International Conference on Mathematics, Computational Methods & Reactor Physics (M&C 2009)
Saratoga Springs, New York, May 3-7, 2009, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2009)

FLAIR: A POWERFUL BUT USER FRIENDLY GRAPHICAL INTERFACE FOR FLUKA

Vlachoudis V
CERN
Dep. EN, Geneva-23
CH-1211 Switzerland
Vasilis.Vlachoudis@cern.ch

ABSTRACT

FLAIR is an advanced user graphical interface for FLUKA, to enable the user to start and control FLUKA jobs completely from a GUI environment without the need for command-line interactions. It is written entirely with python and Tkinter allowing easier portability across various operating systems and great programming flexibility with focus to be used as an Application Programming Interface (API) for FLUKA. FLAIR is an integrated development environment (IDE) for FLUKA, it does not only provide means for the post processing of the output but a big emphasis has been set on the creation and checking of error free input files. It contains a fully featured editor for editing the input files in a human readable way with syntax highlighting, without hiding the inner functionality of FLUKA from the users. It provides also means for building the executable, debugging the geometry, running the code, monitoring the status of one or many runs, inspection of the output files, post processing of the binary files (data merging) and interface to plotting utilities like gnuplot and PovRay for high quality plots or photo-realistic images. The program includes also a database of selected properties of all known nuclides and their known isotopic composition as well a reference database of ~300 predefined materials together with their Sterheimer parameters.

Key Words: Graphical User Interface, Simulation, FLUKA

1 INTRODUCTION

FLUKA [1,2,3] is a multipurpose transport Monte Carlo code, for calculations of particle transport and interactions with matter, covering an extended range of applications spanning from proton and electron accelerator shielding to target design, calorimetry, activation, dosimetry, detector design, Accelerator Driven Systems, cosmic rays, neutrino physics, radiotherapy etc.. FLUKA is able to transport 60 different elementary particles and whichever heavy ion, and can perform hadron-hadron, hadron-nucleus, neutrino, electromagnetic, and μ interactions from 1 keV for charged particles or thermal for neutrons up to 10000 TeV/n. It features a combinatorial geometry which was recently enhanced with the use of parenthesis expansions and geometrical optimizations. Although being an excellent tool for particle simulations problems, the default package offers limited number of auxiliary programs and tools for building the input and post processing the output of the code. It is a common practice that users are in need of creating ad hoc processing programs to exploit the features of FLUKA, which is often a source of errors especially for beginners.

The objective of the FLUKA Advanced Interface (FLAIR) [4] development was to inspect and overcome problems with input syntax and logical errors and in general facilitate the work of the user. FLAIR is trying to address all the above problems and more by providing an Integrated Development Environment (IDE) for all stages from building the input file and post processing the result. Emphasis has been set on the creation and checking of error free input files by flagging incorrect entries so that the user will be able to prevent the resulting logical error. Moreover it contains a fully featured editor for editing the input files in a human readable way with syntax highlighting and error checking, without hiding the inner functionality of FLUKA from the users. It provides means for building the executable, debugging the geometry, running the code, monitoring the status of one or many runs, inspecting the output files, post processing of the binary files (data merging) and interface to plotting utilities like gnuplot [5] and PovRay [6] for high quality plots or photo-realistic images. The program includes also a database of selected properties of all known nuclides and their known isotope properties as well a reference database of ~300 predefined materials together with their Sterheimer parameters [7]. Last but not least FLAIR is an open source project, featuring an Application Programming Interface (API) for manipulating FLUKA input files, as well all individual components of FLAIR can run as standalone applications, greatly enhancing productivity for expert users and supplying means for organizing everything as batch jobs. To ensure a high quality program and improve reliability FLAIR has passed all the software testing, verification phases and methods [8,9,10].

2 IMPLEMENTATION

User Interface (UI) is what allows the end users to interact with an application. A good UI will make an application intuitive and easy to use. Excellent applications without good UI could be less popular than inferior ones with a good UI. This is the case of FLUKA an excellent simulation tool but with no interface that renders its learning curve steep for beginners. FLAIR is trying to address the above problem, therefore during the creation of FLAIR we tried to respect the following concepts:

- Simple: even though it is meant as a user interface for a complex program like FLUKA, simplicity was a key factor. This was achieved by organizing in a step-by-step approach the various sub-tasks that the user will perform;
- Intuitive: everything is labeled and self explained. The interface is organized in a way that the user is not in a constant need of the manual. Pop up tip-dialogs show a message on what is expected from the user;
- Respect the commonly accepted conventions: There are several conventions that the greatest fraction of the new programs are using e.g. Keyboard conventions like Ctrl-C for copy etc., or tool bars for easier accessing of commands, right-clicking will pop up a sub menu with the most important commands at each stage;
- Visually organized: common functions and information are grouped together in a visually organized manner for easier accessing;
- Native look: follows the look and semantics of most modern applications;
- Easy installation procedure: through standard installation programs like rpm [11], and minimum dependence on external software;

- Extensible / Programmable: The use of iterative design, parallel development, and high modularity all enhance maintainability, ease of verification, and code reusability. The source code is written in an organized way following Object Oriented technologies. The full source code as well the auxiliary programs are supplied with the package and the user can use them for programmatically modify or create FLUKA input files and the related post processing;
- Not hiding the inner functionality of FLUKA: The user using the program has full control of all options/formats in FLUKA.

2.1 Programming Choices

FLAIR is build entirely using python [12] as programming language and Tkinter [13] for the graphical interface. Python is a well established interpreted interactive object-oriented language and portable almost on all operating systems. Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. Python is very rich in libraries for data processing and offers interfaces to the system, network, as well as to various windowing platforms (X11, Motif, Tk, Mac, MFC, wxWidgets). When executing FLAIR through python, the program is initially compiled and optimized in a pseudo-code assembly language and a virtual machine is interpreting this pseudo-code.

Among the various window interfaces the choice for FLAIR was to use Tkinter which is the default graphical user interface widget set for Python. Tkinter has limited possibilities in comparison with other graphical toolkits, nevertheless it was selected because of its availability on all operating systems, and it is distributed together with python, without the need for installing additional packages.

For the plot generation, the wide spread package Gnuplot was used. Gnuplot is a command-line driven, interactive function plotting program specially suited for scientific data representation. Since version 4.0 gnuplot can be used to plot functions and data points in both two and three dimensions and in many different formats. A special interface class has been developed to allow a two way communication through piping between FLAIR and gnuplot.

2.2 Concepts

FLAIR is operating with the concept of the "FLUKA projects". A FLAIR FLUKA project is a structure containing the following information:

- general project information like title, notes, override formatting options for the input file;
- links to the filenames for the default input, optional geometry files and executable;
- maintain a list of geometry debugging regions;
- links to auxiliary Fortran files and libraries for compiling a user FLUKA executable if necessary;
- list of runs. A project can contain multiple runs based on the same input file by overriding the default preprocessor defines, title, random seed, number of start particles, run cycles and executable file;
- list of output files and rules for merging the output of scoring cards;
- list of user defined plots, for Geometry, all `USRxxx` cards, `RESNUCLEi` and visualization of the input file information.

All the above information is stored in a text file with the extension *.FLAIR*, and is “editable” with the FLAIR program.

FLAIR is able to read and write all formats recognized by FLUKA, but internally it works always in the name-based format, and treats the input as a list of extended cards. The default format for saving is always fixed with names for the input and free with names for the geometry. The user can override the default exporting format by using the appropriate FLUKA cards like `FREE`, `FIXED`, `GLOBAL`, `GEOBEGIN`. Moreover there is the possibility to export to other Monte Carlo simulations programs like MCNP/X [14,15], MARS [16] and various plotting formats.

FLAIR takes care to reorganize the input file before saving to disk by moving all the “special” cards in the appropriate position in order to have a correct FLUKA input file. The reorganization of the input file taking into account all the preprocessor cards that might alter the flow of the input file in case they are displaced. Therefore FLAIR will try to move or clone the preprocessor cards in order to create a logically correct input file.

All FLUKA cards in FLAIR, are described by the extended input card to have a uniform treatment. Each extended card is composed by:

- comment lines, the ones preceding the card definition, as well the in-line comments;
- a tag, which corresponds to the FLUKA card name. With a few additions: regions are defined with the REGION card. All preprocessor cards have tag names like `#define`, `#undef`, `#if` ...
- a variable number of whats. Starting from what(0) which corresponds to the sdum, what(1)..what(6) are the same with the FLUKA whats, what(7)-what(12) correspond to the what(1)-what(6) of the first continuation line etc.
- Some cards carry additional information which is accessible as what(-1). It is used also to store long strings or multi-line information used by some cards, like `TITLE`, `GEOBEGIN`, `PLOTGEOM` or `REGION`
- state of the card can be either enabled or disabled. Since cards cannot be commented-out with FLAIR, the only way of excluding them from the input without deleting them is to disable. by inserting around the card an `#if 0`..`#endif` block. On the contrary all commented cards present in the input file, (with no space between the * and the card tag) will be converted to disabled cards.

All obsolete cards present in the input will be converted to the closest match if any, otherwise will be treated as errors. All unknown cards will be converted to the card `error` and be disabled.

FLUKA cards in FLAIR are grouped into categories for easier accessing and more flexible editing of the input file. Table I describes the category classification scheme used.

Table I. Card grouping

Category	**Description**
General	Cards of general purpose (like `TITLE, DEFAULTS, GLOBAL`, etc..)
Primary	Cards dealing with the definition of the primary starting particles
Geometry	Cards related to the definition of the geometry bodies/regions/lattices plotting and rotations/translations
Media	Cards for the definition of materials
Physics	Cards defining physics properties for the simulation
Transport	Cards that modify the way particles are transported in FLUKA
Biasing	Cards for importance biasing definition
Scoring	Cards related to scoring
Developers	Cards reserved by the developers
Preprocessor	Preprocessor definitions for creating conditional input files

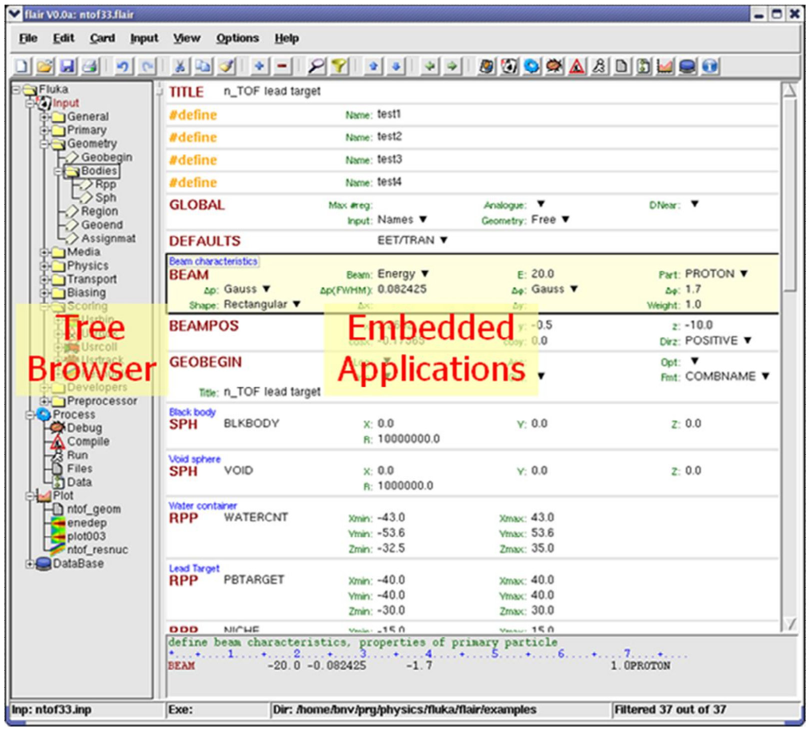
3 INTERFACE

FLAIR provides an all-in-one user friendly graphical Interface, with minimum requirements on additional software. When the application starts the main FLAIR window appears (Figure 1), which features the standard tools like menu bar, toolbar, status bar and navigation tree the main frame is a wrapper window that encapsulates all the project frames used for editing the information stored in the project file. The program is written in such way that all process frames could run as independent applications, even outside the main window of FLAIR. it reduces the development time, plus it allows the program to be modular for future enhancements, and also allow users to work in a batch mode. In the following sections you will find a short description of the main parts of the FLAIR program.

3.1 Input Editor

The input editor allows the user to create and edit a FLUKA input file. For this purpose it contains a fully featured editor like any other text editor, providing copy & paste, drag & drop, unlimited undo/redo functionality, error checking of the input provided by the user as well as more advanced features like geometrical optimizations and transformations. FLUKA is driven by input cards, and FLAIR is respecting this convention by displaying mini-dialogs (Figure 2) for each card. Each mini-dialog is a dynamically generated panel via the use of an internal database of all FLUKA cards. It displays the contents of each card in an interpreted human readable way. A lot of effort during the creation of the program was focused on the presentation of these mini-dialogs in order to provide a coherent and easy interface.

During editing of the input file FLAIR performs error checking and checks the card against logical errors that the user might introduce. The error items are highlighted with red and a popup dialog provides an explanation on the error. For the moment only logical errors on each value of a card are checked, while there is work going on to introduce error checking on correlated information among the cards.



```
TITLE      n_TOF lead target
#define                 Name: test1
#define                 Name: test2
#define                 Name: test3
#define                 Name: test4
GLOBAL                  Max #reg:              Analogue:  ▼          DNear:  ▼
                        Input: Names ▼         Geometry: Free ▼
DEFAULTS                EET/TRAN ▼
Beam characteristics
BEAM                    Beam: Energy ▼         E: 20.0               Part: PROTON ▼
          Δp: Gauss ▼   Δp(FWHM): 0.082425     Δφ: Gauss ▼           Δφ: 1.7
   Shape: Rectangular ▼ Δx:                    Δy:                   Weight: 1.0
BEAMPOS                                        y: -0.5               z: -10.0
                                               cosy: 0.0             Dirz: POSITIVE ▼
GEOBEGIN                                                             Opt:  ▼
                                                                     Fmt: COMBNAME ▼
          Title: n_TOF lead target
Black body
SPH        BLKBODY      X: 0.0                 Y: 0.0                Z: 0.0
                        R: 10000000.0
Void sphere
SPH        VOID         X: 0.0                 Y: 0.0                Z: 0.0
                        R: 1000000.0
Water container
RPP        WATERCNT     Xmin: -43.0            Xmax: 43.0
                        Ymin: -53.6            Ymax: 53.6
                        Zmin: -32.5            Zmax: 35.0
Lead Target
RPP        PBTARGET     Xmin: -40.0            Xmax: 40.0
                        Ymin: -40.0            Ymax: 40.0
                        Zmin: -30.0            Zmax: 30.0
RPP        NICHE        Xmin: -15.0            Xmax: 15.0

define beam characteristics, properties of primary particle
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
BEAM          -20.0 -0.082425     -1.7                                   1.0PROTON

Inp: ntof33.inp | Exe: | Dir: /home/bnv/prg/physics/fluka/flair/examples | Filtered 37 out of 37
```

Figure 1: FLAIR main application window

The input editor provides also means of filtering the cards in order to reduce the number of visible cards in the display or to group cards that are related by a specific property. By selecting the appropriate category or manually creating filters, FLAIR displays only the relevant cards and hides all the unwanted ones. One important feature of FLAIR is the filtering of linked cards, by a single click of the mouse the user can display only the cards that are linked by a specific property e.g. the same material name, or region etc. Higher level filters can be created by a specialized dialog permitting the user to create a filter based on a combination of rules and allowable ranges for various types.

During the setting up of a simulation calculation the user will spend most time on building the geometry. For this reason FLAIR provides several tools to facilitate this task, like automatic completion of the body names, optimization of the geometry by parenthesis expansion, removal of void terms, and geometrical transformation of bodies. By creating a series of body transformations (translation, rotation, scaling, mirroring) a specific portion of the geometry could be transformed in a single click.

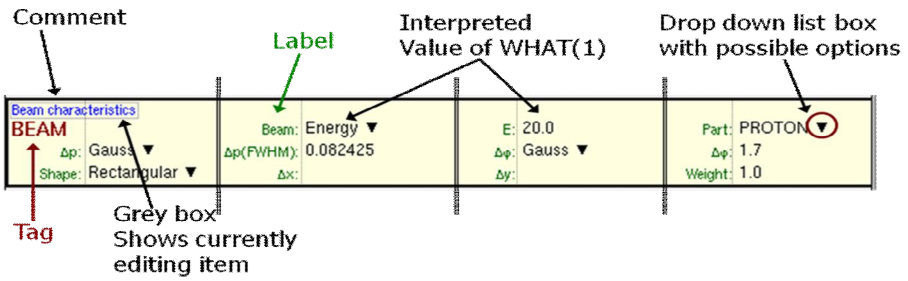
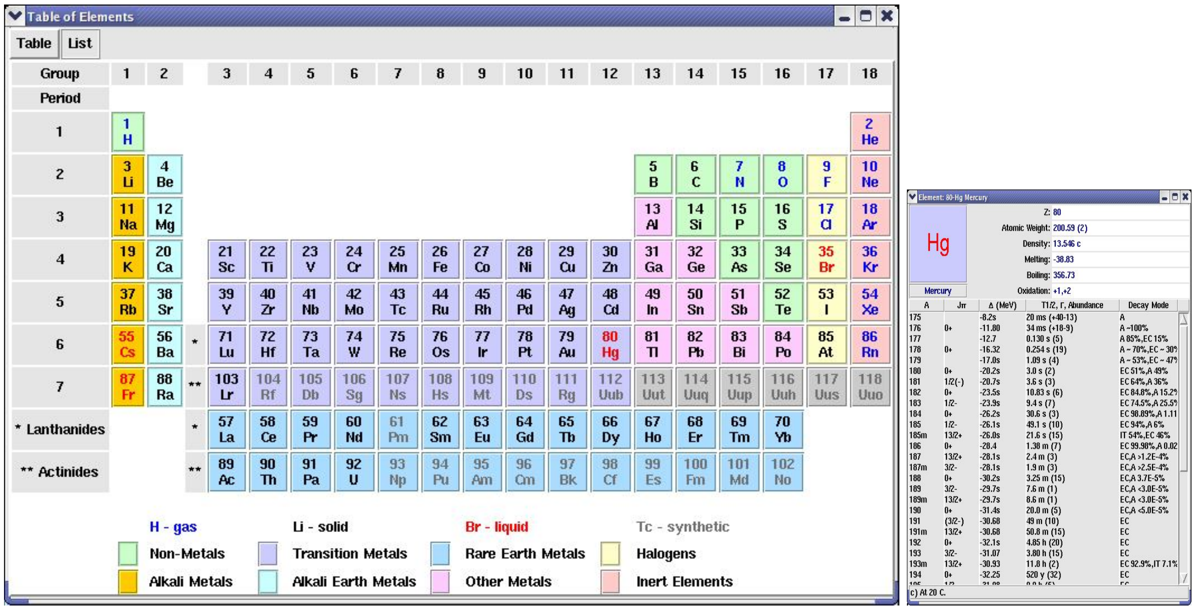


Figure 2: Example of a card mini-dialog during input editing

3.2 Isotope and Material Database

To facilitating the input editing FLAIR contains a version of the Nuclear Wallet Cards [17] (Figure 3) with the nuclear properties of all isotopes. It includes also a database of about 500 predefined materials with 300 of them with the Sternheimer [7] parameters and Displacement Per Atom (DPA) threshold energy. The user has the possibility to include any of the materials in the FLUKA input file and select which extra properties he will add. The program gives the ability to the user to edit and create his own database.



Group	1	2		3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Period																			
1	1 H																		2 He
2	3 Li	4 Be												5 B	6 C	7 N	8 O	9 F	10 Ne
3	11 Na	12 Mg												13 Al	14 Si	15 P	16 S	17 Cl	18 Ar
4	19 K	20 Ca		21 Sc	22 Ti	23 V	24 Cr	25 Mn	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	31 Ga	32 Ge	33 As	34 Se	35 Br	36 Kr
5	37 Rb	38 Sr		39 Y	40 Zr	41 Nb	42 Mo	43 Tc	44 Ru	45 Rh	46 Pd	47 Ag	48 Cd	49 In	50 Sn	51 Sb	52 Te	53 I	54 Xe
6	55 Cs	56 Ba	*	71 Lu	72 Hf	73 Ta	74 W	75 Re	76 Os	77 Ir	78 Pt	79 Au	80 Hg	81 Tl	82 Pb	83 Bi	84 Po	85 At	86 Rn
7	87 Fr	88 Ra	**	103 Lr	104 Rf	105 Db	106 Sg	107 Ns	108 Hs	109 Mt	110 Ds	111 Rg	112 Uub	113 Uut	114 Uuq	115 Uup	116 Uuh	117 Uus	118 Uuo
* Lanthanides			*	57 La	58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb		
** Actinides			**	89 Ac	90 Th	91 Pa	92 U	93 Np	94 Pu	95 Am	96 Cm	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No		

H - gas Li - solid Br - liquid Tc - synthetic

🟩	Non-Metals	🟪	Transition Metals	🟦	Rare Earth Metals	🟨	Halogens
🟧	Alkali Metals	🟦	Alkali Earth Metals	🟪	Other Metals	🟥	Inert Elements

Element: 80-Hg Mercury

Hg

Mercury

Z: 80
Atomic Weight: 200.59 (2)
Density: 13.546 c
Melting: -38.83
Boiling: 356.73
Oxidation: +1,+2

A	Jπ	Δ (MeV)	T1/2, Γ, Abundance	Decay Mode
175		-8.2s	20 ms (+40-13)	A
176	0+	-11.80	34 ms (+18-9)	A ~100%
177		-12.7	0.130 s (5)	A 85%,EC 15%
178	0+	-16.32	0.254 s (19)	A ~ 70%,EC ~ 30%
179		-17.0s	1.09 s (4)	A ~ 53%,EC ~ 47%
180	0+	-20.2s	3.0 s (2)	EC 51%,A 49%
181	1/2(-)	-20.7s	3.6 s (3)	EC 64%,A 36%
182	0+	-23.5s	10.83 s (6)	EC 84.8%,A 15.2%
183	1/2-	-23.9s	9.4 s (7)	EC 74.5%,A 25.5%
184	0+	-26.2s	30.6 s (3)	EC 98.89%,A 1.11
185	1/2-	-26.1s	49.1 s (10)	EC 94%,A 6%
185m	13/2+	-26.0s	21.6 s (15)	IT 54%,EC 46%
186	0+	-28.4	1.38 m (7)	EC 99.98%,A 0.02
187	13/2+	-28.1s	2.4 m (3)	EC,A >1.2E-4%
187m	3/2-	-28.1s	1.9 m (3)	EC,A >2.5E-4%
188	0+	-30.2s	3.25 m (15)	EC,A 3.7E-5%
189	3/2-	-29.7s	7.6 m (1)	EC,A <3.0E-5%
189m	13/2+	-29.7s	8.6 m (1)	EC,A <3.0E-5%
190	0+	-31.4s	20.0 m (5)	EC,A <5.0E-5%
191	(3/2-)	-30.68	49 m (10)	EC
191m	13/2+	-30.68	50.8 m (15)	EC
192	0+	-32.1s	4.85 h (20)	EC
193	3/2-	-31.07	3.80 h (15)	EC
193m	13/2+	-30.93	11.8 h (2)	EC 92.9%,IT 7.1%
194	0+	-32.25	520 y (32)	EC

c) At 20 C.

Figure 3: Nuclear Wallet Cards embedded in FLAIR

3.3 Compilation of the Executable

Under the interface titled “Compile” the user has the possibility to compile and create a FLUKA executable. FLAIR is checking the input file for cards that refer to user routines and gives the possibility to the user to include a copy of the prototype of these routines inside the project. By clicking the button “Build” FLAIR will try to compile all necessary modules in the same way like the Unix make program is doing, based on the dependency rules on the existence of a file and its time-stamp.

3.4 Geometry Debugging

FLAIR provides two possibilities of debugging the geometry of a FLUKA input file. One is the traditional way by activating the debugging with the special option of the `GEOEND` card, and another via the use of a specialized interface. In the Geometry debugging frame the user can define multiple regions to be debugged outside the input file. By activating the debugging FLAIR is creating temporary input file with the appropriate GEOEND card and submits a FLUKA job for each region to be debugged.

3.5 Running the Simulation

With the Run Frame the user has the ability to run the simulation or create several runs by overriding some parameters. Each run is identified by a unique input name and is based on the same input file. The parameters that can be changed are the title of the run, random number seed, starting particles, and to activate or deactivate preprocessor define cards. To increase the statistics or to benefit of a multi-core or cluster architecture, the user can submit many runs based on the same input by changing the random number history.

FLAIR is monitoring the progress on the run by inspecting the information on the files created by FLUKA, rather than using the process information. FLAIR performs a series of complicated tests based on the time elapsed, the information that is written by FLUKA. The method is quite reliable but not at 100% level, especially in very long simulations or in case that the simulation for some error situation is looping. However this method provides portability across various platforms and is working when the run is taking place on distrusted systems like Linux clusters, where any other method based on the CPU process information would have failed.

3.6 Inspecting the Files

FLAIR contains a file explorer that lists the files generated by FLUKA, grouped per FLUKA run, cycle and special ones like temporary files during the run. This frame gives the possibility to the user to inspect the output files even during the run and also to perform a clean up of the directory without losing the important files.

3.7 Data processing

To assist the user, FLAIR automatically detects all output files generated by FLUKA and their type from the input file definition and creates automatic rules for post processing these data. The user, simply by pressing a button starts the process and the data from all cycles are "merged" using the standard FLUKA post-processing routines. These rules can be fully customized by the user if required.

3.8 Plot Generation

FLAIR offers the possibility to the user to visualize in a graphical way the output of FLUKA (Figure 4). Currently with FLAIR one can plot the following:

- Geometry: 2D cross sections of the geometry either as boundary lines or color plots superimposed with materials, regions, lattices or magnetic field maps;
- Single differential quantities: through a common interface all single differential tallies like track length estimators, boundary crossing, collision or yield estimators. FLAIR allows to super impose many data on the same plot and even external files like experimental data for comparison;
- Double differential quantities: represented as a surface or color plots from estimators like boundary crossing or residual nuclei information;
- 3D mesh data: as two dimensional cross sections of 3D binning data, superimposed with the geometry, or as projection on a single axis;
- 3D photo realistic: the geometry can be exported to a ray tracer format, and photo realistic images can be generated with the use of Povray;

All plotted information could be normalized either by a fixed value or a user defined expression of the result. FLAIR is providing an interface to the user to define the desired plotting parameters which are sent to gnuplot to perform the actual plot. There are several text widgets for every plot frame that allow the user to send commands directly to the plotting program in order to allow to exploit the full power of gnuplot. To assist the user FLAIR provides also an automatic tool that scans the input file and proposes default plots for every scored quantity.

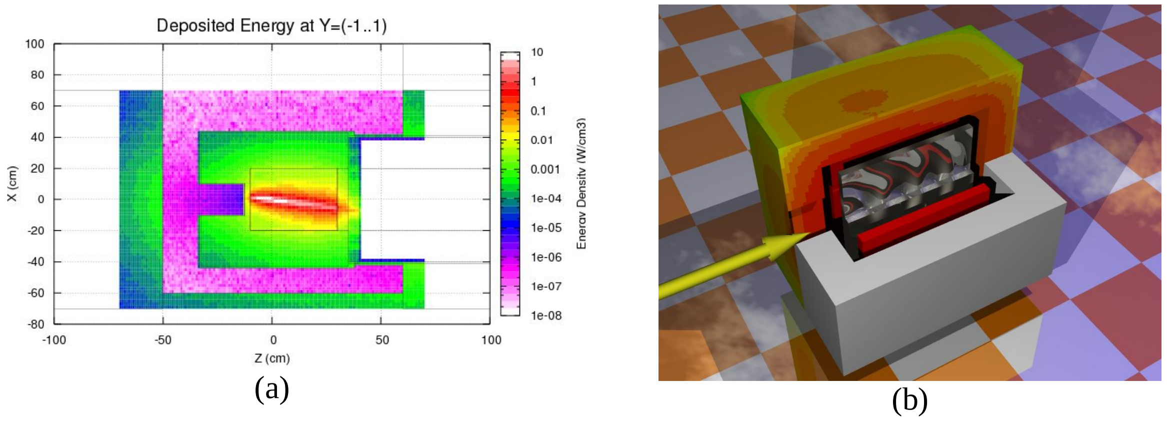


Figure 4: Examples of the plotting capabilities of FLAIR, (a) plot of USRBIN output on deposited energy density; (b) USRBIN output surface mapped on a 3D ray traced model

3.9 Help

A very useful feature of FLAIR is the hypertext FLUKA manual browser. With the press of F1 FLAIR displays the appropriate section of the FLUKA manual to the user. The first time FLAIR parses the complete ASCII version of the FLUKA manual and creates a python structure for faster accessing and searching.

4 EVOLUTION AND FUTURE

All FLUKA graphical interfaces before the appearance of FLAIR were made to perform precise tasks. Most of them were focused on the post processing of the USRBIN binning output, geometry, and just a couple of them were trying to address the problem of creating input files. There was not a single interface that was addressing all development stages during a simulation as well the post processing of the results in a unique package.

During the FLUKA course in the University of Houston, Texas in 2005, we had several users with quite different background and most of them with limited or almost no knowledge of Linux and text editing programs. A great fraction of the course time was spent in correcting syntactic errors, formatting problems as well basic text editing. With the increasing FLUKA community it was obvious that we needed to address the above problems to help the learning and use of the code. We needed an interface that allows the users to stay focused on their specific problem and the FLUKA implementation rather than wasting time in technicalities.

After investigating various concepts of such an interface, FLAIR was born in May 2006. During the first year it had a rapid development counting 50000 lines of code. It was introduced experimentally for the first time in the FLUKA course at Houston in 2007 and it was very well accepted by the FLUKA users. Since then it became the standard tool for teaching FLUKA in all courses and very quickly it became quite popular in the FLUKA community. The popularity can be seen by the number of downloads amounting to 300 every time a new version is released (Figure 5), and also the expressed interest of users via private communications with the author. The number of downloads has to be compared with the 3000 registered users of FLUKA over the past 10 years.

FLAIR has greatly evolved during the last two years and many new features have been introduced. There are several features to be added in the code, where the most important are:

i) a macro language for preprocessing, that will permit the user to create dynamic input files
ii) many more plotting features, like visual representation of the input file, plots of the USRDUMP and DETECT cards etc.;
iii) User databases for storing geometrical models;

During the last year, most of the effort was concentrated to make it more reliable. With the introduction in FLAIR of an automatic mechanism to record and transmit possible error messages and stack traces to the author via the web, many bugs have been fixed and now the code is quite stable and robust.

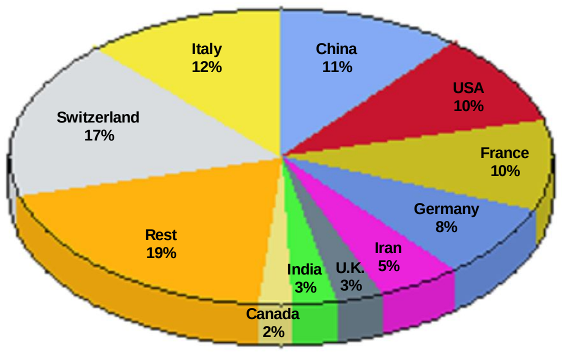


Figure 5: Country of origin of downloads

5 CONCLUSIONS

FLAIR is a user friendly graphical interface for FLUKA that offers both front-end and back-end interface to the user. The program is versatile, easy to use and provides in a single package all the necessary tools to the user to work with FLUKA. The program had a rapid development and quickly became very popular. FLAIR proved to be very robust and stable under the heavy test from beginners. It was successfully introduced as the default working environment in all FLUKA courses, after the positive and encouraging comments from the users. The experience in the FLUKA courses, showed that it greatly increased the learning curve of the new-users, since when using the interface they are focused on the problem they want to solve and not on practicalities.

6 ACKNOWLEDGMENTS

The author would like to thank Sylvestre Catin, Toni Empl, Adonai Herrera-Martinez, Marco Mauri, Lucia Sarchiapone, David Sinuela for their contribution to the FLAIR project. Alfredo Ferrari, Paola Sala, Markus Brugger, Francesco Cerutti, Stefan Roesler for their comments and the fruitful discussions. Also to thank the FLUKA collaboration for their support, and the users of the program for their comments and suggestions.

7 REFERENCES

1. A. Ferrari, P.R. Sala, A. Fassò, and J. Ranft, "FLUKA: a multi-particle transport code", CERN 2005-10 (2005), INFN/TC_05/11, SLAC-R-773
2. A. Fassò, A. Ferrari, J. Ranft, and P.R. Sala, "FLUKA: a multi-particle transport code", CERN-2005-10 (2005), INFN/TC_05/11, SLAC-R-773.
3. A. Ferrari, and P.R. Sala, "The Physics of High Energy Reactions", in Proceedings of Workshop on Nuclear Reaction Data and Nuclear Reactors Physics, Design and Safety, A. Gandini, G. Reffo eds., Trieste, Italy, April 1996, 2, 424 (1998).
4. V.Vlachoudis "FLUKA Advanced Graphical Interface", http://www.fluka.org/FLAIR
5. T. Williams, C. Kelley "GNUplot: an interactive plotting program" Version 3.5.
6. POV Team, "Persistency of Vision Ray Tracer (POV-Ray)", Version 3.0. Technical report. http://www.porray.org. 1998
7. R.M. Sternheimer, M.J. Berger, S.M. Seltzer "Density effect for the ionization loss of charged particles in various substances" At. Data Nucl. Data Tab. 30, 261-271 (1984)
8. R. Black, "Managing the Testing Process" (second ed), Wiley, New York (2002).
9. C. Kaner, J. Falk and H.Q. Nguyen, "Testing Computer Software" (second ed), Wiley, New York (1993).
10. C. Kaner, J. Bach and B. Pettichord, "Lessons Learned in Software Testing. A Context-Driven Approach", Wiley, New York (2001).
11. RedHat Inc. "RedHat Package Manager", http://www.rpm.org
12. M. Lutz, "Programming Python", O"Reilly & Associates, Sebastopol, Calif., 1996.
13. Fredrik Lundh. 1999. "An introduction to Tkinter". http://www.pythonware.com/library/tkinter/introduction/index.htm
14. Briesmeister, J. F., "MCNP - A general Monte Carlo N-particle transport code version 4A", LA-12625-M, 693 pp., Los Alamos Natl. Lab., Los Alamos, N.M., 1993.
15. L.S. Waters (Ed.), Los Alamos National Laboratory Report LA-UR 99-6058, Los Alamos, NM, USA, http://mcnpx.lanl.gov/, "MCNPX User's Manual", 1999.
16. N. V. Mokhov, "The MARS Code System User's Guide", version 13(95), Fermilab-FN-628 (1995).
17. Tuli, J. K., 1995, "Nuclear Wallet Cards", U.S. Nuclear Data Center.