Journal of Nuclear Materials 406 (2010) 166–174

Contents lists available at ScienceDirect

Journal of Nuclear Materials

journal homepage: www.elsevier.com/locate/jnucmat

A not-so-short description of the PERFECT platform

S. Bugat *, A. Zeghadi, G. Adjanor

EDF Research and Development Division, Les Renardières Site, Route des Renardières, F77818 MORET-SUR-LOING Cedex, France

A B S T R A C T

This article describes the building of the so-called 'PERFECT platform', which main issue was to allow the development of the PERFECT end-products dedicated to the prediction of the degradation of material properties due to irradiation. First, the general principles used to build the platform are detailed. Such principles guided the choices of preferential development language, architecture, and operating system. The architecture of the platform is then described. It allows an easy development of the end-products, and a 'black-box' integration of the codes developed during the project. Each end-product can be seen as a sequence of modules, each module representing a physical phenomenon in time and space. The platform is very flexible, so that different methodologies can be tested and compared inside an end-product. The second part is devoted to the description of a classical PERFECT study, defined thanks to the graphical user interface developed in the project. Focus is made in particular on how a selection of modules is done, how the input data can be entered, and how the study execution is fully controlled by the user. A final description of the post-processing facilities on the results is exposed.

1. Introduction

The project PERFECT (Prediction of Irradiation Damage Effects in Reactor Components) of the 6th Framework Program aimed at developing and building predictive tools for Reactor Pressure Vessels and Internal structures. In particular, the project focused on the development of predictive tools for the fracture toughness temperature dependence of RPV steels, and also on attempts of development of predictive tools for irradiation hardening, plastic flow channelling, void swelling and IASCC sensitivity of Internals.

Such a multiscale modelling of the micromechanical behaviour of metals has already been attempted, as in Ghoniem et al. [5] for instance. It has also been applied to irradiation damage on nuclear materials for different reactor types, as for instance the British Magnox power stations [3]. The purpose of this paper is to have an integration process that will allow a possible successful future industrial exploitation of these multi-scale models.

The main objective of this integration was to build two 'Virtual Reactors' simulating the effect of irradiation respectively on Reactor Pressure Vessel Fracture Toughness and on Internal structure Irradiation Assisted Stress Corrosion Cracking. Specific modules and database – dealing with damage production, microstructure evolution, mechanical and corrosion phenomena induced by irradiation – complemented each of these Virtual Reactors. The resulting numerical tools were integrated in a Software Integration Platform.

To reach its objectives, PERFECT was organised into four technical sub-projects as follows:

- SP I: Integration
  The main activities of this sub-project were the numerical coupling of the calculation modules, provided by the sub-projects II, III and IV listed below. The results of this coupling were realized on the Software Integration Platform developed by EDF with support from CEA, which provided a common computer architecture.
- SP II: Physics modelling of RPV and Internals, to predict the radiation-induced microstructure upon physical basis.
- SP III: Mechanics modelling of the RPV, to predict the fracture behaviour of Reactor Pressure Vessel components as a function of material type and characteristics, loading and irradiation conditions.
- SP IV: Mechanics and corrosion modelling of RPV Internals, to formulate predictive models for initiation and propagation of radiation-induced stress corrosion cracks in irradiated stainless steels.

The relationship between the various sub-projects is shown in Fig. 1.

This paper will focus on the main achievements of SP I, that is, the software integration platform and the associated end-products. Those end-products were evaluated by the User's Group sub-project which refers to the last sub-project of PERFECT. The accuracy

* Corresponding author.
E-mail addresses: stephane.bugat@edf.fr (S. Bugat), asmahana.zeghadi@edf.fr (A. Zeghadi), gilles.adjanor@edf.fr (G. Adjanor).

0022-3115/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.jnucmat.2009.04.022

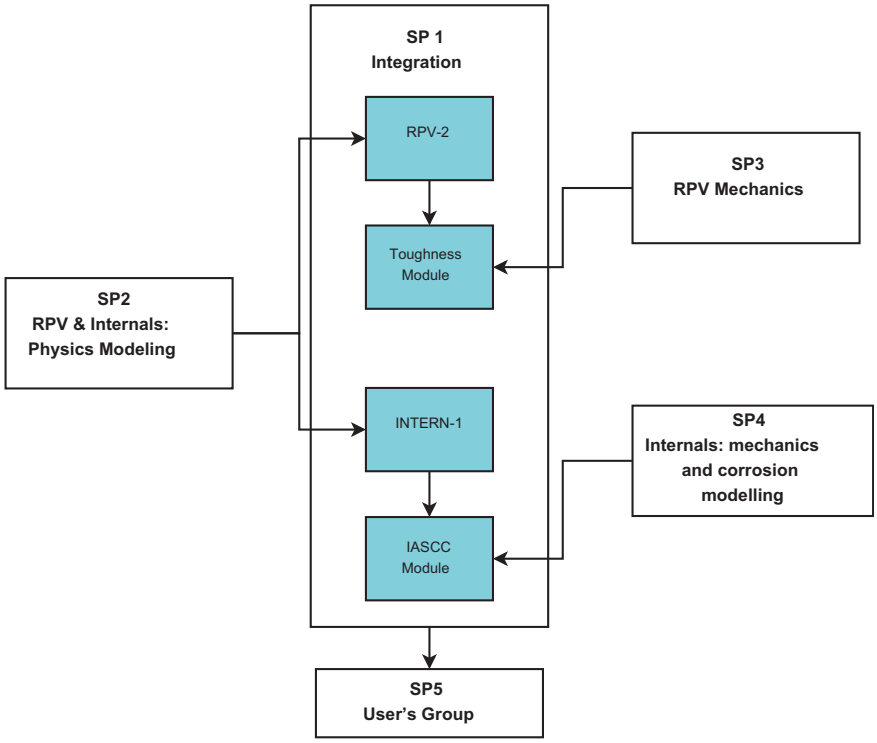

Fig. 1. Organisation of the PERFECT project. The Integration sub-project includes all outputs from SP-II, SP-III and SP-IV to build the required end-products. Those end-products are also used by the User's Group members.

of the end-products with respect to industrial references cases will not be presented in this paper.

2. Description of the platform

2.1. Some historical notes

The history of the platform is older than the PERFECT project. During the REVE initiative Jumel et al. [7], Malerba et al. [10], Jumel and Van-Duysen [8], a first prototype of so-called 'Virtual Test Reactor' (VTR) was developed: RPV-1. This VTR was already based on a chaining of several modules, to predict the increase of critical resolved shear stress due to irradiation.

However, each module had to be improved especially from the point of view of the parametrisation of the different codes used. Moreover, the architecture of the VTR was not very flexible, so that only one chaining was allowed, and the design and implementation of new modules was very complex. Finally, many different languages (shell scripts, C scripts, python scripts) were used to define the chaining so that the maintenance, portability and development of the platform remained complicated.

2.2. A few words about the end-products

The description of the end-products is the following:

(1) RPV-2 is a product that aims to predict the irradiated microstructure of RPV steels, as well as their microstructural hardening due to irradiation. The input data for RPV-2 are typically the chemical composition of the steel, the neutron spectrum, the irradiation temperature and the irradiation time.
The specifications of RPV-2 were built with support from SP-II 'Physics Modelling'.

(2) ToughnessModule is a product that uses the outputs from RPV-2, that is mainly the microstructural hardening, to predict both the macroscopic behaviour of the irradiated steel, and the subsequent decrease of fracture toughness. Some information on the metallurgy of the steel are also requested for the good operating of this end-product (microstructural morphology, texture, bainitic lath size, carbide size distribution for instance).
Specifications of the ToughnessModule were built commonly with SP-III 'RPV Mechanics'.

(3) INTERN-1 is a similar product to RPV-2, except that it deals with austenitic stainless steels of the Internals.

(4) IASCC Module was developed by the SP-IV as a standalone product. For the moment, it is not yet fully integrated into the software integration platform. It aims to predict both the local chemistry at crack tip, and a crack extension prediction based on the slip dissolution model.

Each end-product is based on a chaining of one or more 'modules', each module having a specific physical role. For instance, in RPV-2, the first module IRRAD aims to transform the neutron spectrum into a PKA[1] spectrum.

Each module, according to its complexity, can use one or more codes. Each code is generally specific to the targeted physical phenomenon: the IRRAD module uses for instance a simplified version of the SPECTER [6] code. Those codes are developed generally in different frameworks, and by different entities, not necessarily involved in the project. Their sources usually cannot be modified: their integration had to be considered as a 'black-box' integration, not a full integration.

Moreover, to get the same physical output, different methodologies can be used. For example, the prediction of the long term irradiated microstructure of RPV steels can be performed either by using rate-kinetic type models, or by Object Kinetic Monte-Carlo methods. As one aim of PERFECT was also to compare both pre-

[1] Primary Knocked-on Atom.

dictions, such models had to be implemented in the platform, so that the user can chose one or the other.

Finally, the platform should allow any user to implement new methodologies. To do so, the development of a new module should be eased in order to encourage the implementation of any model, with a minimum support from the SP-I. The new modules have also to be easily documented, so that in particular this documentation can be maintained together with the module itself. This prevents the developer from having to write the documentation after the module was implemented.

2.3. General principles

The requirements described above defined some key guidelines for the definition of the end-products, and the subsequent software architecture:

- A consistent, flexible and upgradable format for the definition of the input and output data:
  - for the exchange of data between the different chained modules of a given end-product;
  - for the use of different modules having the same role (named *branches*);
  - for an easier definition of new modules or branches.
- A consistent system of scripts for the chaining of the modules:
  - a preferential language for adaptation of the inputs/outputs to the different codes;
  - a unified and simple format for definition of the modules, avoiding any re-compiling of the platform.

In addition to those initial constraints about the end-products, another constraint was imposed by the type of codes that would be used on the platform. As the majority of these codes were developed on POSIX compliant architectures, like Unix or GNU/Linux systems, the PERFECT platform should be available at least for this platform. Moreover, a strong need was identified for the ability to launch studies either in batch mode, or with a graphical user interface. As the latter is fully comprehensive especially from the point of view of end-users, the former is related to the ability to launch parametric studies for instance, in a more innovative way of use dedicated to research studies.

Those constraints lead to the following choices for the architecture of the platform:

- A unique language for the definition of input/output data, the structure of the end-products, the graphical user interface, and the definition of the studies: `python`.[2]
  This choice allows a very fast prototyping and a flexible development due to the possibilities of this language, which is also based on a huge library of modules [9]. `python` is also available on almost all operating systems, enhancing therefore the possible portability of the platform.
- A complete partition between the 'software environment' (data model, graphical user interfaces, documentation, etc.) and the 'dynamic part' (the end-products themselves).
  This kind of design is usually called as *orthogonal*: *orthogonality guarantees that modifying the technical effect produced by a component of a system neither creates nor propagates side effects to other components of the system.*[3]
- A *data model* described by specific python classes
  The use of OOP[4] allows an easy definition and extension of input and output data. Some basic types are first defined. They correspond to files, directories, floats, integers, character strings, numerical tables and so on. Then more structured classes are built by recursive integration of those simple classes.
- A description of the end-products and their sub-modules as simple *python modules*This choice allows to have the same architecture between the end-products and the platform. Therefore the development and extension of existing modules is made easy.
- Some *accessibility levels* are defined for all data: 3 levels have been defined, according to the current knowledge of the user (`normal`, `expert`, or `non-modifiable`). Those levels allow the experimented users to finely tune some default parameters in the modules.
- The graphical user interface uses the *qt3* graphical library, so as to be consistent with the SALOME platform [12] in which the PERFECT end-products can be included.

For what concerns the documentation of the end-products and data, the choice has been made to use the LaTeX formatting language. This choice is consistent with the large dissemination and use of this typesetting language in the scientific community. To ease this documenting process, each module of an end-product and each class of data embeds its own documentation as a string attribute. The full documentation of each end-product or structured class is then built by recursive concatenation of the documentation attributes of their sub-modules or objects. To generate the documentation in a WYSIWYG.[5] format, some drivers are defined to export them in PDF or HTML formats. The softwares `dvips`, `ps2pdf` and `latex2html`, freely available on GNU/Linux, are used.

More generally, the platform was built with respect to a certain number of coding standards of the open-source community. This ensures that the development and maintenance will be eased in the future, and that the platform will use only open-source softwares, with the exception of the scientific codes themselves of course.

2.4. Structure of the platform

The structure of the platform is described on Fig. 2. Not all the sub-directories are represented, but only those of interest from a macroscopic point of view. Each folder represents a python package or module.

The acronyms used for the different folder names refer to their function in the platform:

- `PDM` (*python data model*):
  The data model contains the declarations of all the python classes needed to build the input and output objects of all modules.
- `PSM` (*perfect study manager*):
  This python package contains the 'patterns' for the definition of modules inside the end-products, and the definition of PERFECT studies.
- `PMM` (*perfect modules model*):
  Inside this package are located all the end-products, gathered by type of material (Reactor Pressure Vessel Steel or Internals). Their structure is consistent with the structure of the end-products.
- `Perspycace` (*perfect graphical user interface*):
  This package is related to the implementation of the graphical user interface (GUI) aiming to define, follow and process PERFECT studies. It uses the `python-qt` graphical library.

[2] See http://en.wikipedia.org/wiki/Python_%28programming_language%29.

[3] See http://en.wikipedia.org/wiki/Orthogonal#Computer_science for a detailed description of this concept.

[4] Object-Oriented Programming, see http://en.wikipedia.org/wiki/Object_oriented.

[5] What You See is What You Get.

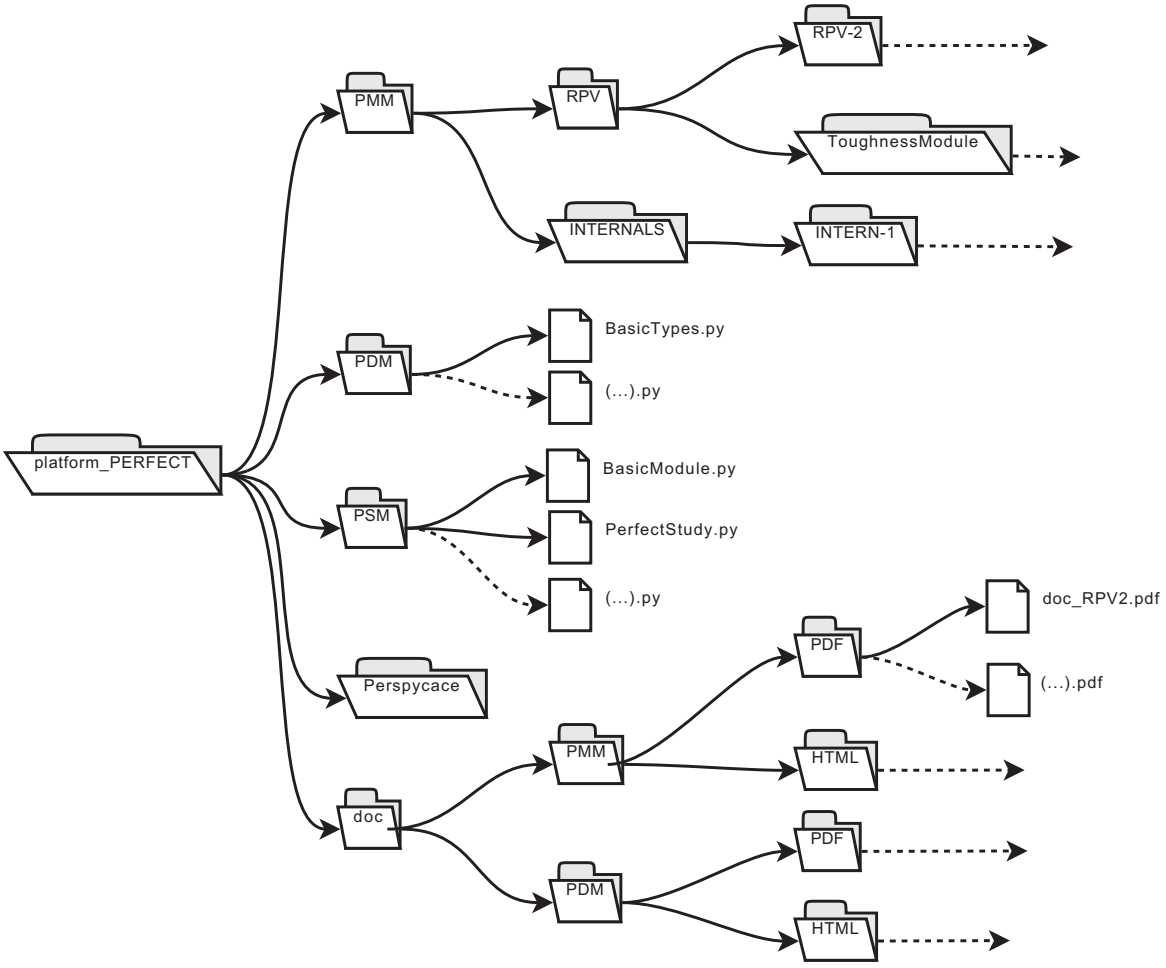

Fig. 2. Schematic structure of the software integration platform.

- `doc` (*documentation*):
  In this folder are located all the documentations for the input/output data of the different modules, and for the end-products and their sub-modules. These documentations are available in PDF[6] or HTML[7] formats.

Each package is briefly described hereafter.

2.4.1. The PDM package

As mentioned above, this package details the basic types needed to build more complex types of data for input/output of the modules. It refers to a *static* description of the data needed. Each type of data is a python class. Each python class has different attributes, however some of them are common to almost all basic classes:

- `value`: when the class refers to a numerical float, integer or string, this attribute gives access to its current value;
- `validity_range`: when the class refers to a numerical value, this attribute declares the range of validity of this value;
- `unit`: for classes related to numerical values, this argument indicates in which unit the value is expressed. Some conversion facilities between units are provided.
- `access_level`: this attribute indicates if a user with a given profile (normal or expert) can modify the value of the class;
- `long_doc`: this attribute contains a LaTeX string describing the role of this class.

If we take the example of the Young's modulus for the RPV steel: it is a float, whose default value and unit are fixed at 200 000 MPa, whose range of validity is set to [150 000, $+\infty$[, whose access level is `expert` (meaning that the default value is sufficient for most studies), and whose documentation is `"Young's modulus of the RPV steel"`.

Structured classes are built from those basic types as trees, where each node is an attribute instance of a structured class, and each leaf is a simple type. If we take as an example the constitutive equations for the description of a single-crystal behaviour, namely the Cailletaud-Méric law [11], it has been implemented through the class `SingleCrystalCailletaud` chosen in the ToughnessModule. Its structure is the following:

```
[SingleCrystalCailletaud]
|—+ NortonKinematicFlow [NortonKinematicFlow]
| |—+ C [Coefficient] =': 0.0 MPa'
| |—+ K [Coefficient] =': 10. MPa.s$ ^{1/n'}
| |—+ n [Coefficient] =': 20.'
|
|—+ LHM [LHM]
| |—+ h2 [Coefficient] =': 1.0'
| |—+ h3 [Coefficient] =': 1.0'
| |—+ h1 [Coefficient] =': 1.0'
| |—+ h4 [Coefficient] =': 1.0'
|
```

[6] See http://en.wikipedia.org/wiki/Portable_Document_Format for details.
[7] See <http://en.wikipedia.org/wiki/HTML>.

```
|—+ Elasticity [Elasticity]
| |—+ young [Stress] ='Sigma: 200 000 MPa'
| |—+ poisson [Coefficient] =': 0.3'
| |—+ alpha [inversetemperature] ='T^-1: 0.0 K^-1'
|
|—+ IsotropicHardening [IsotropicHardening]
| |—+ Q [Stress] ='Sigma: 20.0 MPa'
| |—+ b [Coefficient] =': 10.0'
| |—+ tau0 [Stress] ='tau0: 123. MPa'
|
|—+ KinematicHardening [KinematicHardening]
| |—+ D [Coefficient] =': 0.0'
```

Such a structuring allows to set that a full behaviour law is constituted by the setting of a viscoplastic flow (here the Norton kinematic model), a latent hardening matrix (defined here by four coefficients h_1 to h_4), the elastic properties (here considered as isotropic and defined by the Young's modulus and the Poisson coefficient) and two hardenings (one non-linear isotropic and one non-linear kinematic).

2.4.2. The PSM package

This package is the *engine* of the PERFECT studies. It holds two important python modules: the first one, `BasicModule.py`, describes how an end-product module has to be constituted. As mentioned above, each module in an end-product corresponds to a python module, described in the `PMM` directory (see Section 2.4.3). However, there is a common structure for all modules, and this structure is described here. To be more precise, a module should contain the following elements:

- a list of input and output data, referenced by their names and their corresponding classes in `PDM`;
- a list of template files, describing the structure of the input files needed for the module to operate (without any numerical value at this point);
- a list of sub-modules, if there is some, or a declaration of module to be run if not;
- a documentation, given as a LaTeX string.

The classes described in the package `family BasicModule` also contains methods to access to the different elements of a module, for instance its parent, its corresponding end-product, its documentation and so on.

The second important python package delivered inside `PSM` is the `family PerfectStudy` package. It defines the structure of a study. The main attributes of a study are: the so-called *chain of modules*, corresponding to the sequence of sub-module of the selected end-product to be run, and a list of *input data*, corresponding to the numerical data inputed by the user and needed by the chain of modules. Those input data depends effectively on the selected chain of modules. As some outputs of a given module can be inputs for the following modules, a common system of identification names is defined. The list of inputs is thus a dictionary from the software point of view, where the keys define the unique names of the data, and the values corresponds to the instance of the class. This dictionary is enriched by the outputs of the different modules when they are ran. The final dictionary contains all input and output data, referenced by their names.

2.4.3. The PMM package

The `PMM` package contains the python modules referring to the different modules of the end-products. Inside a given end-product, the modules and their possible sub-modules can have two different kind of relationships: either they are *chained*, or they are *exclusive*. Two modules are chained when they can be run one after the other, in that sense that the second module will use as input data some of the output data of the first module. Two modules are exclusive when they aim to provide the same output data, by using two different physical methodologies, so that the user has to chose one or the other and cannot chain them.

For instances, in the FractureBehaviour module of the ToughnessModule end-product, there is a sub-module named LocalApproach aiming to predict the failure probability of a 1T-CT specimen versus loading by applying a local approach post-processor like the Beremin model [2]. For that, a first sub-module CTCalculation performs the computation in 2D-plain strain of the CT specimen. Then, a second sub-module PostProcessor uses the computed strain and stress fields to perform a Beremin post-processor from this first computation results. Those two modules are thus chained.

On the other side, in the HARD module of RPV-2, two different methodologies can be used to estimate the microstructural hardening due to irradiation: a first sub-module, called OrowanBacon, uses the Bacon analytical model [1] to provide the increase of critical resolved shear stress. A second sub-module, DUPAIR, uses a simplified dislocation code combined with a Foreman and Makin model [4] to simulate the unpinning of dislocations around obsta-

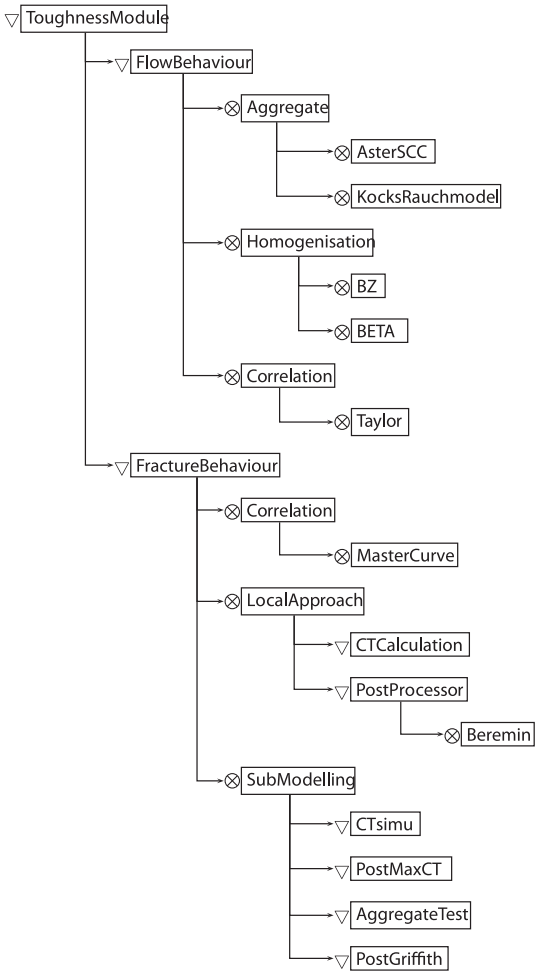

Fig. 3. Treeview of the ToughnessModule.

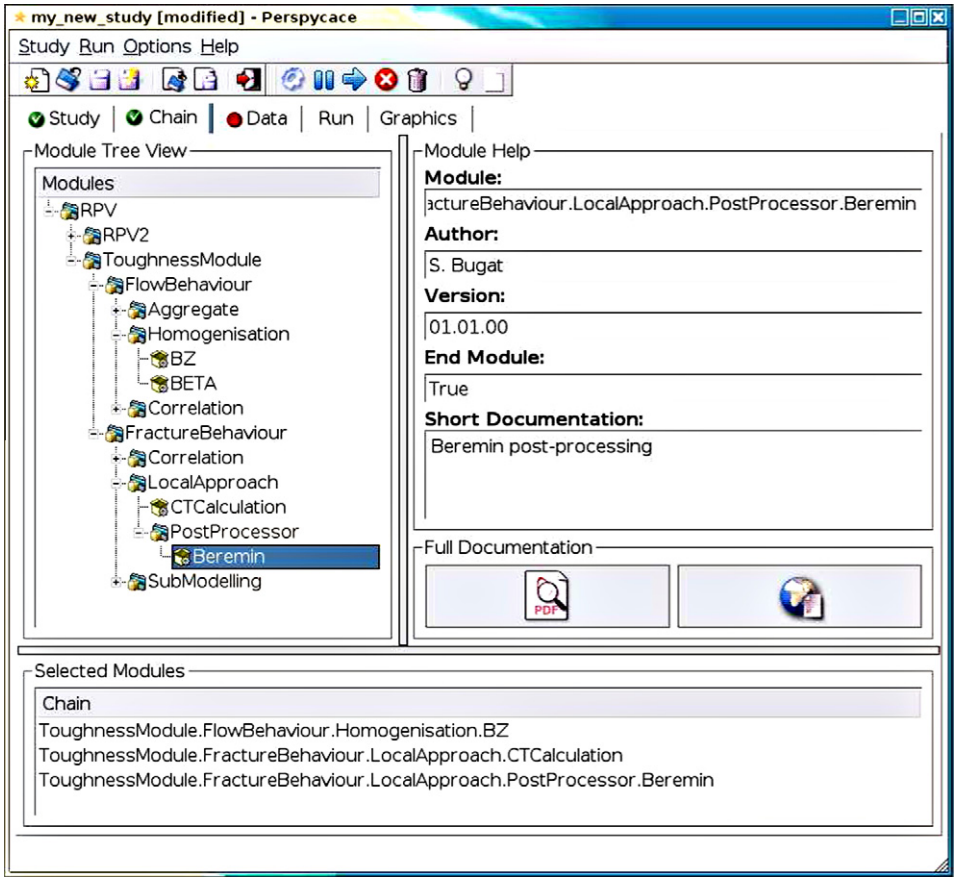

Fig. 4. Screenshot of the perspycace GUI for the definition of a list of chained modules (page *Chain* of the notebook).

cles, and also to provide the increase of CRSS. Both sub-modules are exclusive and cannot be chained.

Another specificity of the modules is that they can be simple 'containers' for other modules, or what is called 'end-modules', that is module that really perform a physical computation. In the above example, HARD is a container for the two end-modules OrowanBacon and DUPAIR. To describe a more complex structure of end-product, the Fig. 3 details the different modules and sub-modules of the ToughnessModule.

2.4.4. Other important packages

The `doc/` folder contains all the documentation, in PDF or HTML formats, for both the end-products described in `PMM/`, but also for all classes defined in `PDM/`. The documentation of the use of the graphical user interface perspycace is also included, as well as some documents concerning the different codes used in the platform. The generation of the documentations of modules and classes uses specific python drivers that extracts their LATEX documentation strings (as well as for their sub-modules or attributes) and recursively builds a complete LATEX document that is compiled and converted into PDF and HTML using `dvips`, `ps2pdf` and `latex2html`. As an example, the diagram of the Fig. 3 was generated with this process.

There is also a developer documentation that is located in the `apidoc/` folder. This documentation is generated thanks to the `epydoc` utility[8] from the docstrings of the functions, classes and methods.

The `Perspycace` python package contains the definition of the GUI components. `Perspycace` stands for *Perfect Study interface based on Python for Cascade Computation of end-products*, but is also an acronym of *clear-sighted* in French. The GUI is based on pyth-on-qt, and was designed with the help of the qt-designer[9] software. The key point is that this GUI is implemented separately from the 'engine' that allows to define and launch PERFECT studies, described in Section 2.4.2. This non adhesion allows also to define easily other graphical user interface if wanted.

All compiled binaries of the codes used in the platform are located in the `tools/` directory. For the moment, only validated versions of these codes are placed here. The binaries are available for 32-bits platforms.

3. Description of a PERFECT study

This section will briefly describe the functionalities of the platform from a user's point of view, and how it can be employed to define, launch, follow and process a study. As mentioned previously, this can be done either in 'batch mode' (via a python script) or via the perspycace interface. We will focus on the use of the latest.

3.1. Defining the study

A PERFECT study is correctly defined by the following elements:

- a set of information regarding the user (name), the date, the description of the study;
- a chain of modules selected from a given end-product;

[8] <http://epydoc.sourceforge.net/>.

[9] http://trolltech.com/products/qt/features/tools/designer.

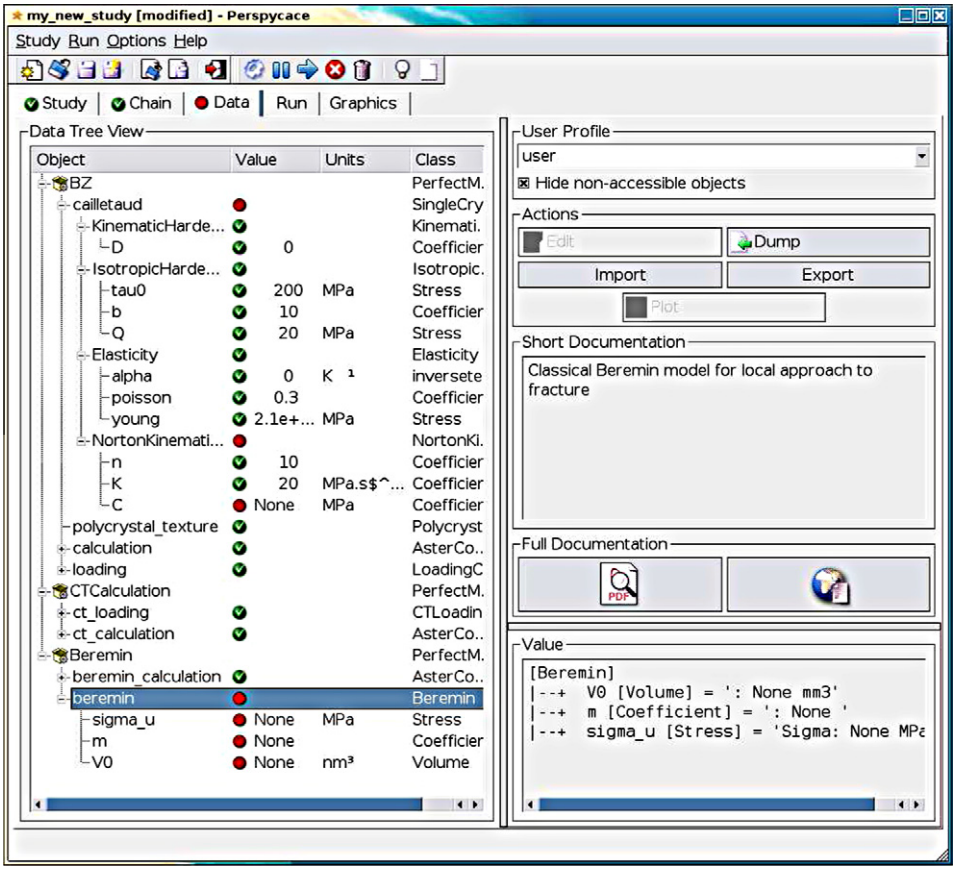

Fig. 5. Screenshot of the *Data* page of the notebook. The input data needed for the selected modules can be modified by the user. One can recognise the structure of the SingleCrystalCailletaud class given in Section 2.4.1.

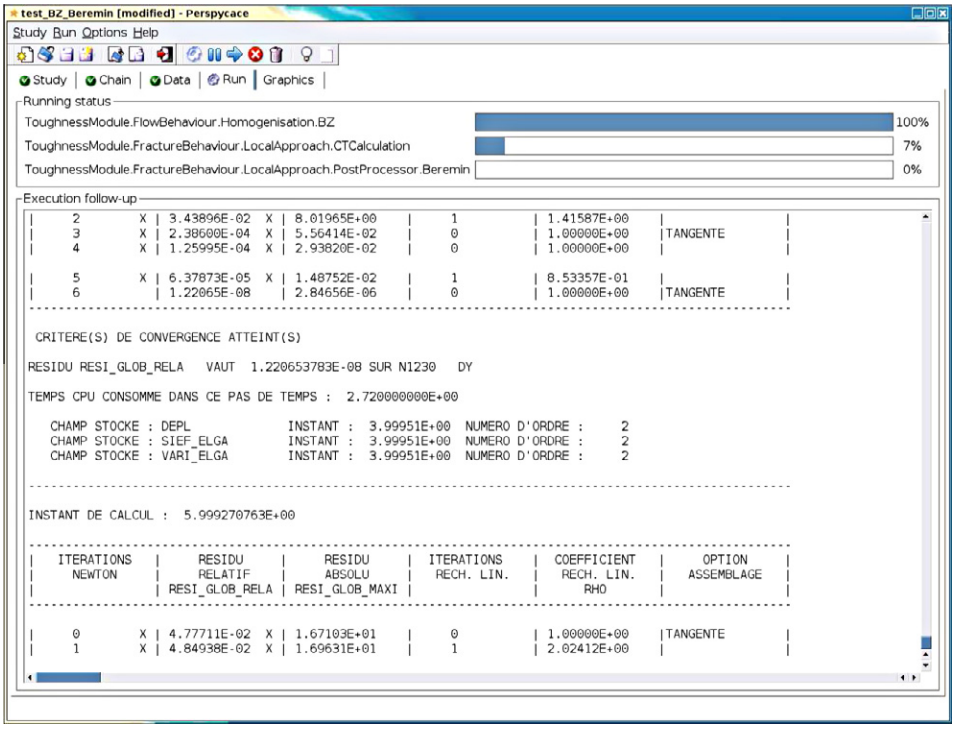

Fig. 6. Follow-up of the execution of a PERFECT study. For each module a progress bar displays the current state of advance of the computation. A text frame also displays the output of execution of the current running module.

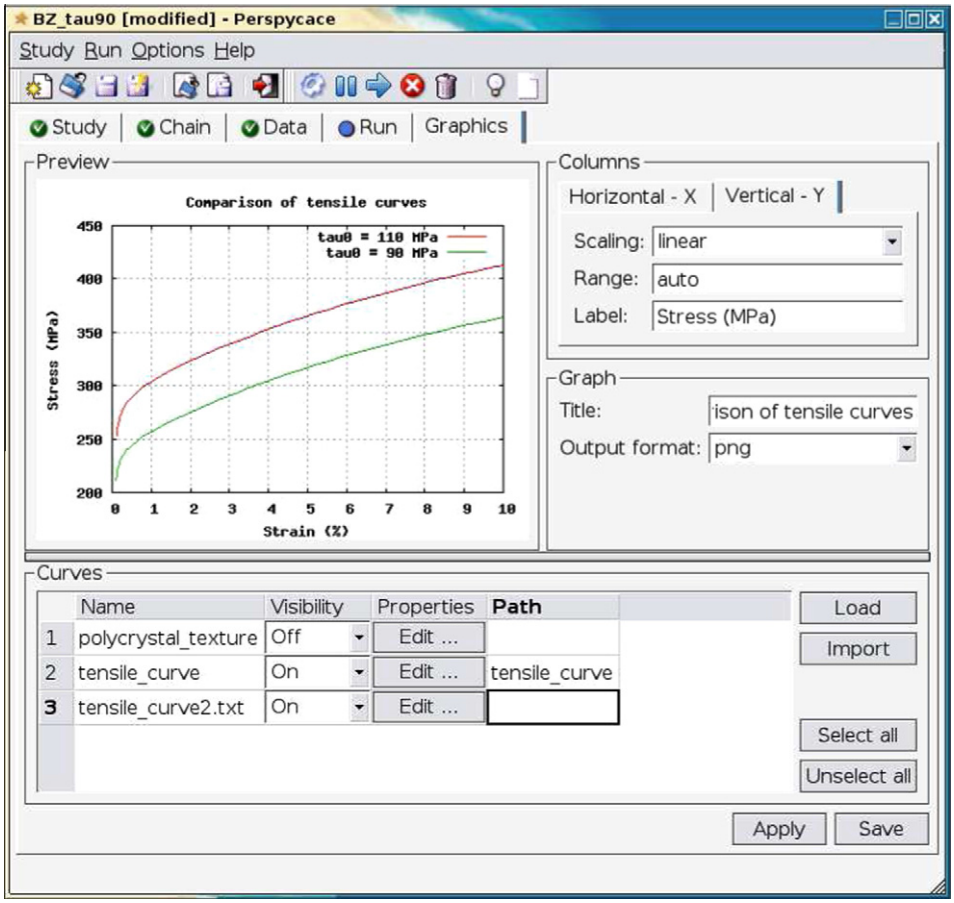

Fig. 7. Example of curves plotting within perspycace. The gnuplot software is used to generate plots of the output tables produced in the study. Experimental curves can be plotted as well.

- a collection of input data, depending on the selected modules, and the accessibility level of the user.

An example of selection of a chain of modules is given on Fig. 4. The chosen modules are part of the ToughnessModule. The full 'treeview' of the end-products is available on the left frame. Clicking on a given module gives access to a short documentation about the module, its author, its version, and so on. Moreover, the full module documentation in PDF or HTML format can be obtained with the respective buttons on the right frame.

The selected modules are listed in the bottom frame. An automatic recognition of the compatibility of the modules is done, so that only chain-able modules can be selected. Any selected module can be removed from the chain by simply double-clicking on it. Once a chain of modules is correctly define, it can be saved in a file for further use in another study.

The dictionary of input data is presented on the Data page of the notebook of perspycace (see screenshot of Fig. 5). The whole structure of each data is presented on the treeview and updated thanks to the current list of selected modules. Only data corresponding to the access level of the user, defined on the right panel with the *User profile* choice, are visible and modifiable.

The current value of the data as well as its unit when available are also listed in the treeview. Editing any data allows the user to modify its default value. It is worthwile to notice that mathematical expressions can be used to define a value as well. Once the value is correctly defined and is consistent with the validity range attribute of the object, the status of the object becomes valid (green bullet instead of red bullet in the *Value* column).

Numerical tables can also be parsed from text files, as some import facilities are available. On the other side, those tables can also be exported into ASCII files with a space-type field separation or in the csv[10] format. This is convenient to define tables from experimental results in particular.

Documentation in PDF and HTML formats for each input data can be browsed by clicking on the appropriate button on the right panel. Selecting an object also displays a small help in the *Short documentation* frame.

The object can also be exported to some specific files in order to be used furtherly in other studies (*export* button). They can be imported using the *import* button as well. The format used to store these objects is based on the XML syntax.

Once all values are correctly defined, the Data page of the notebook displays a green bullet instead of a red one, and the user is able to launch the study. At this stage, the study can also be exported as a python script via an option in the *Study* menu. The file generated can thus be modified and launched by the experimented user who, of course, should also have a basic knowledge of the python language.

3.2. Launch and follow-up of the study

The launch and follow-up of the execution of the study is available through the *Run* page of the notebook of perspycace. The execution of the study is fully controlled by the user through the

[10] refer to <http://en.wikipedia.org/wiki/Comma-separated_values>.

menubar of the application: a study can be launched, interrupted, resumed and stopped when desired.

Once the study is launched (see screenshot of Fig. 6), the chain of selected modules and the input data are no more modifiable. The only way to set a new value for a given input data is to stop and reinitialise the study. This parapet avoids any confusion on the initial parameters of the study. After launching, the execution of the current running module is available thanks to a progress bar and a text frame displaying its output messages.

In case of failure of execution of a given module, some troubleshooting tools are available: the so-called 'execution stack' can be displayed by clicking on the lamp button. It gives access, for each module, to the return code of execution, the standard output messages of execution, and the error output messages.

3.3. Post-processing of the results

Once the study is finished, all output data provided by the modules that correctly operated are available in the Data page of the notebook. They can be displayed to the user's view, or exported to text files if they represent numerical tables. These tables can also be plotted within perspycace, through the *Graphics* page of the notebook. Plots use the gnuplot[11] free software to generate PNG preview graphs. Although only 2D curves can be plotted, they can also be imported from text files, so that comparisons with experimental results are also allowed, as shown in Fig. 7.

At this stage, studies can be stored with their output results for comparison with other studies. A LATEX report can also be generated. It contains not only the input and output data, but also if wanted the execution stack and the documentation of the selected modules. This report can be processed with `latex` to get a PDF report of the study, ready to be printed.

4. Conclusions and outlooks

The PERFECT platform was developed during the 6th Framework Program project PERFECT. It aims at integrating different end-products, dedicated to the prediction of irradiation effects on RPV and Internals steels, into a common software environment. Each end-product can be seen as a sequence of modules, chain-able or exclusives, aiming to solve a specific physical phenomenon at one scale. The modules involve different codes developed in the relative scientific community.

The choices made to build the software environment allowed to have a consistent and upgradable platform. python was selected as the preferred language for the development. More generally, *Open Source* software and standards were used to enhance the portability of the platform and to ease its enrichment, especially for scientists not necessarily familiar with software development.

The choice of an orthogonal conception also allows to use the platform either in batch mode, or through a graphical user interface. The architecture of the platform is consistent with this choice. The different packages can be developed apart from one another. The documentation of the end-products and the input/ output data is also automatically generated, as each object embeds its own documentation.

A study corresponds to the definition of a sequence of chainable modules, and a list of input data necessary for the modules to operate. Once all data are correctly filled by the user, the study can be launched and fully controlled thanks to the graphical user interface. If a failure occurs during the execution, some debugging facilities are available to identify the possible causes. The executed study can be stored with its output results. Output results can be also plotted and compared with experimental results or results coming from other PERFECT studies.

The PERFECT platform is distributed to the members of the PERFECT project thanks to a *Live-DVD*, containing a Debian[12] GNU/Linux distribution with all prerequisites, and the platform itself. This distribution mode allows any user having a computer with enough RAM to test the platform without installing anything on its hard disk. However, for better performance of the platform it should be installed on a compatible Linux distribution. As all the prerequisites required by the platform are available on every distributions, this does not represent a strong difficulty.

Further development of the platform will be focused essentially on the post- processing facilities. The latest version of the platform already contains an optimiser based on the Nelder-Mead simplex algorithm[13] which allows to identify specific input data on some given results, in batch mode. A specific graphical interface was also developed for this purpose.

Moreover, a strong interest for the use of the platform with regard to parametric studies, sensitivity analyses and confidence analyses has been identified. The development of such extensions to the platform will be treated in the follow-up project of PERFECT, but needs strong skills in probabilistic approaches.

Finally, the PERFECT platform could also benefit from the post-processing facilities already available in the SALOME platform, in particular for the visualisation of the results of mechanical computations (stress and strain fields), or for curve plotting instead of gnuplot.

References

[1] D.J. Bacon, U.F. Kocks, R.O. Scattergood, Philos. Mag. 28 (6) (1973) 1241.
[2] F. Beremin, Metall. Trans. A (Phys. Metall. Mater. Sci.) 14A (11) (1983) 2277.
[3] P. Flewitt, Mater. Sci. Eng. A (Struct. Mater.: Prop. Microstruct. Process.) A365 (1–2) (2004) 257. <http://dx.doi.org/10.1016/j.msea.2003.09.084>.
[4] A. Foreman, M. Makin, Philos. Mag. 13 (1966) 911.
[5] N.M. Ghoniem, E.P. Busso, N. Kioussis, H. Huang, Philos. Mag. 83 (31–34) (2003) 3475. <http://dx.doi.org/10.1080/14786430310001607388>.
[6] L. Greenwood, R. Smither, SPECTER: Neutron Damage Calculations for Materials Irradiations, Argonne National Laboratory, Argonne, IL, ANL/FPP/TM-197, January 1985.
[7] S. Jumel, C. Domain, J. Ruste, J.-C. Van Duysen, C. Becquart, A. Legris, P. Pareige, A. Barbu, E. Van Walle, R. Chaouadi, M. Hou, G.R. Odette, R.E. Stoller, B.D. Wirth, J. Test. Eval. 30 (1) (2002) 37.
[8] S. Jumel, J.C. Van-Duysen, J. Nucl. Mater. 340 (2–3) (2005) 125. <http://dx.doi.org/10.1016/j.jnucmat.2004.10.131>.
[9] G. Lindstrom, IT Prof. 7 (5) (2005) 10.
[10] L. Malerba, E. van Walle, C. Domain, S. Jumel, J.-C. Van Duysen, State of Advancement of the International Reve Project: Computational Modelling of Irradiation-induced Hardening in Reactor Pressure Vessel Steels and Relevant Experimental Validation Programme, Arlington, VA, USA, 2002, p. 8.
[11] D. Nouailhas, J.-P. Culie, G. Cailletaud, L. Meric, Eur. J. Mech., A/Solids 14 (1) (1995) 137.
[12] A. Ribes, C. Caremoli, Salome Platform Component Model for Numerical Simulation, vol. 2, Beijing, China, 2007, p. 553. <http://dx.doi.org/10.1109/COMPSAC.2007.185>.

[11] <http://www.gnuplot.org>.

[12] <http://www.debian.org>.

[13] see http://pylab.sourceforge.net/packages/optimize.py.