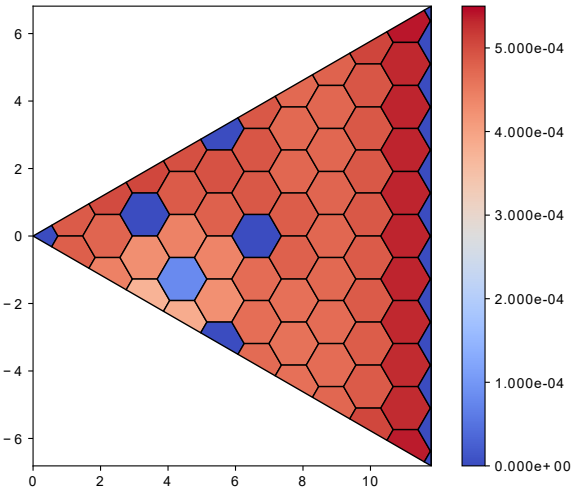


Fig. 9. Sample representation of U238 absorption rate on a hexagonal fuel lattice in $cm^{-3}\,s^{-1}$.

analysis with respect to each parameter characterizing the problem under analysis. Also, since each code may return data in different symmetries, the analyst may need to obtain the QoI on unfolded plain geometry.

In terms of requirements, this means that the frontend shall allow the user to fully tune both the problem in terms of the description and the settings to be transferred to the solver. The results obtained may be collected and compared against reference calculations in aggregate form. The actual capability of performing these analyses depends on the numerical methods available in the backend solvers, so that the verification may occur only when considering the actual strengths and limitations of the numerical strategies available. Again, from these use cases defined by the validation expert, it is clear that the frontend and the backend are tightly bonded to ensure the coherence and the operation of the platform and that the various users have different needs that impact different parts of the platform.

As a case of application, shown more in detail in Section VI, the verification and validation experts shall have the possibility to verify the quality of the choices done on the calculations scheme. In the example shown in Section VI, the results of the lattice code APOLLO3® are compared to those obtained with the industrial code APOLLO2-A and the Monte Carlo code TRIPOLI-4®, when analyzing the application or not of the normalization of the trajectories defined by the Method Of Characteristics (MOC)[34] transport solver.

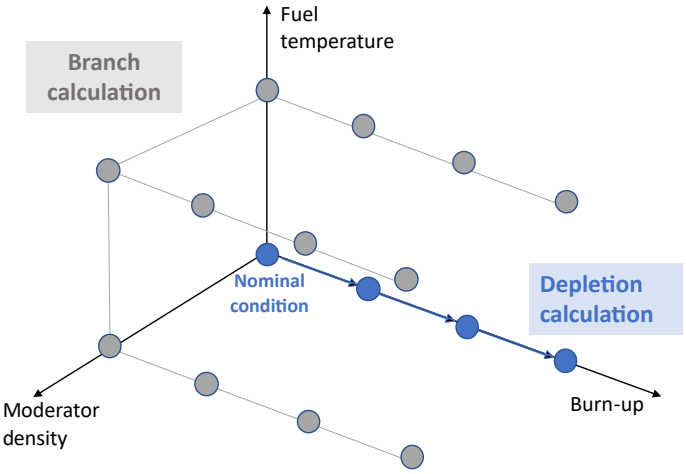


Fig. 10. Schematic representation of the parameter tree for a generic multi-parameter library with three different parameters: burn-up, fuel temperature, and moderator density. In typical applications, several other possible state parameters are considered, e.g. boron concentration, rod configuration, etc.

V.E. Multi-Parameter Output

This category 5 refers to UCs related to the multi-parameter output data library structure (called MPO in the APOLLO3® namespace) generated by the lattice calculation platform. This category of UCs allows links to be made between the lattice calculation uses and the related core calculation needs.

The multi-parameter library is generated by combining several types of calculation: the depletion and successive branch simulations, as is shown in Figure 10. For this purpose, the engineer may want to choose specific state parameters and their associated values that depend on the needs of the core calculations, e.g. application and interpolation scheme at the core level. From the point of view of the requirements for the lattice platform, the code shall be able to change the state parameters according to the value prescribed by the user via the frontend and deal with the transformation of simple transport solutions and homogenization on the output geometry.

To feed the core model, the multi-parameter library shall respect a format prescribed by the core code input interface. Several core solver exists: for example, in the CAMIVVER project, the core calculations are carried out with APOLLO3®. It means that, in this case, the multi-parameter library shall respect the APOLLO3® characteristics.

The flat sequence of calculation points is organized in a hypercube covering the possible

combinations of the selected state parameters, for instance, boron concentration, fuel temperature, and moderator density. Consequently, the frontend shall allow the user to specify the various values that can be assumed by each parameter and orchestrate the ensemble of state points obtained with the vector product of the set of values of all specified parameters. Orchestration means not only the organization of the state points in the hypercube, but also the distribution of the calculation effort on a set of processors that may be located on a single workstation or on separate computational nodes of an HPC cluster. Consequently, preparing a multi-parameter output data library involves the definition of the lattices and/or reflector to be analyzed (as seen in Section V.C), the implementation of the calculation scheme provided in Section V.B, and the actual selection of the computational machines on which to execute the instances of the backend.

The results of these calculations are typically stored in one or multiple multi-parameter data files. It follows that the component in charge of this operation is a critical design choice. It may be the frontend via a dedicated output driver that may be changed for each core solver to be employed without the need for costly conversions, or it may be the backend with its inner mechanics, with potentially some benefits if lattice and code solver are included in the same package. In either case, it is important that the multi-parameter object file must associate each state point with its corresponding values for each one of the state parameters used, so that either the core solver or the core analyst can retrieve the operating conditions of each calculation point.

V.F. Processing utilities and tools

An additional category of UCs is related to processing utilities and tools. Indeed, during the simulation, the engineer using the platform may need to assess the ongoing calculation to perform branching decisions and/or identify potential issues.

In order to take into account the needs identified by different actors, several methodologies exist to address this task. A dedicated option of the calculation backend may be activated to enable a tracing that may take the form of a dedicated output file, typically in the form of listing, containing debugging messages normally related to functions calls and data flow contents. The interest in this kind of message is somewhat limited, especially when symbolic debugging is available since this methodology does not offer the possibility to adopt on-the-fly corrections of the data processing. A more interesting option is available when the calculation backend offers an interface

allowing to interrogate the state of each specific calculation. In this case, the upper software layer of the calculation backend is organized to accept user input data in the form of specific user data objects and to provide the results and the states via outcome objects and/or modifications of the initial objects. At this point, the engineer may continuously assess the calculation details in a programmable way: branching decisions and on-the-fly corrections of potential issues are therefore possible. In addition, stopping and restarting the simulation with different modeling options or discarding some calculation points suffering from instabilities may become part of routine activities.

Processing utilities also include post-processing tools to analyze the results, and this turns into requirements for the frontend that shall allow the users to choose which quantities of interest have to be plotted and to manipulate the multi-parameter data objects. To facilitate the employment of multi-parameter object data files, it is recommended to provide a unique interface from which the engineer may retrieve the results, using the state parameters as selection keys.

VI. APPLICATION CALCULATION SCHEME OPTIONS INVESTIGATION

The overall analysis based on the identification of use cases presented in the Section V has been applied to a real-world scenario to show how the formal functional requirements maps to a given task. Suppose that we are in the position of a *calculation scheme and verification and validation expert* who needs to improve the accuracy of the calculation protocol by tweaking the solver options. As seen in Section II, the developer's perspective of calculation methodologies contains the elements of the ordinary end user with additional needs of advanced functionalities.

Starting from the consideration that present lattice codes often relies on the Method Of Characteristics (MOC)[34] for the resolution of the transport equation, the case study is designed to illustrate the global process discussed in the previous sections by analyzing the trajectory normalization strategies in the long characteristics solver. In the last 20 years, solvers based on the method of characteristics are gaining increased application in the industrial sector given their capabilities to precisely represent the spatial features in a full unstructured geometry.[35] They can be seen as an iterative solution of the collision probability method that employs a discrete ordinates (SN) angular representation and supports angular anisotropy.[36]

In order to show the impact of two normalization options (with and without angular normalization), two simulation toolboxes have been used: the industrial platform APOLLO2-A[27] and

the next generation code APOLLO3®[14]. The reference will be provided by the Monte Carlo code TRIPOLI-4®[18].

Trajectories normalization, following initial Askew methodology[37], aims to force coherence between the numerical and analytical volumes. In the trajectory-based domain discretization, it is possible to compute a numerical volume for each region and direction of the SN quadrature formula. The difference between the numerical and analytical values is a good indicator of the quality of the spatial integration, which is necessary to obtain a precise simulation. The ratio between a volume's analytical and numerical value can be used to normalize the chords length crossing a computational region for a given direction. The drawback of this approach is that artificially modifying the length of the chords for each computational region directly impacts the particle transport since it modifies the optical length ($\tau = \Sigma l$), which is a key factor in the integral form of the transport equation. This can be problematic, particularly when applying a symmetry boundary condition for cyclic trajectories, which can result in computing very long trajectories.[38] In these cases, the cumulative difference between a real and a normalized trajectory can be important. Therefore, this analysis aims to quantify the impact of these two calculation settings in real-world fuel assemblies.

In this type of parametric study, it is necessary to be able to test the impact of the scheme option on a set of conditions and parameters that are representative of the domain on which the calculation scheme should be validated. Three types of fuel have been considered: uranium-based oxide (UOX), mixed uranium and plutonium oxide (MOX), and uranium-based fuel with gadolinium pins (UGD). Their geometries, material compositions, and operating conditions are determined based on the information available in public benchmarks.[39][40] These fuel assemblies have been depleted up to $30\,\mathrm{GW\,d/}t_{\mathrm{IHM}}$ in nominal conditions by considering a power mass density of $37.5\,\mathrm{W/g}$ for UOX and UGD and $36.6\,\mathrm{W/g}$ for MOX. Configurations without control rods inserted (ARO) and with the insertion of an absorbent rod cluster (AIC, B4C) are investigated by modifying the simulation configuration for selected points among the irradiation steps.

The first step is to define for each code the geometric representation of the three test cases considered. This type of analysis implicitly considers a high-level abstraction that should be capable of managing the concept of fuel assembly as seen in Section V.C while having multiple configurations as required in Section V.A. This is a typical example of frontend structure, close to the engineering view. At the same time, the study aims to investigate the accuracy of the solver. It

treats the transport equation on a given spatial mesh, which is a data object processed internally by the backend.

Both simulation tools allow defining this geometrical representation through a set of blocks and keywords/values in a rather similar way to their native interface. It is worth noting that this particular characteristic is managed inside APOLLO2-A by the industrialized frontend, while in the case of APOLLO3® this is provided by the native backend interface. Requirements generated in Section V.A are therefore fulfilled by the systems through different architectural decompositions. One may stress that the separation between frontend and backend responsibilities is part of the specification tree as outlined in Section IV: the chosen architecture will decide the exact allocation of responsibilities and, therefore, the flexibility of the platform as a whole. At the same time, the end user is bound to high-level use cases that correspond to acceptance criteria regardless of the actual architecture chosen, provided that the functionalities are available.

Once the geometry is defined, the material compositions must be linked to it. Indeed, the geometrical representation given by the user should be transformed into a spatial mesh and an ensemble of media that contain the cross section information, which in turn depends on the isotopic mixture. Typically, engineers employ isotopic vectors in terms of mass density, potentially from the data available during manufacturing and/or before fabrication. At the same time, nuclear data libraries provide raw microscopic cross sections, which are later employed to construct macroscopic cross sections included in the media associated with the spatial regions processed by the solver.

For this reason, lower-level backends in APOLLO2 and APOLLO3® require giving the isotopic compositions directly in at/cm^3. APOLLO2-A, through its pre-processor, allows defining these compositions by using mass densities associated with isotopic vectors or pre-defined mixing, which are later translated to at/cm^3 for its computational kernel. This user interface has been chosen to let the engineers think and interact with the technology databases in terms of mass densities and, eventually, isotopic vectors. It is, therefore, the responsibility of the frontend to perform the translation between mass density and isotopic composition, as seen in Section V.A

The operating conditions (temperature, rod configurations, the irradiation range on which the specified comparisons are performed, etc.), can then be assigned by the user. They act as an ensemble of state parameters from which the actual condition under analysis should be defined. More specifically, the computational kernel may solve the neutron transport and depletion equa-

tions for a given set of parameters characterizing the current fuel assembly. The responsibility of the frontend is thus to provide the input data objects capable of collecting state parameters and later feeding them to the computational backend for media modifications and isotope evolution. APOLLO2-A manages and optimizes the ordering of the loop of state points during its preprocessing before preparing the corresponding execution sequence for APOLLO2. APOLLO3®, instead, provides an interactive user interface that allows calling specific calculation blocks interactively.

The next building block of the analysis is to reproduce the same calculation scheme, considering the same options with both codes. As seen in Section V.B, it is based on three main requirements:

- the possibility to define the calculation scheme by combining several operations such as self-shielding, spatial meshing, flux solver, output homogenization, leakage model, depletion model, etc., as a sequence;
- save this sequence to a serialized format to allow its implementation in multiple calculations without redefinition;
- the ability to tweak parameters of a given calculation scheme from an already available recipe.

In the context of this study, a best estimate calculation scheme has been implemented, which consists of a single level MOC calculation at 281 energy groups (SHEM-MOC).[41] In APOLLO2-A this sequence of operations and their options are defined through an ensemble of relevant options saved in a keyword-based data file together with blocks of instructions for APOLLO2. The frontend is then responsible for translating the protocol in the corresponding calls of the dynamic language called GIBIANE, which interacts with the calculation kernel. This interface is located downstream of the frontend, and the option for normalizing trajectories is not available in the configuration options. Since it is not directly accessible from the frontend, working at this lower level is required. In APOLLO3® on the other hand, this option is exposed through a dedicated dictionary from which the calculation protocol is defined. One may note that the requirements identified in Section V.B are fulfilled in a quite different way when using these two software stacks since in APOLLO3® only the relevant dictionaries of options should be managed, provided that they offer the configuration requested, while in APOLLO2-A it may be needed to operate at a higher level in the frontend, or at lower-level by intercepting the calls to the calculation backend

with a specific tool that modifies the GIBIANE calls on the fly. This example shows how striking a balance between flexibility and simplicity may result in different solutions, which, again, should pass the acceptance criteria induced by the use cases.

As mentioned before, validating the parameter(s) of interest (here, the normalization strategy) on the largest possible domain is often useful. This implies a significant number of parameters to be implemented. Performing such parametric study is more effective when dealing with a calculation tool offering a programmable input and/or interactive processing. In this specific case, this implies dynamic modification before run-time to parameterize the calculation with a given set among the whole Cartesian product considered, which, in this case, corresponds to the selected fuel assembly technologies and the solver options under study.

Once this is done, it is necessary to define a calculation sequence to establish the studied domain. First, the calculation is done for a given state. Next, it is possible to perturb this state (for example, with the insertion of an absorbent rod cluster). Finally, the fuel is depleted, and the previous steps are repeated. As seen before, the generation of this sequence is part of the frontend responsibilities that are available directly in APOLLO2-A or that can be implemented via the dynamic interactive processing in APOLLO3®.

One may note there is an important difference between the state parameters, which corresponds to operating and/or accidental conditions studied for a given fuel assembly and a given calculation protocol, and those that correspond to a parametric analysis, which corresponds to the ensemble of fuel assemblies and calculation options under analysis. While the former is strictly related to the capability to assemble all the results corresponding to the state parameters in a unique output data object as seen in Section V.E, the latter is intrinsically connected to the interactivity of dynamic processing requested in Section V.F, i.e., the ability to reuse and programmatically modify input decks to facilitate parametric analyses.

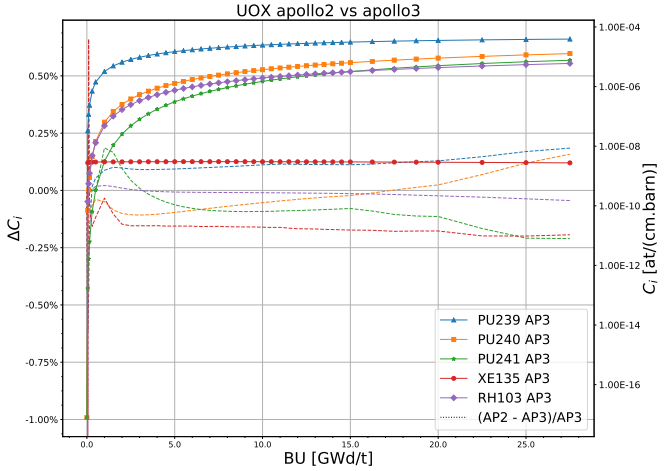
For verification and validation purposes, as discussed in Section V.D, the platform shall provide the results in an accessible format, so that the frontend can easily query these results and simplify the post-processing of the results. This is strictly connected to the need for introspection of data-like objects containing the results of the calculations, which should be based not only on a well-defined output data format, but also shall support the capability to define general-purpose API. Both platforms support those requirements (except during interactive processing), having

accessible results with the HDF5 file format[42]. APOLLO2-A already provides a specific API to query the results, which has been developed on top of the APOLLO2 backend. On the other side, APOLLO3® guarantees this possibility through the chosen format. A dedicated post-processing tool has been therefore developed to perform this activity.

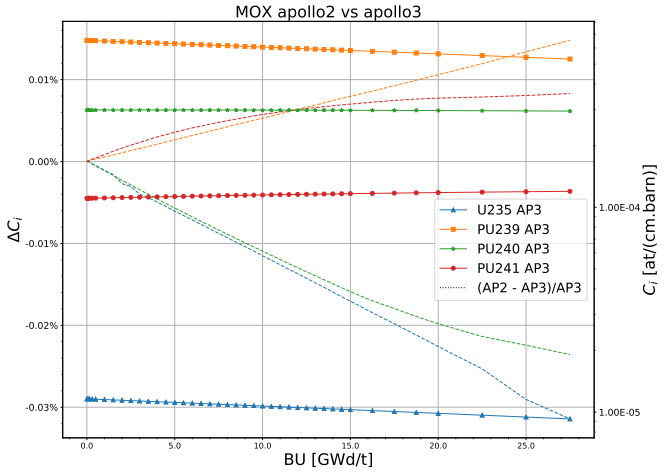
Numerical verification also demands comparisons with Monte Carlo simulations. Translating high-level problem descriptions into Monte Carlo decks by filling in the geometric and composition information with each state of the calculation is, therefore, a significant asset to enable an efficient comparison for a broad test base. In parallel, one may note that Monte Carlo transport solvers require isotopic compositions, which are the results of the solution of the Bateman depletion equations. The preparation of Monte Carlo input decks in the framework of code-to-code comparisons should thus take into account a unique set of material definitions during depletion. In this work, the automatic generation of TRIPOLI-4® decks included in APOLLO3® has been employed, which is capable of re-injecting the isotopic mixing computed during evolution in a set of separate input files for each burnup step.

To ensure correctness of the TRIPOLI-4® decks generated by APOLLO3®, it is necessary to ensure that the compositions, during the evolution, do not deviate significantly with respect to APOLLO2. This is shown in Figure 11: for both UOX (a) and MOX (b), the deviation does not exceed 0.2 % on the main fissile isotopes, as well as on the main fission products. The UGD (c) case shows that for both ^{155}Gd and ^{157}Gd, the deviation is less than 2 % as long as a significant amount remains.

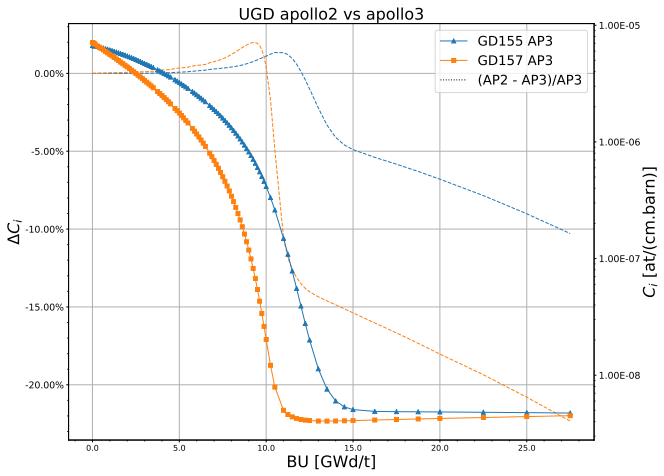
Figure 12 shows the result obtained with (ANGL, red) and without (NON, blue) angular normalization. The mean absolute difference, as well as the standard deviation in reactivity compared to TRIPOLI-4® are presented for the All Rods Out (ARO) configuration, together with the rodded (AIC, B4C) configuration. For both APOLLO3® and APOLLO2-A, no normalization gives better results for rodded configuration, decreasing the average deviation on reactivity by about 100 pcm for AIC, and about 140 pcm for B4C. The dispersion of differences is also slightly reduced. There is almost no change when dealing with the ARO configurations. These results are consistent with recommendations of CEA experts[43]: instead of activating trajectory normalization, it is more effective to use the error between the numerical and angular values in order to check if the chosen integration parameters, i.e., the distances between trajectories for numerical



(a)



(b)



(c)

Fig. 11. Relative difference APOLLO2-A vs APOLLO3® and associated isotopic concentrations in APOLLO3® for several isotopes: (a) UOX; (b) MOX; (c) UGD.

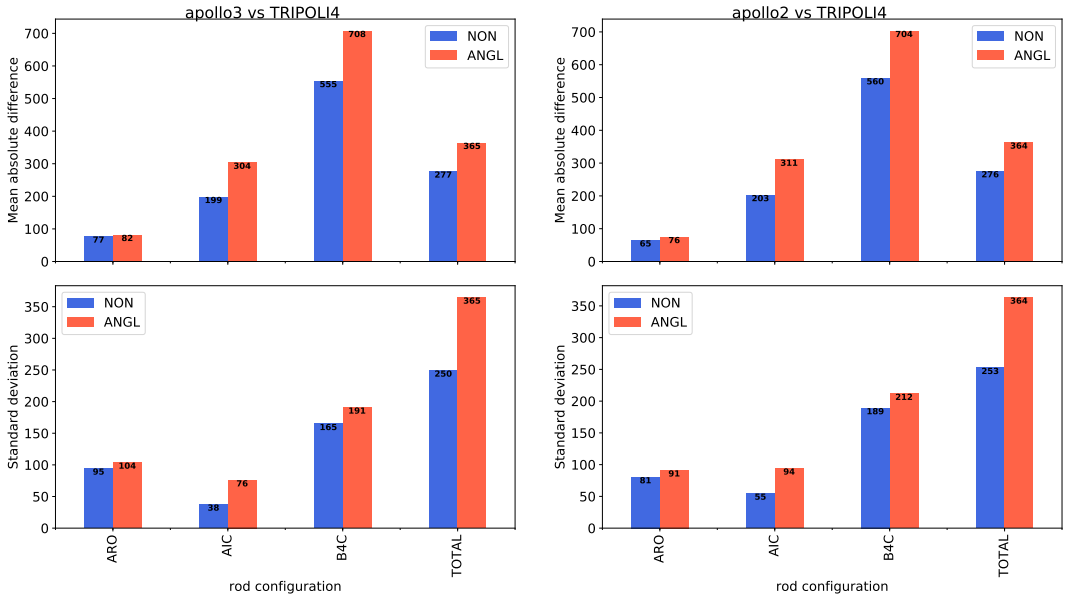


Fig. 12. Mean absolute difference and standard deviation on reactivity against TRIPOLI-4® for several rod configurations with (ANGL) and without (NON) angular normalization: (a) APOLLO3®; (b) APOLLO2-A.

angular volumes and distance and quadrature formula and the number of directions for numerical volumes, lead to a precise enough numerical integration, instead of forcing the correct values of the volumes and artificially modifying the distance traveled by the particles.

VII. CONCLUSIONS

The application of transport theory to the analysis of the neutron and photon population inside a nuclear reactor originates from the problem of specifying the radiation field in an atmosphere that scatters light and it was well-defined and thoroughly studied before the discovery of the neutron.[44]. Around the middle of the XX century, and especially at the end of World War II, this theory found its application also to solve the neutron field inside the multiplying media of interest for nuclear technology. New deterministic methodologies have since been conceived, gaining popularity in solving the transport problem.[45, 46, 47] In parallel, Monte Carlo methodologies have been conceived and steadily developed, representing a complementary technology to tackle neutron and photon population analysis.[48]

Fast-forward to the beginning of the XXI century, 80 years after the first criticality of the

Chicago Pile-1, the deterministic and stochastic numerical solutions of the neutron transport problem are routinely applied in the industrial sector to conceive, assess, and operate nuclear reactors. While research efforts are still ongoing to improve solving techniques, it should be recognized that the maturity of the numerical methods has opened the doors to widespread application in industrial activities, where the user scenarios may be subject to quite different requirements. While recognizing the paramount importance of the research of new solver techniques, this work aims at demonstrating that an equal effort should be put in place for conceiving efficient calculation platforms compatible with user needs.

This work demonstrates how the systems engineering approach is capable of identifying the organizational and technical relationships between stakeholders and software components in the field of codes and methods in neutronics. Without taking anything away from the general applicability of this technique, the analysis focused on developing a lattice calculation platform to be applied to PWR and VVER light water reactors. This methodology has been successfully applied at Framatome, in collaboration with its partners for the lattice neutronics work package of the H2020 CAMIVVER project.

In particular, the analysis of software architecture choices underlined how the clever management of the life cycle of data objects via a well-defined transition between states and the association of memory to the appropriate layer is critical for composing several operations in a functional way. On the other side, monolithic approaches applied to very efficient underlying solvers may undermine the effective application of the most advanced numerical technologies because of the inability to interactively combine different software packages in the same framework.

The importance of a tight collaboration with all the actors involved in the calculation platform from both the development and end-user perspectives has been thoroughly analyzed and formalized with a requirements management plan that shows the relationships with internal and external constraints. The formal generation of use cases permitted to classify the typical needs of a lattice neutronics analyst in a coherent set of categories that fulfill a wide spectrum of scenarios for all the actors involved. Following the proposed process of organizing correspondence among the needs of the actors detailed at different abstraction levels and their allocation to concrete software components alleviates the complexity of developing a lattice neutronics calculation platform, while still maintaining full traceability of the support for user scenarios. To better provide a real-world

application of this methodology, a case study related to a calculation scheme and numerical verification activity for fuel assembly assessment has been presented underlying the connections with the use cases presented. This study provides a general guidance for the application of normalization options in the tracking operations for methods of characteristics solvers, and showed how angular normalization may provide worse results with respect to Monte Carlo references.

Therefore, the purpose of this work is to provide general guidance on an efficient methodology to boost collaboration among teams and develop simulation toolboxes while taking into account the needs in a systematic way along the product life cycle.

ACKNOWLEDGMENTS

The H2020 CAMIVVER Project received funding from the Euratom Research and Training Programme 2019–2020 under grant agreement No. 945081. The authors would like to thank the Framatome colleague Laurent Graziano and the former intern Alessia di Francesco for their efforts in preparing input decks and post-processing tools for the APOLLO3® code. The authors would like to thank the Framatome colleague Laurent Graziano for his expertise and support on the methods of characteristics solver in APOLLO2 and APOLLO3®. The authors would like to thank the APOLLO3® development team for their efforts in developing the code and especially Pietro Mosca, Simone Santandrea, Daniele Tomatis, Jean-François Vidal, and Igor Zmijarevic for the support during the analysis of APOLLO3® as a neutronic calculation backend. The authors would like to thank the EDF colleagues, especially Ivan Dutka-Malen, Pierre Laurent, Yohann Pipeau, and Adrien Willien for interesting discussions about the overall construction of a lattice neutronic platform. The authors would like to thank the Framatome colleague Wesley Ford for his help in improving the overall language quality of the manuscript. APOLLO3® is a registered trademark of the CEA developed under a long-term partnership and support of EDF and Framatome.

REFERENCES

[1] R. Sanchez, "Prospects in deterministic three-dimensional whole-core transport calculations," *Nuclear Engineering and Technology*, **44**, *2* (2012); 10.5516/NET.01.2012.501.

[2] A. Hébert, *Applied Reactor Physics*, Presses internationales Polytechnique (2009).

[3] J. Dufek, "Building the nodal nuclear data dependences in a many-dimensional state-variable space," *Annals of Nuclear Energy*, **38**, *7*, 1569 (2011); 10.1016/j.anucene.2011.03.006.

[4] A. Santamarina, C. Collignon, and C. Garat, "French calculation schemes for light water reactor analysis," *The Physics of Fuel Cycles and Advanced Nuclear Systems: Global Developments (PHYSOR 2004)*, Chicago, Illinois, USA, 25-29 April 2004, American Nuclear Society (2004).

[5] D. Tomatis, "Compression of Assembly Power Form Factors for Core Calculations," *Nuclear Science and Engineering*, **193**, *6*, 622 (2019); 10.1080/00295639.2018.1553428.

[6] D. Tomatis, "A multivariate representation of compressed pin-by-pin cross sections," *EPJ Nuclear Sci. Technol.*, **7**, 8 (2021); 10.1051/epjn/2021006.

[7] K. S. Smith, "Assembly homogenization techniques for light water reactor analysis," *Progress in Nuclear Energy*, **17**, *3*, 303 (1986); 10.1016/0149-1970(86)90035-1.

[8] R. Sanchez, "Assembly homogenization techniques for core calculations," *Progress in Nuclear Energy*, **51**, *1*, 14 (2009); 10.1016/j.pnucene.2008.01.009.

[9] J.-F. Vidal, P. Archier, B. Faure, V. Jouault, J.-M. Palau, V. Pascal, G. Rimpault, F. Auffret, L. Graziano, E. Masiello, and S. Santandrea, "APOLLO3 homogenization techniques for transport core calculations-application to the ASTRID CFV core," *Nuclear Engineering and Technology*, **49**, *7*, 1379 (2017); 10.1016/j.net.2017.08.014.

[10] A. Galia, I. Zmijarevic, and R. Sanchez, "3D core calculation based on the method of dynamic homogenization," *Transition to a Scalable Nuclear Future (PHYSOR 2020)*, Cambridge, United Kingdom, 29 March - 2 April 2020 (2020).

[11] P. Girieud, I. Daudin, C. Garat, P. Marotte, and T. S., "Science version 2: the most recent capabilities of the Framatome 3-D nuclear code package," *International Conference on Nuclear Engineering*, Nice Acropolis, France, 8-12 Apr 2001, Societé Française d'Énergie Nucléaire (SFEN) (2001).

[12] A. Calloo, D. Couyras, F. Févotte, M. Guillo, C. Brosselard, B. Bouriquet, A. Dubois, E. Girardi, F. Hoareau, M. Fliscounakis, H. Leroyer, E. Noblat, Y. Pora, L. Plagne, A. Ponçot, and N. Schwartz, "COCAGNE: EDF new neutronic core code for ANDROMÈDE calculation chain," *International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering (M&C 2017)*, Jeju, Republic of Korea, 16-20 April 2017, Korean Nuclear Society (2017).

[13] D. Verrier, B. Vezzoni, B. Calgaro, O. Bernard, A. Previti, C. Lafaurie, A. Hashymov, P. Groudev, A. Stefanova, N. Zaharieva, F. Damian, P. Mosca, D. Tomatis, U. Bieder, A. Willien, N. Dos Santos, L. Mercatali, V. H. Sanchez-Espinoza, N. Forgione, and S. Paci, "Codes and Methods Improvements for VVER Comprehensive Safety Assessment: The CAMIVVER H2020 Project," *28th International Conference on Nuclear Engineering*, vol. 2: Nuclear Fuels, Research, and Fuel Cycle; Nuclear Codes and Standards; Thermal-Hydraulics, Virtual, online. 4-6 August 2021 (2021); 10.1115/ICONE28-64169.

[14] D. Schneider, F. Dolci, F. Gabriel, J.-M. Palau, M. Guillo, B. Pothet, P. Archier, K. Ammar, F. Auffret, R. Baron, A.-M. Baudron, P. Bellier, L. Bourhrara, L. Buiron, C. De Saint Jean, J.-M. Do, B. Espinosa, E. Jamelot, V. Jouault, J.-J. Lautard, R. Lenain, J.-C. Le Pallec, L. Lei Mao, E. Masiello, F. Moreau, P. Mosca, M. Muniglia, N. Ordy, V. Pascal, B. Roque, A. Targa, C. Patricot, S. Santandrea, D. Sciannandrone, J.-F. Vidal, and I. Zmijarevic, "APOLLO3 CEA/DEN deterministic multi-purpose code for reactor physics analysis," *Unifying Theory and Experiments in the 21st Century (PHYSOR 2016)*, Sun Valley, USA, May 2016 (2016).

[15] D. Tomatis, F. Bidault, A. Bruneton, and Z. Stankovski, "Overview of SERMA's Graphical User Interfaces for Lattice Transport Calculations," *Energies*, **15**, *4* (2022);

10.3390/en15041417.

[16] R. Sanchez, I. Zmijarevic, M. Coste-Delclaux, E. Masiello, S. Santandrea, E. Martinolli, L. Villate, N. Schwartz, and N. Guler, "APOLLO2 year 2010," *Nuclear Engineering and Technology*, **42** (2010); 10.5516/NET.2010.42.5.474.

[17] B. Vezzoni, S. Santandrea, L. Graziano, and I. Zmijarevic, "Best estimate schemes in lattice calculations with the help of a new leakage synthetic algorithm," *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2021)*, Raleigh, North Carolina, USA (2021).

[18] E. Brun, F. Damian, C. M. Diop, E. Dumonteil, F. X. Hugot, C. Jouanne, Y. K. Lee, F. Malvagi, A. Mazzolo, O. Petit, J. C. Trama, T. Visonneau, and A. Zoia, "TRIPOLI-4®, CEA, EDF and AREVA reference Monte Carlo code," *Annals of Nuclear Energy*, **82**, 151 (2015); 10.1016/j.anucene.2014.07.053.

[19] W. R. Martin, "Challenges and prospects for whole-core Monte Carlo analysis," *Nuclear Engineering and Technology*, **44**, *2*, 151 (2012); 10.5516/net.01.2012.502.

[20] B. D. Ganapol, *Analytical Benchmarks for Nuclear Engineering Applications Case Studies in Neutron Transport Theory*, OECD NEA Publishing, Paris (2009).

[21] "Guide ASN 28: Qualification des outils de calcul scientifique utilisés dans la démonstration de sûreté nucléaire - première barrière," , Autorité de Sûreté Nucléaire (2017).

[22] W. Royce, "Managing the Development of Large Software Systems," *IEEE WESCON*, vol. 26, 328–388, 1-9. Los Alamitos, CA, USA (1970).

[23] "Systems and software engineering - Life cycle processes - Requirements engineering," , ISO/IEC/IEEE 29148:2011 (2011).

[24] R. Pressman and B. Maxim, *Software Engineering: A Practitioner's Approach*, McGraw Hill (2001).

[25] A. Hebért, "DRAGON5: Designing computational schemes dedicated to fission nuclear reactors for space," *Nuclear and Emerging Technologies for Space (NETS 2013)*, 368–375, Albuquerque, New Mexico, USA, 25-28 February 2013 (2013).

[26] R. Sanchez, J. Mondot, u. Stankovski, A. Cossic, and I. Zmijarevic, "APOLLO II: A User-Oriented, Portable, Modular Code for Multigroup Transport Assembly Calculations," *Nuclear Science and Engineering*, **100**, *3*, 352 (1988); 10.13182/NSE88-3.

[27] E. Martinolli, T. C. Carter, F. Clement, P. M. Demy, M. Leclere, P. Magat, A. Marquis, V. Marotte, M. Schneider, L. Villatte, J. Marten, S. Misu, and S. Thareau, "APOLLO2-A - AREVA's new generation lattice physics code: Methodology and validation," *Advances in Reactor physics to Power the Nuclear Renaissance (PHYSOR 2010)*, Pittsburgh, PA, USA (2010).

[28] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy Approach to Requirements Syntax (EARS)," *2009 17th IEEE International Requirements Engineering Conference*, 317–322, Atlanta, Georgia, USA, 31 August 2009 - 04 September 2009 (2009); 10.1109/RE.2009.9.

[29] A. Brighenti, A. Hebert, B. Calgaro, E.-Y. Garcia-Cervantes, G. Huaccho Zavala, P. Laurent, L. Mercatali, P. Mosca, A. Previti, S. Santandrea, B. Vezzoni, J.-F. Vidal, A. Willien, and I. Zmijarevic, "Development of a multi-parameter library generator prototype for VVER and PWR applications based on APOLLO3," *The International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2023)*, Niagara Falls, Ontario, Canada, 13-17 August 2023 (2023); (submitted).

[30] J.-F. Vidal, O. Litaize, D. Bernard, A. Santamarina, C. Vaglio-Gaudard, and T. N. Tran, "New modelling of LWR assemblies using the APOLLO2 code package," *Joint International Topical Meeting on Mathematics & Computation and Supercomputing in Nuclear Applications (M&C + SNA 2007)*, Monterey, California, USA, 15-19 April 2007 (2007).

[31] A. Hébert, "A Consistent Technique for the Pin-by-Pin Homogenization of a Pressurized Water Reactor Assembly," *Nuclear Science and Engineering*, **113**, *3*, 227 (1993); 10.13182/NSE92-10.

[32] S. Marguet, *The Physics of Nuclear Reactors*, Springer International Publishing (2018); 10.1007/978-3-319-59560-3.

[33] T. Clerc, A. Hébert, H. Leroyer, J. P. Argaud, B. Bouriquet, and A. Ponçot, "An advanced computational scheme for the optimization of 2D radial reflector calculations in pressurized water reactors," *Nuclear Engineering and Design*, **273**, 560 (2014); 10.1016/j.nucengdes.2014.03.044.

[34] S. Santandrea and R. Sanchez, "Analysis and improvements of the DPN acceleration technique for the method of characteristics in unstructured meshes," *Annals of Nuclear Energy*, **32**, 163 (2005); 10.1016/j.anucene.2004.08.011.

[35] K. Smith and J. Rhodes, "CASMO characteristics methods for two-dimensional PWR and BWR core calculations," *Transactions American Nuclear Society*, **83**, 294 (2000).

[36] Commissariat à l'énergie atomique et aux énergies alternatives - Direction de l'énergie nucléaire (Editor), *La neutronique*, Éditions du Moniteur (2015).

[37] J. R. Askew, "A characteristics formulation of the neutron transport equation in complicated geometries," , UK Atomic Energy Establishment: AAEW-M 1108 (1972).

[38] R. Sanchez, L. Mao, and S. Santandrea, "Treatment of Boundary Conditions in Trajectory-Based Deterministic Transport Methods," *Nuclear Science and Engineering*, **140**, 23 (2002); 10.13182/NSE140-23.

[39] A. Yamamoto, T. Ikehara, T. Ito, and E. Saji, "Benchmark Problem Suite for Reactor Physics Study of LWR Next Generation Fuels," *Journal of Nuclear Science and Technology*, **39**, *8*, 900 (2002); 10.1080/18811248.2002.9715275.

[40] A. Godfrey, "VERA Core Physics Benchmark Progression Problem Specifications," , CASL Technical Report: CASL-U-2012-0131-002 (2013).

[41] N. Hfaiedh and A. Santamarina, "Determination of the optimized SHEM mesh for neutron transport calculation," *International Conference on Mathematics and Computation M&C 2005*, Avignon, France, American Nuclear Society (2005).

[42] "Hierarchical Data Format 5," , https://www.hdfgroup.org/hdf5/.

[43] S. Santandrea, "Personal communication," .

[44] S. Chandrasekhar, *Radiative Transfer*, Dover (1960).

[45] B. G. Carlson and G. I. Bell, "Solution of the transport equation by the Sn method," *Second international conference on the peaceful uses of atomic energy*, vol. 16, 535 (1958).

[46] G. I. Bell and S. Glasstone, *Nuclear Reactor Theory*, R. E. Krieger Publishing Company (1970).

[47] R. Sanchez and N. McCormick, "A Review of Neutron Transport Approximations," *Nuclear Science and Engineering*, **80**, *4*, 481 (1982); 10.13182/NSE80-04-481.

[48] F. Brown, "Recent Advances and Future Prospects for Monte Carlo," *Progress in Nuclear Science and Technology*, **2** (2011); 10.15669/pnst.2.1.